VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
ELEKTRONINIŲ SISTEMŲ KATEDRA

Justinas Balinskas

VEIDO ATPAŽINIMO ALGORITMŲ TYRIMAS IR ĮGYVENDINIMAS OPERACINĖJE *ANDROID* SISTEMOJE

ANALYSIS OF FACE RECOGNITION ALGORITHMS AND IMPLEMENTATION IN *ANDROID* OPERATING SYSTEM

Magistro baigiamasis darbas

Informatikos inžinerijos studijų kryptis
Informacinių elektroninių sistemų studijų programa, valstybinis kodas 621E15003
Signalų apdorojimo sistemų specializacija

VILNIUS, 2012

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
ELEKTRONINIŲ SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas

_____
(parašas)

prof. habil. dr. R. Martavičius
2012 m. _____ mėn. ___ d.

Justinas Balinskas

VEIDO ATPAŽINIMO ALGORITMŲ TYRIMAS IR ĮGYVENDINIMAS OPERACINĖJE *ANDROID* SISTEMOJE

ANALYSIS OF FACE RECOGNITION ALGORITHMS AND IMPLEMENTATION IN *ANDROID* OPERATING SYSTEM

Magistro baigiamasis darbas

Informatikos inžinerijos studijų kryptis
Informacinių elektroninių sistemų studijų programa, valstybinis kodas 621E15003
Signalų apdorojimo sistemų specializacija

Vadovas doc. dr. Andrius Ušinskas _____ _____
(parašas) *(data)*

Lietuvių kalbos konsultantė Daiva Macko _____ _____
(parašas) *(data)*

VILNIUS, 2012

Vilniaus Gedimino technikos universiteto egzaminų,
sesijų ir baigiamųjų darbų rengimo bei gynimo
organizavimo tvarkos aprašo 2011-2012 m. m.
1 priedas

(Baigiamojo darbo sąžiningumo deklaracijos forma)

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Justinas Balinskas, 20065055

(Studento vardas ir pavardė, studento pažymėjimo Nr.)

Elektronikos fakultetas

(Fakultetas)

Informacinės elektroninės sistemos, SASfm-10

(Studijų programa, akademinė grupė)

BAIGIAMOJO DARBO (PROJEKTO) SĄŽININGUMO DEKLARACIJA

2012 m. gegužės 23 d.

Patvirtinu, kad mano baigiamasis darbas tema „Veido atpažinimo algoritmų tyrimas ir įgyvendinimas operacinėje "Android" sistemoje“ patvirtintas 2010 m. lapkričio 5 d. dekano potvarkiu Nr. 196el, yra savarankiškai parašytas. Šiame darbe pateikta medžiaga nėra plagijuota. Tiesiogiai ar netiesiogiai panaudotos kitų šaltinių citatos pažymėtos literatūros nuorodose.

Parenkant ir įvertinant medžiagą bei rengiant baigiamąjį darbą, mane konsultavo mokslininkai ir specialistai: Daiva Macko. Mano darbo vadovas doc. dr. Andrius Ušinskas.

Kitų asmenų indėlio į parengtą baigiamąjį darbą nėra. Jokių įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs (-usi).

	Justinas Balinskas
(Parašas)	(Vardas ir pavardė)

TURINYS

ILIUSTRACIJŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

Vilniaus Gedimino technikos universitetas
Elektronikos fakultetas
Elektroninių sistemų katedra

SBN ISSN
Egz. sk. .....
Data .....-.....-.....

Antrosios pakopos studijų **Informacinių elektroninių sistemų** inžinerijos programos baigiamasis darbas

Pavadinimas: **Veido atpažinimo algoritmų tyrimas ir įgyvendinimas operacinėje *Android* sistemoje**

Autorius **Justinas Balinskas**
Vadovas **doc. dr. Andrius Ušinskas**

Kalba: Lietuvių

Anotacija

Baigiamajame magistro darbe yra apžvelgti metodai, naudojami veidų atpažinimui bei išanalizuotas jų veikimas. Apžvelgus veidų atpažinimo metodus buvo pasirinkti trys algoritmai (tikrinių veidų, Fišerio veidų ir 2D–DCT+SOM), kurie išsamiai išanalizuoti ir įgyvendinti *MATLAB* aplinkoje bei ištirti įvairūs jų parametrai. Pagal gautus rezultatus buvo išrinktas optimalus algoritmas, tinkantis įgyvendinimui *Android* operacinėje sistemoje ir ten įgyvendintas. Baigiamajame darbe taip pat buvo apžvelgtos ir išanalizuotos problemos, su kuriomis susiduriama perkeliant algoritmą į *Android* operacinę sistemą, pateikti siūlymai algoritmo patobulinimui bei išvados. Visi užsibrėžti tikslai buvo pasiekti, o uždaviniai – išspręsti.

Veido atpažinimo algoritmų tyrimas ir įgyvendinimas operacinėje *Android* sistemoje. Magistro baigiamasis darbas informatikos inžinerijos laipsniui. Vilniaus Gedimino technikos universitetas. Vilnius, 2012, 185 p., 49 iliustr., 6 lent., 74 bibl., 6 priedai.

Prasminiai žodžiai

veidų atpažinimas, tikriniai veidai, *Android* operacinė sistema, veidų aptikimas, įterptinėje sistemoje

Vilnius Gediminas Technical University
Faculty of Electronics
Department of Electronic Systems

SBN ISSN
Copies No. .....
Date .....-.....-.....

Master Degree Studies **Informacinių elektroninių sistemų inžinerijos** study programme Final Thesis

Title:	**Analysis of face recognition algorithms and implementation in *Android* operating system**
Author	**Justinas Balinskas**
Academic supervisor	**Assoc Prof Dr Andrius Ušinskas**

Thesis language: Lithuanian

Annotation

The main goal of Master degree thesis is to review face recognition algorithms and analyze their performance. After this survey three face recognition algorithms (eigenfaces, fisherfaces and 2D–DCT+SOM) have been chosen for detailed analysis and investigation of their various parameters in *MATLAB* environment. According to the results obtained during this research only one algorithm, which is optimal for implementation in *Android* operating system, has been implemented on the mobile platform. This Master degree thesis also includes problems and suggestions regarding eigenface's algorithm implementation in *Android* operating system, proposals for algorithm improvement and detailed conclusions. All the objectives have been achieved and all problems – solved.

Analysis of face recognition algorithms and implementation in *Android* operating system. Master Thesis for Informatics Engineering degree. Vilnius Gediminas Technical University. Vilnius, 2012, 185 p., 49 figures, 6 tables, 74 references, 6 appendices.

Keywords

face recognition, eigenfaces, *Android* operating system, face detection, embeded system

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

ELEKTRONIKOS FAKULTETAS

ELEKTRONINIŲ SISTEMŲ KATEDRA

Technologijos mokslų sritis
Elektros ir elektronikos inžinerijos mokslo kryptis
Informatikos inžinerijos studijų kryptis
Informacinių elektroninių sistemų studijų programa, valst. kodas 621E15003
Signalų apdorojimo sistemų specializacija

TVIRTINU
Katedros vedėjas
prof. habil. dr. R. Martavičius
2012 m. 02 mėn. 6 d.

MAGISTRO BAIGIAMOJO DARBO UŽDUOTIS

Studentui Justinas BALINSKAS

Baigiamojo darbo tema: Veido atpažinimo algoritmų tyrimas ir įgyvendinimas operacinėje „Android" sistemoje

Analysis of face recognition algorithms and implementation in Android operating system

Patvirtinta 2012 m. 11 mėn. 05 d. dekano įsakymu Nr. 196 el

Baigiamojo darbo užbaigimo terminas 2012 m. 06 mėn. 1 d.

Darbo tikslas – ištirti tris pasirinktus veidų atpažinimo algoritmus ir geriausią jų įgyvendinti Android operacinėje sistemoje

Darbo uždaviniai

1. Atlikti veidų atpažinimo algoritmų apžvalgą
2. Ištirti pasirinktus veidų atpažinimo algoritmus
3. Pateikti pasirinkto algoritmo patobilinimo metodus
4. Įgyvendinti pasirinktą algoritmą Android operacinėje sistemoje

Aiškinamojo rašto turinys

1. Įvadas
   1.1. Darbo aktualumas ir tikslas
   1.2. Darbo uždaviniai
   1.3. Naudoti tyrimo ir analizės metodai
   1.4. Darbo naujumas ir praktinė nauda
   1.5. Darbo struktūra
2. Veidų atpažinimo algoritmų analitinė apžvalga
3. Optimalaus veidų atpažinimo algoritmo metodo analizė

4. Testavimo duomenų atranka

5. Veidų atpažinimo tyrimo rezultatai

5. Algoritmo įgyvendinimas operacinėje „Android“ sistemoje

6. Apibendrinimas. Išvados

Literatūra

Priedai

A priedas. Algoritmų schemos

B priedas. Pranešimo Lietuvos jaunųjų mokslininkų konferencijoje „Elektronika ir elektrotechnika 2012“ medžiaga

C priedas. Pasirinktų algoriktmų įgyvendintų aplinkoje MATLAB pirminis programos tekstas

D priedas. Pasirikto algoritmo įgyvendinto operacinėje „Android“ sistemoje pirminis programos tekstas

Vadovas (*parašas*) doc. dr. Andrius Ušinskas

(*mokslinis vardas ir laipsnis, vardas, pavardė*)

Užduotį gavau (*parašas*) 2012 m. _____ mėn. ____ d.

ĮVADAS. UŽDUOTIES ANALIZĖ

Kompiuterine rega – tai mokslo sritis, kuria susidomėjimas per pastarąjį dešimtmetį labai išaugo. Atsižvelgiant į tai, jog kompiuterinių skaičiavimų galia padvigubėja kas 18 mėnesių (Mūro dėsnis), veidų aptikimo ir atpažinimo uždaviniai iš paslaptingų pamažu tapo labai populiarūs ir aktualūs. Terminą veidų atpažinimas galima būtų apibrėžti kaip asmens ar asmenų identifikavimą arba verifikavimą iš statinio vaizdo ar video medžiagos, pasinaudojant sukaupta veidų duomenų baze.

Veidų atpažinimas yra turbūt pati natūraliausia biometrinė priemonė, nes žmonės šią užduotį sugeba atlikti ypač tiksliai. Neseniai atlikti tyrimai parodė, jog tam tikri veidų atpažinimo algoritmai jau pranoksta žmones atpažįstant veidus [8]. Kadangi veidų atpažinimo algoritmai yra vis plačiau pritaikomi įvairiose aplinkose, šiems algoritmams yra nuolat keliami didesni reikalavimai – norima, kad algoritmas veiktų sparčiau, tiksliau, būtų atsparus apšvietimo, veido išraiškos ir pozos variacijoms, ignoruotų žmogaus amžių ar veido plaukuotumą. Tačiau vertėtų nepamiršti, jog dažniausiai didžiausi veidų atpažinimo algoritmų patobulinimai įvyksta specialiose laboratorijose, kur vaizdo raiška ir aplinka yra kontroliuojami veiksniai, ko dažniausiai nebūtų galima pasakyti apie praktinio taikymo aplinkas.

Pačios populiariausios asmenų identifikavimo sistemos yra tos, kurios visiškai nereikalauja vartotojo įsikišimo. Būtent šiai kategorijai ir būtų galima priskirti veidų atpažinimą. Turbūt didžiausias veidų atpažinimo algoritmų trūkumas yra jų patikimumas, tačiau tai atperka algoritmo natūralumas, naudojimo lengvumas ir tai, jog jiems nereikia tiesioginio vartotojo įsikišimo. Automatinis asmens identifikavimas turi didžiulį potencialą ir begalę pritaikymo sričių, pradedant nuo draugų automatinio atpažinimo nuotraukų galerijoje iki ypač aukšto lygio saugumo sistemų, kuriose yra būtina identifikuoti asmenį.

Šio darbo tikslas yra apžvelgti ir išanalizuoti literatūrą susijusią su veidų atpažinimo algoritmais bei pasirinkti tris algoritmus, kurie bus detaliau ištirti. Ištyrus šiuos tris algoritmus, iš jų visų bus pasirinktas vienas algoritmas, kuris yra geriausias ir labiausiai tinkantis įgyvendinimui įterptinėje sistemoje. Geriausias algoritmas bus renkamas atsižvelgiant į spartą, tikslumą ir galimybes įgyvendinti įterptinėje sistemoje. Šis algoritmas su optimaliais parametrais bus įgyvendintas *Android* operacinėje sistemoje.

Tiriant veidų atpažinimo algoritmus bus išanalizuota, kaip tokie parametrai kaip paveikslų raiška bei jų kiekis turi įtakos algoritmų atpažinimo tikslumui, apsimokymo bei atpažinimo trukmei. Taip pat bus mėginama išsiaiškinti, kaip veidas turi būti pozicionuojamas nuotraukoje bei kokios turi būti iškerpamo veido proporcijos, jog veidų atpažinimo santykis būtų kuo aukštesnis. Įgyvendinus algoritmą *Android* operacinėje sistemoje bus išmatuota algoritmo sparta, ir jei pavyks – algoritmas bus pritaikytas vaizdo srauto apdorojimui realiu laiku.

Android operacinė sistema buvo pasirinkta atsižvelgiant į tai, jog šiuo metu visame pasaulyje sparčiai populiarėja išmanieji telefonai, kurie praktiškai kasdien tampa vis našesni ir sugeba atlikti vis sudėtingesnius skaičiavimus, o *Android* operacinė sistema šiai dienai yra tarp populiariausių visame pasaulyje. Išanalizavus programų, skirtų veidų atpažinimui pasiūlą šiai platformai buvo pastebėta, jog tokių programų yra labai nedaug. Tai gali būti dėl to, jog ši platforma egzistuoja palyginus neseniai, vos keletą metų, tad veidų atpažinimo niša joje dar neatrasta ir neišplėtota. Išmėginus jau egzistuojančias programas buvo pastebėta, jog dauguma jų veikia labai nestabiliai arba išvis neveikia, tad šiame darbe bus mėginama sukurti geresnę, tiksliai veikiančią veidų atpažinimo programą *Android* operacinėje sistemoje.

Kadangi išmanieji telefonai tampa neatsiejami nuo šeimininko visuose gyvenimo etapuose, įgyvendinus tokią programą jai galima sugalvoti daugybę praktinio panaudojimo sričių. Vienas iš pavyzdžiui galėtų būti automatinis nuotraukos susiejimas su egzistuojančiu kontaktu adresų knygelėje, ar net integravimas į socialinius tinklus. Susiejus veidų atpažinimo programą išmaniajame telefone su socialiniais tinklais, nuotraukose galėtų būti automatiškai identifikuojami draugai ir su jais būtų pasidalinama naujai užfiksuota akimirka. Kita pritaikymo sfera galėtų būti mobiliojo telefono apsauga, kas šiuo metu kiekvienam turėtų būti itin aktualu. Panaudojus veidų atpažinimo algoritmą būtų galima uždrausti nepageidaujamiems asmenims prieigą prie tam tikrų jautrių aplikacijų ar net pačio telefono draudžiant jo atrakinimą be atitinkamo asmens identifikavimo.

1. VEIDŲ ATPAŽINIMO ALGORITMŲ ANALITINĖ APŽVALGA

Veido atpažinimas – tai viena iš biometrinių technologijų, kuri lygina tam tikrų bruožų duomenis, kurie yra išskirti iš daugiau nei vieno veido (nesvarbu, ar tai statinis vaizdas, ar video medžiaga), su bruožų duomenimis, laikomais veidų duomenų bazėje [9].

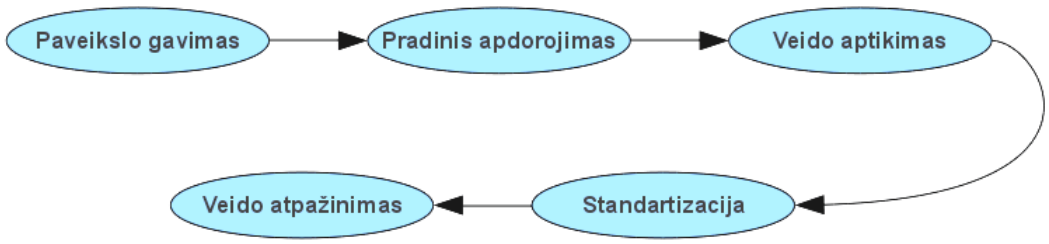
Veidų atpažinimas, ne taip kaip kitos biometrinės technologijos, yra žymiai draugiškesnis vartotojo atžvilgiu – šiai technologijai nereikalingas fizinis kontaktas ar specifinė įranga – jai užtenka paprasčiausios video kameros. Tačiau nevertėtų pamiršti ir pagrindinių šios technikos trūkumų – ji yra priklausoma nuo išorinių veiksnių, tokių kaip šviesa, triukšmas, veido išraiška, šukuosena, laikysena, poza ir t. t.

Kiekviena veidų atpažinimo sistema susideda iš 5 bendrinių etapų, pavaizduotų 1.1 paveiksle. Visų pirma iš video kameros yra gaunamas paveikslas, kuris turi būti išsaugotas atmintyje. Tada iš išsaugoto paveikslo yra pašalinamas triukšmas, kitaip sakant – jis yra paruošiamas veido aptikimo operacijai. Aptikus veidą, iš jo yra išskiriami mus dominantys bruožai, standartizuojamas šviesumas ir geometrija. Galiausiai paveikslo duomenys yra lyginami su duomenimis duomenų bazėje ir nustatoma asmens tapatybė. Tokios sistemos 1–4 etapai yra vadinami „aptikimo moduliu“, o paskutinysis – „atpažinimo moduliu“. Pirmasis modulis yra skirtas ne tik veido aptikimui nuotraukoje, tačiau ir pačios nuotraukos pradiniam apdorojimui ir paruošimui, kad atpažinimo modulis galėtų gauti kuo geresnius rezultatus.

1.1. Veidų aptikimas

Prieš pradedant kurti visiškai automatizuotą veidų atpažinimo sistemą, iš pradžių reikėtų įgyvendinti veidų aptikimo mechanizmą. Nuo šio mechanizmo priklauso tiek veidų atpažinimo kokybė, tiek ir visos sistemos veikimo tikslumas.

Veidų aptikimo algoritmuose dažniausiai yra naudojami statiniai vaizdai, tačiau derėtų pabrėžti, jog tarp statinių vaizdų ir reguliariais laiko intervalais gautos statinių vaizdų sekos esama skirtumų. Statiniuose vaizduose gan nesunkiai galima išskirti veidą iš fono



1.1 pav. Penkios bendrinės veidų atpažinimo sistemos stadijos

1.1 lentelė Veidų aptikimo algoritmų tipai

Tipas	**Bruožai**	**Veikimo charakteristikos**
Neuronų tinklas	Ieško išmokto veido formos juodai baltame statiniame paveiksle	Sugeba aptikti daugiau nei du veidus vienoje nuotraukoje, tačiau šis algoritmas yra lėtas ir jį sunku apmokyti darbui su veidais
Neuronų tinklas ir FFT	Yra naudojamas realiu laiku vienas po kito einantiems juodai baltiems kadrams pritaikant dažninius erdvinius algoritmus	Vidutiniškai sugeba aptikti 1–2 veidus realiu laiku. Jį yra labai sunku apmokyti
Neuronų tinklas ir neraiškioji logika	Neuronų tinkle vietoje pikselio ryškumo vertės yra naudojama neraiškiosios partnerystės funkcija	Gaunami geresni rezultatai nei paprasto neuronų tinklo, tačiau vaizdo apdorojimas trunka ilgiau
Normalizuotos RGB spalvos	Pagal tikimybių teoriją yra ieškoma didžiausio regiono, atitinkančio odos spalvą	Naudingas, kai reikia aptikti vieną veido regioną, tačiau įmanomas klaidos, susijusios su odos spalvos fonu
YIQ spalvos ir kiti paveikslai	Kombinuojama daugiau nei dviejų iš eilės einančių kadrų spalvinė ir judesio informacija	Sunku gauti kritinę vertę, kuri būtų nejautri raudonai spalvai ir yra gaunamas sumaišytas šviesos atspindys
Neraiškiosios spalvos	Modeliuojamos veido spalvos pagal neraiškios partnerystės (angl. *fuzzy membership*) funkciją	Labai paveiktas partnerystės funkcijų ir žinių bazės (angl. *knowledge base*)
PCA	Yra ieškoma paveikslo, kuris yra panašus į veidą lyginant su tinkamu veidu išreikštu vektoriumi	Dažniau yra naudojamas kaip algoritmas, skirtas veido charakteringųjų taškų išskyrimui, o ne veidų aptikimui
Algoritmo šablonas	Veidas yra aptinkamas skaičiuojant koreliaciją tarp veido geometrijos šablono ir paveikslo	Sunkiai pasiduoda įvairiems veido formos ar dydžio pokyčiams

pagal spalvas arba veido šabloną, tačiau jei fonas yra sudėtingas, pavyzdžiui, margas, tuomet tai padaryti tampa žymiai sudėtingiau. Tuo tarpu kintančiame vaizde veidą galima aptikti pagal judesio informaciją kintančiuose kadruose.

1.1 lentelėje yra pateikti populiariausi metodai, kurie yra naudojami veidų aptikimui vaizduose. Pagrindinis visų šių algoritmų tikslas yra surasti veidą, pasinaudojant įvairiausiais vaizdų apdorojimo metodais, tad galima teigti jog nei vienas algoritmas nėra geresnis už kitus. Dažniausiai siekiant optimizuoti galutinius rezultatus keli algoritmai yra apjungiami į vieną visumą, pavyzdžiui judesio, spalvos, šablonų ir dirbtinio intelekto.

Dauguma algoritmų, kurie išskiria veidą pagal spalvas ar judesį, gali kartu su veidu išskirti ir dalį fono, arba atvirkščiai – nukirpti dalį veido. Siekiant išvengti tokių netikslumų yra naudojama geometrinė paveikslų standartizacija, kuri ištaiso šias klaidas išskirdama iš nuotraukos tam tikrus bruožus (akių, nosies, lūpų ir t. t. poziciją) ir pagal jų dydį bei poziciją nusprendžia, ar reikia gautą vaizdą koreguoti geometriškai.

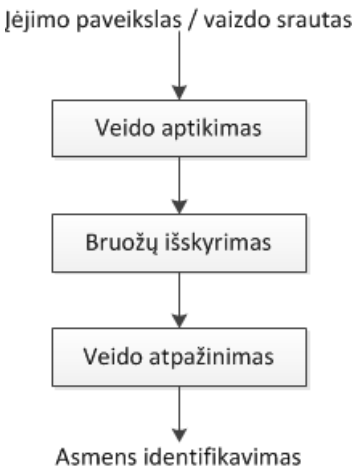
1.2 lentelė Pagrindiniai veidų atpažinimo algoritmai

Tipas	**Bruožai**	**Veikimo charakteristikos**
Geometrinis modelis	Tapatybė yra tikrinama lyginant geometrinių bruožų taškus. Kiekvienas veidas prieš tai turi būti standartizuojamas, o pagrindinių bruožų taškų pozicijos yra labai svarbios	Kadangi veidas yra trimatis bei kinta jo išraiškos, šis algoritmas turi neišvengiamų apribojimų
PCA	Šviesūs ir tamsūs plokščių paveikslų raštai (angl. *patterns*) yra laikomi kaip vienas vektorius tariant, jog veido paveikslas yra tokių vektorių rinkinys	Galimos klaidos, kai pakinta veido ryškumas ar pozicija
Tikimybinis modelis	FLD, EFM, SVM ir t. t.	Šie algoritmai siekia pagerinti savo tikslumą išnaudodami PCA trūkumus
Neuronų tinklai	Sukuriamas daugiasluoksnis perceptronas ir apmokomas dirbti su vaizdais	Algoritmo apmokymas ir apmokymo rinkinio sudarymas yra sudėtingas
Vilnelės ir elastingumo tikrinimas	Išnaudojamas veido pozicijos bei išraiškos pokytis keičiant dažnį	Algoritmo apimtis yra per didelė lyginant su atpažinimo santykiu

1.2. Veidų atpažinimas

Veidų atpažinimo algoritmai veikia lygindami standartizuotą įėjimo veidą su visais veidais, esančiais duomenų bazėje. Plačiausiai visame pasaulyje paplitę veidų atpažinimo algoritmai yra pateikti 1.2 lentelėje. Pagrindiniai iššūkiai siekiant atpažinti veidus yra: 1) surasti ir išskirti tokius veido bruožus, kurie yra pagrindiniai veido apibūdinime ir 2) nuspręsti, kaip būtų galima suklasifikuoti veidus pagal pasirinktus bruožus. Veidams atpažinti egzistuoja daugybė metodų, kurie gali būti suklasifikuoti į šias pagrindines grupes: pagrindinių komponentų analizė (angl. *Principal Componenets Analysis, PCA*), Fišerio tiesinis diskriminantas (angl. *Fisher's Linear Discriminant, FLD*), nepriklausomos komponentės (angl. *Independent Components*), diskriminaciniai bendrieji vektoriai (angl. *Discriminative Common Vectors*) ir Laplaso veidai (angl. *Laplacianfaces*).

Tokios technikos kaip PCA ir FLD veidų atvaizdus apdoroja kaip vektorius daugiamatėje erdvėje ir iš jų išveda mažesnių matmenų atitikmenį (kai naudojama PCA) arba diskriminacinį atvaizdą (kai naudojama FLD). FLD metodas veikia tiksliau lyginant su metodais, pagrįstais pagal tam tikrus veido bruožus, tačiau jį įgyvendinant yra atliekama daugiau ir sudėtingesnių skaičiavimų. Diskriminacinių bendrųjų vektorių metodas yra spartesnis, nes iš viso duomenų masyvo bendrinis vektorius yra parenkamas kiekvienai klasei atskirai, o ne keliems veidams kiekvienoje klasėje. Laplaso veidų metodas labiausiai skiriasi nuo PCA ir FLD, nes jo pagrindinis tikslas yra lokalios informacijos kaupimas, todėl jis gali aptikti esminę daugialypę struktūrą.



1.2 pav. Trys pagrindinės veidų atpažinimo algoritmo dalys

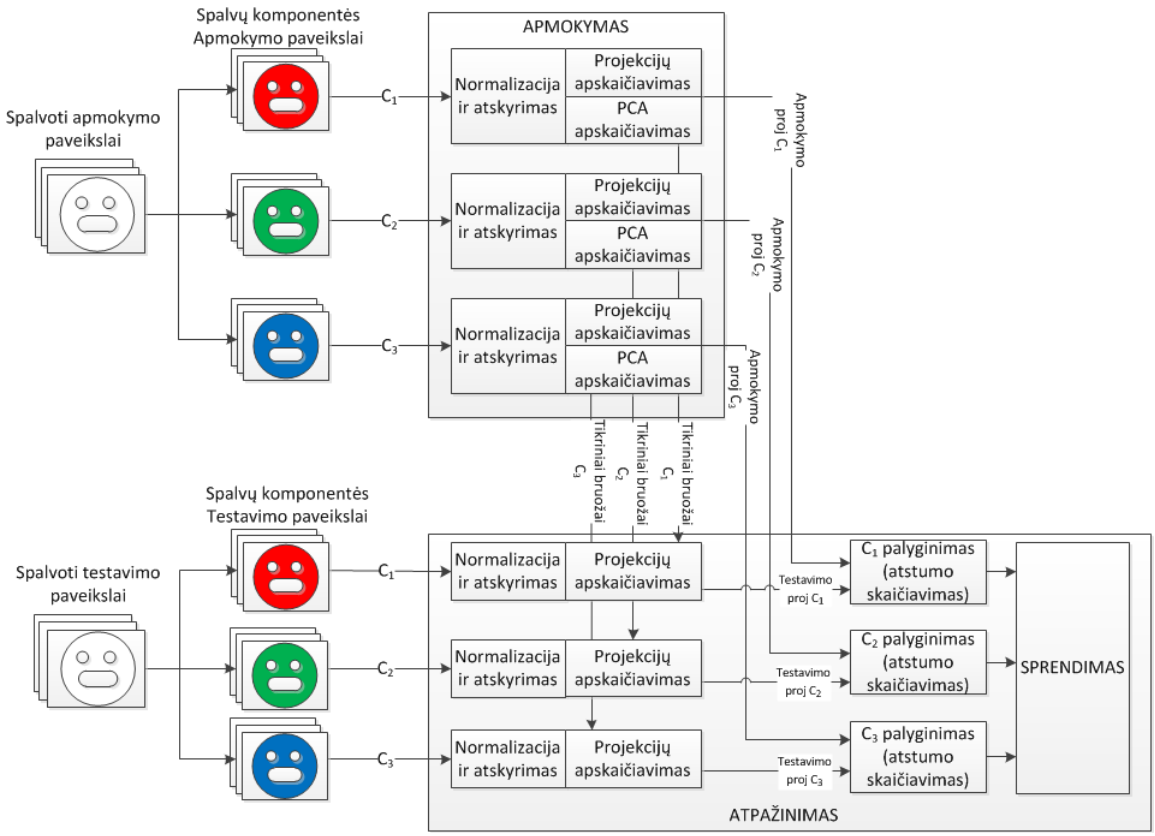
Veidų atpažinimo taikomosios programos gali būti suskirstytos į tris grupes:

1. Veidų atpažinimas protingose aplinkose: tokių programų pagrindinė užduotis – atpažinti veidus nekintančiame fone, pavyzdžiui identifikuoti žmones, įeinančius į „protingą namą“. Tai, dažniausiai, turi būti atliekama nepriklausomai nuo apšvietimo ir galvos pozicijos.
2. Veido atpažinimas protingose mašinose: šiose programų grupėse mašina turi identifikuoti asmenį, su kuriuo ji sąveikauja, pavyzdžiui – automobilis identifikuoja vairuotoją arba kompiuteris identifikuoja naudotoją. Šiais atvejais galvos pozicija praktiškai išvis nekinta arba kinta labai siaurame diapazone (vairuotojas žiūri į kelią, kompiuterio naudotojas – į ekraną), o aplinka už veido, kuris yra atpažįstamas, gali kisti nepriklausomai.
3. Veidų atpažinimas foto–nuotraukose ar video medžiagoje: šioje grupėje veidų atpažinimas yra naudojamas kaip paieškos priemonė objektams aptikti. Tai yra pats sudėtingiausias atvejis, nes visos sąlygos yra nepriklausomos ir nekontroliuojamos [10].

Kiekvienas veido atpažinimo algoritmas susideda iš trijų pagrindinių dalių: 1) veido aptikimo, 2) bruožų išskyrimo ir 3) veido atpažinimo, kaip pavaizduota 1.2 paveiksle [11]. Algoritmas, turintis visas šias tris dalis, yra laikomas pilnai automatizuotu, o turintis ne visas dalis (dažniausiai tik 2 ir 3 dalis) – pusiau automatizuotu algoritmu.

Vertėtų paminėti, jog veidų atpažinimas – tai plati sąvoka, kuri apima:

- identifikavimą – turi būti nustatoma etiketė, atitinkanti tą žmogų;
- atpažinimą – turi būti nuspręsta, ar žmogus jau buvo matytas, ar yra matomas pirmą kartą;
- kategorizavimą – kai veidas turi būti priskirtas kuriai nors egzistuojančiai klasei pagal tam tikrus bruožus.



1.3 pav. Spalvotų paveikslų atpažinimo sistema

Ashish Tiwari atlikto tyrimo metu buvo išskirti duomenų klasifikavimo metodai, labiausiai tinkantys skirtingoms klasėms [12]. Pavyzdžiui, norint suklasifikuoti žmones pagal rasę ar amžių, geriausiai tinka dirbtinis neuronų tinklas (DNT), o klasifikuojant pagal lytį – Parzeno langas arba K – artimiausio kaimyno metodai.

Vienas iš svarbiausių faktorių, lemiantis teisingą veido atpažinimą – veido pozicija. Kaip galima pastebėti, ji taip pat nulemia veidų atpažinimo aplikacijų klasifikavimą. Kai veido nuokrypis nuo kameros yra didesnis nei 30°, yra pastebimas ryškus atpažinimo santykio mažėjimas [13, 14]. Norint išlaikyti pakankamai aukštą atpažinimo santykį, veidas gali būti nukrypęs daugiausiai 15° į kairę arba į dešinę pusę.

1.3. Spalvų įtaka veidų atpažinimui

Spalva, kaip bruožas, apibūdinantis veidą, nors ir atrodo, turėtų būti vienas iš pagrindinių bruožų, atpažįstamų veide, vis dar yra ginčytinas klausimas literatūroje. Daugumoje įgyvendintų veidų atpažinimo modelių bei tyrimų yra pagrįsti nespalvotais paveikslais.

Tyrimo metu B. Karimi ir A. Krzyzak pastebėjo, jog spalvos reikšmė yra labai svarbus bruožas atpažįstant veidus [15]. Šių studijų rezultatai rodo, jog veidų atpažinimas spalvotuose paveiksluose yra žymiai tikslesnis, lyginant su pilkšvais paveikslais.

Nors dauguma veidų atpažinimo sistemų veikia su pilkšvais paveikslais, spalvos šioms

1.3 lentelė Veidų atpažinimo santykis spalvotiems ir nespalvotiems paveikslams naudojant skirtingus metodus

Naudojamas metodas	**Atpažinimo santykis nespalvotiems paveikslams**	**Atpažinimo santykis spalvotiems paveikslams**
PCA	0,73	0,80
FLD	0,80	0,93
Diskriminaciniai bendriniai vektoriai	0,93	1,00
Laplaso veidai	0,93	0,93
PCA + Gaboro filtrai	0,86	0,96
FLD + Gaboro filtrai	0,86	0,93

sistemoms gali turėti esminės įtakos, padedančios geriau atskirti apibūdinančiuosius bruožus. Tokiose veidų atpažinimo sistemose, norint įterpti spalvinius bruožus, reikia įgyvendinti papildomą modulį, pavaizduotą 1.3 paveiksle.

Tokioje sistemoje paveikslų apdorojimas yra atliekamas dviem etapais: 1) apmokymu ir 2) atpažinimu. Pirmas žingsnis kiekviename etape yra suskaidyti spalvotą paveikslą į tris skirtingus paveikslus, kurie apibūdintų skirtingas spalvų plokštumas. Pirmasis paveikslas atitiktų raudonos spalvos plokštumą, antrasis – žalios, o trečiasis – mėlynos. Kitame apmokymo žingsnyje kiekvienas iš trijų paveikslų yra teikiamas veidų atpažinimo algoritmui kaip nespalvotas paveikslas, kad būtų apskaičiuoti reikiamų komponenčių koeficientai. Kai visi koeficientai yra apskaičiuoti ir išsaugoti, apmokymo etapas yra baigtas.

Veidų atpažinimo etape procedūra yra identiška apmokymo procedūrai, tik pačioje pabaigoje yra apskaičiuojamas atstumas tarp trijų paveikslų (raudono, žalio ir mėlyno), o rezultatas yra teikiamas komponentei, kuri parenka atsakymą. Globalus atstumas tarp testuojamo ir apmokymo paveikslo yra skirtingų atstumų svertinė suma, taip, kad kiekvieno eigen-bruožo indėlis atpažinimo etape būtų vienodas ir nediskriminuotų nei vienos spalvų komponentės. Žinoma, tam tikrų bruožų svertiniai faktoriai gali būti pakeisti norint juos labiau pabrėžti arba suteikti didesnį svorį kuriai nors spalvų plokštumai.

Toks vaizdų apdorojimas skirtinguose spalvų plokštumose nulemia geresnius rezultatus veidų atpažinime. Tai galima matyti iš gautų rezultatų, kurie yra pateikti 1.3 lentelėje.

Iš šių rezultatų galima padaryti išvadą, jog spalvos yra reikšmingas faktorius veidų atpažinimui. Spalvotų paveikslų pritaikymas nulemia geresnius rezultatus net ir tuose algoritmuose, kurių veidų atpažinimo santykis buvo ir taip aukštas, o tradiciniuose veidų atpažinimo metoduose (tokiuose kaip PCA ir FLD), kurių atpažinimo santykis nėra labai aukštas – pastebimas labai ryškus pagerėjimas. Nors literatūroje yra teigiama, jog spalvų neskiriantys žmonės sugeba veidus atpažinti taip pat tiksliai kaip ir žmonės, skiriantys spalvas [16], šio tyrimo rezultatai tam prieštarauja ir įrodo, jog spalvų buvimas turi esminį pranašumą.

1.4. Veidų atpažinimas nespalvotuose paveiksluose

Literatūroje yra daug informacijos apie veido spalvos panaudojimą jos segmentacijai, tačiau lyginant segmentuotus veidus dažniausiai atsisakoma spalvinės informacijos ir lyginamos veidų tekstūros, pateiktos pilkumo lygmenimis. Pastebėta, kad baltaodžių ir geltonodžių veiduose dominuoja raudonoji ir žalioji komponentės, o mėlynojoje būna santykinai daug triukšmo. Todėl perėjimo prie pilkumo lygmenų formulėje

$$grey = aR + bG + cB \tag{1.1}$$

rekomenduojama naudoti svorius $a = 0{,}5$, $b = 0{,}5$ ir $c = 0$. Skrupulingai optimizuojant veidų atpažinimo kokybę galima parinkti ir kitus svorius.

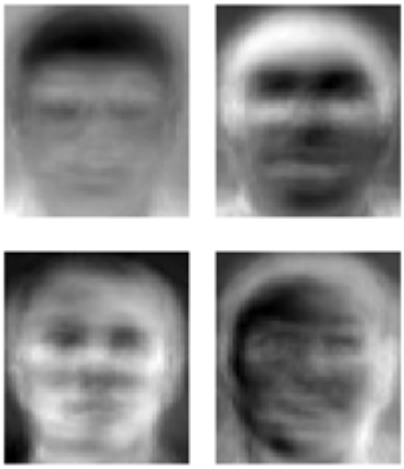
Praktiškai pastebėta, kad, skaičiuojant veidų požymius tiesiogiai naudojantis veido pilkumo lygmenų vaizdu, požymiai tampa pakankamai nestabilūs. Pagrindinis požymių nestabilumo šaltinis yra veido apšvietimas. Kintant apšvietimo šaltinio spektrui ir pozicijai gaunami skirtingų pilkumo lygmenų vaizdai, o tai kenkia atpažinimo kokybei. Norint eliminuoti apšvietimo poveikį yra atliekamas taip vadinamas veido foto–metrinis arba pilkumo lygmenų normalizavimas. Normalizavimo pagrindinė idėja – išnaudoti santykinai lėtą apšvietimo stiprio kitimą gretimuose pikseliuose. Praktiškai normalizavimas yra atliekama tokiu būdu:

1. Atliekamas pilkumo lygmenų vaizdo vidurkinimas σ_1 masteliu;

2. Atliekamas pilkumo lygmenų vaizdo vidurkinimas $\sigma_2 > \sigma_1$ masteliu;

3. Normalizuotas vaizdas apibrėžiamas dviejų vidurkintų vaizdų santykiu arba skirtumu.

Norint realizuoti šią procedūrą reikia pasirinkti vaizdo vidurkinimo būdą. Teoriškai tam tikra prasme optimalus vidurkinimas naudoja Gauso filtrą. Gauso funkcijos vidutinio kvadratinio nuokrypio dydis σ apibrėžia vidurkinimo mastelį. Gauso vidurkinimas yra invariantiškas vaizdo posūkiui, tačiau vidurkinimo procedūra yra lėta. Vidurkinimas sparčiau atliekamas taikant pasirinkto dydžio kvadrato formos slenkamąjį vidurkinimo langą, t. y. vidurkinimo filtras yra lygus 1, kai indeksai patenka į 2σ dydžio kvadratą, kurio vidurys yra taške $(0, 0)$, kitais atvejais filtro reikšmė yra nulis. Tačiau vidurkinimo rezultatas panaudojant šį filtrą gana jautrus vaizdo posūkiui ir poslinkiui, nes filtras yra trūkus, t. y. jo reikšmės šuoliu pereina nuo 0 iki 1 ir atvirkščiai.



1.4 pav. AT&T laboratorijoje gautų tikrinių veidų pavyzdžiai

1.5. Veidų požymių išskyrimas

Veidų atpažinimo algoritmus galima suskirstyti į dvi stambias kategorijas: globalių (angl. *holistic*) ir lokalių (angl. *local*) požymių lyginimo algoritmus. 1987 metais pasirodė tikrinio veido (angl. *eigenface*) metodas [17], kuris iki šiol yra populiarus. Tikrinių veidų metodas yra pagrindinių komponenčių metodo (PCA) dalinis atvejis. Šio metodo viena pirmųjų realizacijų [18] atskleidė metodikos trūkumus ir dėl prastų veido atpažinimo rezultatų yra naudojama dažniausiai kaip apatinis atramos taškas kitų algoritmų rezultatams palyginti. Klasikinis tikrinių veidų algoritmas naudoja globalius požymius.

1.4 paveikslas iliustruoja kelis pirmuosius tikrinius veidus. Tikriniai veidai surandami vidurkinant turimą geometriškai normalizuotų veidų paveikslėlių kolekciją ir apskaičiuojant standartinio dydžio veidų kovariacinę matricą. Toliau yra randamos kovariacinės matricos tikrinės reikšmės ir tikriniai vektoriai, kurie ir yra vadinami tikriniais veidais. Tikriniai veidai surūšiuojami tikrinių reikšmių mažėjimo tvarka, todėl tikriniai veidai su mažiausiais indeksais atspindi charakteringiausius (dažniausiai pasitaikančius) veidų nuokrypius nuo suvidurkinto veido.

LBP (angl. *Local Binary Patterns*) – tai dar vienas požymių išskyrimo metodas, kuris buvo sukurtas palyginus neseniai (2004 metais). Dėl savo paprastumo ir santykinai neblogų rezultatų LBP metodas buvo pritaikytas veidų atpažinimui ir labai išpopuliarėjo. LBP reikšmė priklauso nuo pasirinkto centrinio taškelio ir jo aplinkos. 1.5 paveikslas iliustruoja LBP reikšmės apskaičiavimą viename taške. Kaip matyti iš iliustracijos, LBP

1.5 pav. LBP reikšmės apskaičiavimas

1.6 pav. Veidas, suskaidytas į lokalias sritis LBP histogramoms skaičiuoti [1]

reikšmė priklauso tik nuo santykinio pilkumo lygmenų didumo. Tai reiškia, kad, atlikus bet kokią monotoninę vaizdo pilkumo lygmenų transformaciją, transformuotam vaizdui gausime tas pačias LBP reikšmes. Ši savybė rodo LBP požymių atsparumą ir paprastumą, kas yra pagrindiniai šios metodikos privalumai.

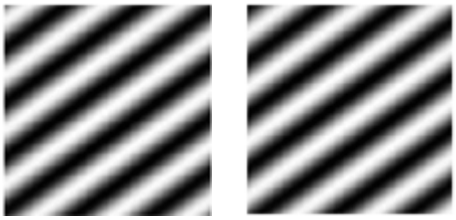
Taikant LBP požymius veidų atpažinimui, veidas yra padalinamas į mažas lokalias sritis, kurioms apskaičiuojamos LBP reikšmių histogramos. Skirtingų veidų LBP reikšmių histogramos yra lyginamos ir jų panašumas atspindi lyginamų veidų panašumą.

Veido suskaidymas į lokalias sritis dar kartą atspindi tą faktą, kad globalūs požymiai nelabai kokybiškai aprašo patį veidą ir todėl yra pereinama prie lokalesnių požymių. Tačiau palyginimui naudoti pavienes LBP reikšmes būtų keblu skaičiavimo laiko prasme ir iškiltų problemos nustatyti tiksliais poslinkio reikšmes, kurios garantuotų lyginamų autentiškų veidų panašumą. Lokalių sričių LBP histogramos yra kompromisinis variantas (1.6 pav.), kurias naudojant veidų palyginimas atliekamas greitai ir toleruojami nedideli lyginamų autentiškų veidų poslinkiai. Matematinės statistikos terminais histogramos atspindi LBP reikšmių skirstinį, todėl lyginant histogramas galima naudotis matematinės statistikos metodais.

LBP histogramos jautriai reaguoja į įvairius apšvietimo pasikeitimus, kartu ir į triukšmą. Charakteringos veido paveikslėlio zonos, kuriose yra santykinai daug triukšmo, yra kakta ir skruostai, todėl kartais lyginant naudojami skirtingi svoriai skirtingų LBP histogramų sritims.

Dar vienas lokalius požymius analizuojantis metodas yra Gaboro lokalūs požymiai ir tūtos. Šie požymiai yra gaunami filtruojant veidą įvairių parametrų Gaboro filtrais, kurie yra taikomi praktiškai visuose veidų atpažinimo sistemose.

Gaboro filtrai yra Gauso (angl. *Gaussian*) ir kosinuso bei sinuso funkcijų sandaugos. Kadangi filtrai taikomi vaizdams, tai visos funkcijos yra dviejų kintamųjų x ir y. Gauso funkcijos reikšmės priklauso tik nuo taško (x, y) atstumo iki koordinačių pradžios ir turi vieną laisvą parametrą σ, kuris apibrėžia funkcijos mastelį. Sinusas ir kosinusas dažniausiai apjungiami į vieną kompleksinę eksponentę ir turi du laisvus parametrus: krypties



a) *b)*

1.7 pav. Gaboro filtro kompleksinės eksponentės fiksuotos krypties ir dažnio *a)* realioji ir *b)* menamoji komponentės

vektorių ir dažnį. Matematiškai Gaboro filtras apibrėžiamas tokia išraiška:

$$G(x,y) = s(x,y) \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right), \tag{1.2}$$

čia $s(x,y)$ yra kompleksinė eksponentė, kurios realioji ir menamoji komponentės skaičiuojamos pagal formules:

$$\operatorname{Re} s(x,y) = \cos(2\pi f(x\cos\alpha + y\sin\alpha)), \tag{1.3}$$

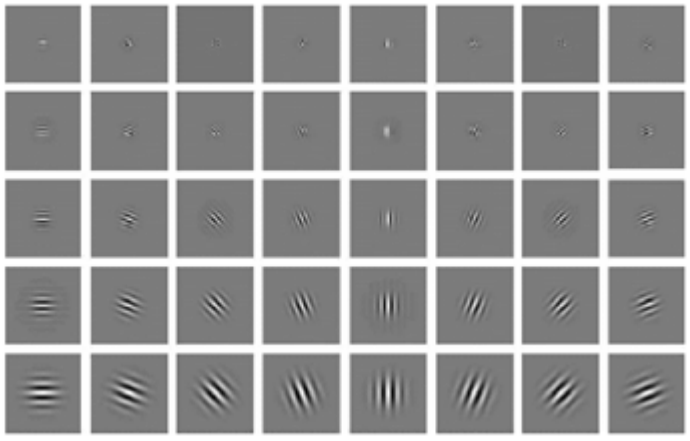
$$\operatorname{Im} s(x,y) = \sin(2\pi f(x\cos\alpha + y\sin\alpha)), \tag{1.4}$$

čia α filtro krypties, o f – dažnio parametrai. 1.7 paveikslas iliustruoja $\operatorname{Re} s(x,y)$ ir $\operatorname{Im} s(x,y)$ Gaboro filtro komponentes.

Jei $u(x,y)$ yra paveikslėlio pilkumo lygmenų intensyvumo funkcija, tai vaizdo Gaboro atsaku taške (x,y) vadinama sąsūka:

$$(u * G)(x,y) = \iint_{-\infty}^{+\infty} u(x-\xi, y-\eta) G(\xi,\eta) \mathrm{d}\xi \mathrm{d}\eta. \tag{1.5}$$

Daugumoje veidų atpažinimo sistemų, naudojančių Gaboro filtrus, naudojami aštuo-



1.8 pav. Aštuonių krypčių ir penkių mastelių Gaboro filtrų kompleksas

nių orientacijų α ir penkių mastelių σ Gaboro atsakai. 1.8 paveikslas iliustruoja šių krypčių ir mastelių filtrus.

Geometriškai ir fotometriškai normalizuoto veido srityje atliekant sąsūkos operaciją su Gaboro filtrų kompleksu kiekviename taške yra gaunamas 8×5 požymių rinkinys, kuris vadinamas Gaboro tūta (angl. *Gabor jet*). Gaboro tūta charakterizuoja veido sritį, esančią apie centrinį Gaboro filtro tašką (tašką, kuriame eksponentinė Gauso komponentė maksimali). Veidų panašumas apskaičiuojamas lyginant kiekvieno veido gautas tūtas. Yra patikrinta praktiškai, kad Gaboro požymiai yra atsparūs apšvietimo kitimui ir mažam veido nuokrypiui, atsiradusiam po geometrinio normalizavimo. Jei veidui netaikomas foto–metrinis normalizavimas ir lyginami veidai yra fotografuoti skirtingo apšvietimo sąlygomis, tai Gaboro tūta tik dalinai atspari apšvietimo trukdžiams ir naudojant tokius požymius kiek pablogėja veidų atpažinimo rezultatai.

Pagrindinis Gaboro požymių trūkumas – jie lėtai apskaičiuojami. Todėl dažnai Gaboro požymiai įvertinami tik išretintame centrų tinklelyje, o tai menkina jų gebėjimą atskirti kiek paslinktus autentiškus veidus. Anksčiau aprašytus vaizdų vidurkinimo simetrinius eksponentinius filtrus galima apibendrinti įvedant dažnio parametrą ir su gautais filtrais galima atlikti greitus skaičiavimus, bet detaliai šios procedūros čia neaprašinėsime. Norint išspręsti Gaboro požymių lėto apskaičiavimo problemą, galima panaudoti eksponentinį vidurkinimą ir Teiloro požymius, kurie taip pat greitai apskaičiuojami.

1.6. Tikėtinumo santykio logaritmas

Lyginant testuojamą paveikslą su apmokyme naudotais pavyzdžiais yra būtina nustatyti jų tarpusavio panašumą ar nepanašumą. Be šio mato niekaip negalima apsieiti surandant testuojamam paveikslui panašiausią atitikmenį. Pagal šią metriką sutapdinamas asmuo su testuojamu paveikslu bus tas, kurio panašumo koeficientas bus didžiausias. Taip pat vertėtu nustatyti šios metrikos ribą, kurios neviršijus testuojamas paveikslas būtų priskiriamas nepažįstamajam asmeniui, tai yra – asmeniui kuris nėra įtrauktas į apmokymo duomenų bazę.

Lyginamų porų panašumo galimų reikšmių sritis priklauso nuo lyginimo metrikos. Tai apsunkina panašumo reikšmių interpretaciją. Tarkime jei vienos poros panašumo reikšmė yra $\rho = 0{,}8$, o kitos – $\rho = 0{,}65$, galime teigti, kad santykinai pirmosios poros veidai panašesni nei antrosios poros veidai, tačiau į kiekybinį klausimą „kiek kartų labiau panašūs“ atsakyti negalima. Todėl biometrikoje vis dažniau taikoma tikėtinumo santykio (angl. *Likelihood Ratio, LR*) metrika, kurios reikšmes galima interpretuoti kiekybiškai. Lyginamos biometrijos poros (X, Y) panašumo tikėtinumo santykis apibrėžiamas formule:

$$LR(X,Y) = \frac{\text{Tikėtinumas, kad lyginamos poros } (X,Y) \text{ biometrikos sutampa}}{\text{Tikėtinumas, kad lyginamos poros } (X,Y) \text{ biometrikos nesutampa}} \quad (1.6)$$

1.4 lentelė Tikėtinumo santykio LR ir jo natūraliojo logaritmo LLR reikšmės

LR	LLR	Interpretacija
1000	6,9	Labai tikėtina, kad lyginami veidai sutampa
403,4	6	Labai tikėtina, kad lyginami veidai sutampa
100	4,6	Pakankamai tikėtina, kad lyginami veidai sutampa
20,1	3	Tikėtina, kad lyginami veidai sutampa
10	2,3	Labiau tikėtina, kad lyginami veidai sutampa
7,4	2	Labiau tikėtina, kad lyginami veidai sutampa
1	0	Vienodai tikėtina, kad lyginami veidai sutampa arba nesutampa
1 / 7.4	-2	Labiau tikėtina, kad lyginami veidai nesutampa
1 /10	-2,3	Labiau tikėtina, kad lyginami veidai nesutampa
1 / 20.1	-3	Tikėtina, kad lyginami veidai nesutampa
1 / 100	-4,6	Pakankamai tikėtina, kad lyginami veidai nesutampa
1 / 403.4	-6	Labai tikėtina, kad lyginami veidai nesutampa
1 / 1000	-6,9	Labai tikėtina, kad lyginami veidai nesutampa

Bendrąja prasme tikėtinumas yra modelio tikimybė, kai žinomi atlikto eksperimento rezultatai. Mūsų atveju „eksperimento rezultatai“ yra veidai X ir Y ir jų požymiai. Skaitiklio tikėtinumas yra kokio nors modelio tikimybė gauti („išmatuoti“) X ir Y veidų požymius, darant prielaidą, kad veidai yra to paties asmens, o vardiklio tikėtinumas yra tikimybė gauti tuos pačius X ir Y požymius, darant prielaidą, kad veidai yra skirtingų asmenų. Pagal apibrėžimą LR reikšmės gali kisti nuo 0 iki ∞. Jei $LR = LR(X, Y) < 1$, labiau tikėtina, kad lyginami veidai X ir Y yra skirtingi. Priešingai, jei $1 < LR < \infty$, labiau tikėtina, kad lyginami veidai sutampa. Kad išvengti intervalų $(0, 1)$ ir $(1, \infty)$ ilgių asimetrijos, dažnai vartotojui pateikiama LR natūraliojo logaritmo reikšmė, kuri sutrumpintai žymima $LLR(X, Y) = LLR = \log(LR(X, Y))$ (angl. *Log Likelihood Ratio, LLR*). Jei tikėtinumo santykio logaritmas teigiamas, labiau tikėtina, kad lyginamos poros veidų pavyzdžiai yra vieno asmens. Priešingu atveju, kai tikėtinumo santykio logaritmas yra neigiamas, labiau tikėtina, kad tiriamos poros veidų pavyzdžiai priklauso skirtingiems asmenims.

1.4 lentelėje pateiktos LR ir LLR reikšmės bei jų interpretacija. Pavyzdžiui, jei $LLR = 6{,}9$, tai apie 1000 kartų labiau tikėtina, kad lyginamojo ir tiriamojo veidų pavyzdžiai X ir Y priklauso vienam ir tam pačiam asmeniui nei skirtingiems ir atvirkščiai, jei $LLR = -6{,}9$, tai apie 1000 kartų labiau tikėtina, kad lyginamojo ir tiriamojo veidų pavyzdžiai priklauso skirtingiems asmenims nei tam pačiam asmeniui. Tikėtinumo santykiui įvertinti naudojami visi turimi lyginamieji veidų pavyzdžiai ir vieno tiriamųjų katalogo veidų pavyzdžiai. Todėl kuo lyginamųjų daugiau, tuo LLR patikimumas didesnis.

Žinoma, egzistuoja ir kitokių požymių palyginimo algoritmų. Pavyzdžiui, formuluojant apibendrinamuosius statistinius rezultatus, galima išskirti du pagrindinius kriteri-

jus [19] kuriais reikia remtis:

1. Centrinės tendencijos matas: reikia pasirinkti statistiką kuri rodytų kiek panašūs yra skirtingi objektai;
2. Statistinio kintamumo matas: pasirinkti kitą statistiką, kuri rodytų kaip labai jie skiriasi.

Šiuo atveju, apie tai ar lyginami veidai sutampa galima spręsti jau pagal dvi savybes – jų panašumą ir nepanašumą.

1.7. Skyriaus apibendrinimas

Veidų atpažinimui yra sukurta daugybė skirtingų algoritmų, kurie yra vienas už kitą pranašesni. Pagrindinis iššūkis projektuojant veidų atpažinimo sistemą yra nuspręsti kokius veido bruožus pasirinkti ir kaip pagal juos klasifikuoti veidus. Taip pat projektuojant tokią sistemą yra būtina atsižvelgti į tai, kas bus jei į jos įėjimą bus pateiktas paveikslas kuriame yra nežinomas veidas arba paveikslas kuriame išvis nėra veido.

Didelę įtaką veidų atpažinimui turi veido, kuris yra teikiamas į sistemos įėjimą, kokybė. Ši kokybė gali būti apibūdinama tokiais parametrais kaip fokusavimas, rezoliucija, centruotumas bendrame lange arba veido simetriškumas. Yra pastebėta, jog žmogaus galvai esant pasuktai daugiau nei 15° į kairę ar į dešinę pusę nuo kameros objektyvo, veidų atpažinimo santykis stipriai mažėja. Taip pat vertėtų pabrėžti, jog nors daugumą algoritmų veidų atpažinimui naudoja pilkšvus paveikslus, tyrimų metu yra pastebėta, kad spalvotuose paveiksluose veidai atpažįstami tiksliau. Žinoma, spalvų kiekis taip pat įtakoja ir paveikslo apdorojimo trukmę.

Visi veidų atpažinimo algoritmai gali būti suskirstyti į dvi pagrindines grupes: globalių (angl. *holistic*) ir lokalių (angl. *local*) požymių. Pavyzdžiui, veido suskaidymas į lokalias sritis LBP metode atspindi tą faktą, kad globalūs požymiai nelabai kokybiškai aprašo patį veidą ir todėl palaipsniui yra pereinama prie lokalesnių požymių. Gaboro požymiai taip pat yra priskiriami lokaliųjų grupei. Šie požymiai pasižymi ilga jų apskaičiavimo trukme, todėl dažniausiai yra naudojami tik išretintame centrų tinklelyje, kas menkina jų gebėjimą atskirti veidus. Siekiant išspręsti lėto požymių apskaičiavimo problemą neprarandant atpažinimo tikslumo galima naudoti eksponentinį vidurkinimą ir Teiloro požymius. Taip pat vertėtų paminėti, jog Gaboro tūta yra dalinai atspari įvairiems šviesos trukdžiams.

Globalių požymių grupei yra priskiriamas tikrinių veidų metodas, kuris yra spartus, nereikalauja daug vietos duomenų kaupimui ir gali būti pritaikytas tiek veidų atpažinimui, tiek ir jų aptikimui. Vienas pagrindinių šio metodo trūkumų yra tai, jog jis labai priklausomas nuo apšvietimo ir veido posūkio kampo.

2. OPTIMALAUS VEIDŲ ATPAŽINIMO ALGORITMO METODO ANALIZĖ

Siekiant išrinkti optimalų algoritmą, kurį būtų galima įgyvendinti *Android* operacinėje sistemoje, visų pirma reikėtų detaliau išanalizuoti bent keletą algoritmų. Šiame skyriuje bus mėginama apžvelgti tris veidų atpažinimo algoritmus – tikrinių veidų, Fišerio veidų bei 2D–DCT+SOM algoritmą. Kiekvienas šių algoritmų bus detaliai išanalizuotas – aptarta jų visų apmokymo bei testavimo stadijos. Pagal teorinę medžiagą šie trys algoritmai sekančiame skyriuje bus įgyvendinti *MATLAB* aplinkoje ir ištirti įvairus jų parametrai.

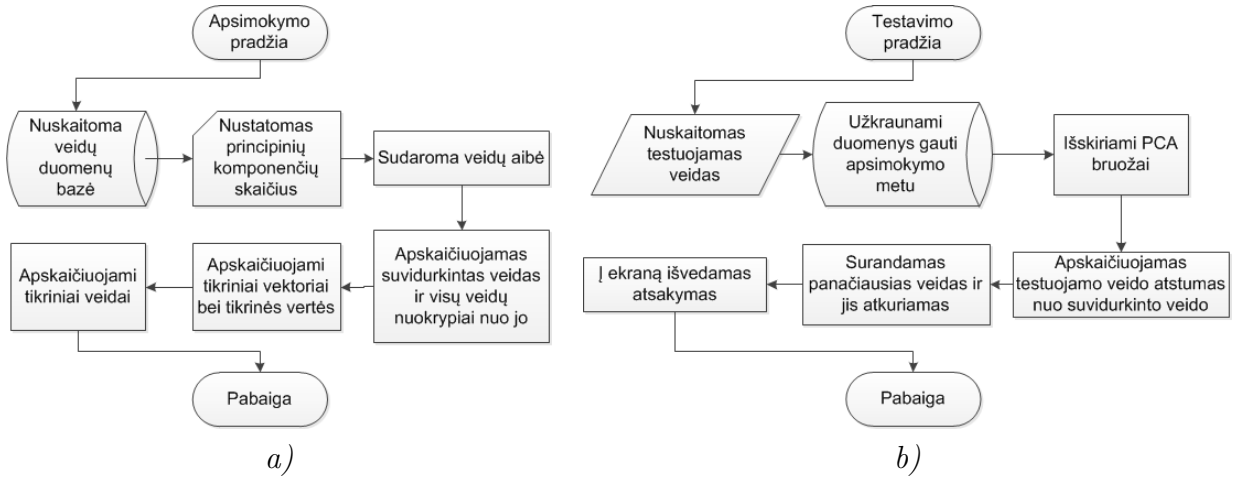
Tikrinių veidų algoritmas buvo pasirinktas dėl savo populiarumo – jau daugiau nei du dešimtmečiai šis algoritmas ir įvairios jo atmainos yra plačiai analizuojami įvairiausiuose literatūros šaltiniuose. Tuo tarpu Fišerio veidų algoritmas savo veikimo principu yra labai panašus į tikrinių veidų algoritmą, tik pasak kai kurių literatūros šaltinių – jis atsparesnis aplinkos poveikiams už pastarąjį. 2D–DCT ir SOM algoritmas savo veikimo principu yra visiškai kitos – jis remiasi 2D–DCT koeficientais ir sugeba savarankiškai (be vartotojo įsikišimo) apsimokyti. Visi šie trys skirtingi algoritmai bus aptarti sekančiuose poskyriuose.

2.1. Tikrinių veidų algoritmas

Tikriniai veidai (angl. *eigenfaces*) ištiesių tėra rinkinys tikrinių vektorių (angl. *eigenvectors*) kurie yra naudojami įvairiems kompiuterinės regos uždaviniams spręsti, pavyzdžiui – veidų atpažinime. Šis metodas gali būti tiesiogiai susietas su Furje transformacija, kuri sudaro elektronikos inžinerijos pagrindus. Furje transformacija parodo, kad susumavus įvairių amplitudžių ir įvairių dažnių sinusinius signalus, galima puikiai atkurti originalų signalą. Tokiu pačiu principu veikia ir tikrinių veidų algoritmas – tikrinių veidų suma su tam tikrais koeficientais gali puikiai atkurti tam tikro asmens veidą. Siekiant tai įgyvendinti yra panaudojama principinių komponenčių analizė [20] (angl. *principal component analysis*, toliau PCA), kuri išskiria tam tikrus bruožų taškus.

Pradinė idėja jog tikrinius veidus galima taikyti veidų atpažinime buvo pradėta plėtoti 1987–ais metais mokslininkų L.Sirovich ir M.Kirby, o vėliau M.Turk ir A.Pentland ją pritaikė veidų klasifikavimo uždaviniams spręsti. Šis tikrinių veidų panaudojimas veidų atpažinimui yra laikomas pirmuoju sėkmingu veidų atpažinimo modeliu.

Šiame algoritme naudojant PCA yra siekiama didelių dimensijų duomenis sumažinti iki mažesnių dimensijų perkeliant juos į požymių erdvę – ko pasėkoje duomenys yra aprašomi ekonomiškiau [21]. Pagrindinė PCA idėja veidų atpažinime yra išreikšti didelių matmenų vienmatį vektorių, kuris buvo gautas iš dvimačio duomenų masyvo atitinkančio veido paveikslą, į kompaktiškas principines komponentes bruožų erdvėje. Tai dar gali būti



2.1 pav. Tikrinių veidų *a)* apsimokymo ir *b)* atpažinimo algoritmai

vadinam eigen–erdve (angl. *eigenspace*). Ši erdvė yra sudaryta iš eigen–vektorių, gautų iš veidų paveikslų [18] (vektorinėje formoje). Į kiekvieną tikrinį veidą galima žiūrėti kaip į bruožą, kuriam galima priskirti atitinkamą svorio koeficientą. Naudojant PCA veidų atpažinimui yra lyginami testuojamo paveikslo bruožų vektorius ir visi kiti bruožų vektoriai iš apmokymo aibės. Šiuo lyginimu yra ieškoma pačio panašiausio vektoriaus, kuris atitiktų tą patį veidą.

Tikrinių veidų metodas susideda iš dviejų pagrindinių etapų, tai yra algoritmo apmokymo su pasirinktais pavyzdžiais ir algoritmo testavimo arba kitais žodžiais tariant – veidų atpažinimo, abu šie etapai yra pavaizduoti 2.1 paveiksle. Kiekvienas šis etapas susideda iš smulkesnių žingsnių kuriuos visus reikia atlikti norint jog algoritmas veiktų.

Tikrinių veidų algoritmo apmokyme galima išskirti 6 pagrindinius etapus. Visų pirma, norint apmokyti algoritmą reikia turėti veidų rinkinį S kuris susidėtų iš M vienodų matmenų veidų (paveikslų), kurie pirmajame etape bus transformuojami.

1. Sudarant veidų rinkinį S, kiekvienas paveikslas yra transformuojamas į N ilgio vektorių Γ. N yra apskaičiuojamas sudauginant paveikslo horizontalius ir vertikalius matmenis pikseliai (pavyzdžiui, jei paveikslo matmenys yra 50×50 px, tai $N = 2500$). Visi šie vektoriai yra lygiagrečiai sugrupuojami į matricą S:

$$S = \{\Gamma_1, \Gamma_2, \Gamma_3, \ldots, \Gamma_M\}. \tag{2.1}$$

2. Kai yra sudarytas veidų rinkinys S, tuomet galima apskaičiuoti suvidurkintą veidą Ψ kuris yra pavaizduotas 2.2 paveiksle. Jis yra gaunamas sudedant visus apmokymo

2.2 pav. Suvidurkintas veidas

veidų vektorius Γ ir padalinant iš apmokymo veidų skaičiaus M:

$$\Psi = \frac{1}{M} \sum_{N=1}^{M} \Gamma_n. \tag{2.2}$$

3. Toliau yra apskaičiuojamas skirtumas Φ tarp kiekvieno apmokymo paveikslo Γ ir suvidurkinto veido Ψ:

$$\Phi_i = \Gamma_i - \Psi. \tag{2.3}$$

4. Apskaičiuojamas M dydžio ortonormalių vektorių (u_n) rinkinys kuris geriausiai apibūdina duomenų pasiskirstymą. k–tasis vektorius u_k yra parenkamas toks, kad:

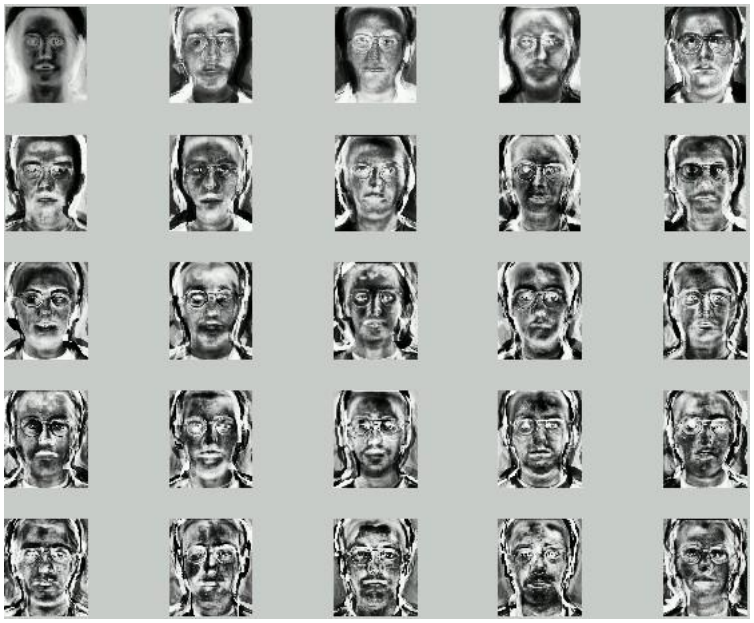
$$\lambda_k = \frac{1}{M} \sum_{n=1}^{M} (u_k^T \Phi_n)^2. \tag{2.4}$$

būtų maksimalus atsižvelgiant į:

$$u_l^T u_k = \delta_l k = \begin{cases} 1, & \text{jei } l = k \\ 0, & \text{kitais atvejais} \end{cases}. \tag{2.5}$$

Šiose formulėse u_k ir λ_k yra kovariacinės matricos C tikriniai vektoriai (angl. *eigenvectors*) ir tikrinės vertės (angl. *eigenvalues*) atitinkamai.

5. Kovariacinė matrica C pasirinktam duomenų rinkiniui yra apskaičiuojama pasinau-



2.3 pav. Tikriniai veidai

dojant skirtumu Φ kuris buvo gautas 3 žingsnyje:

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = \frac{1}{M}\sum_{n=1}^{M}\begin{pmatrix} var(p_1) & \cdots & var(p_1,p_N) \\ \vdots & \ddots & \vdots \\ cov(p_N,p_1) & \cdots & var(p_N) \end{pmatrix}_n = AA^T, \tag{2.6}$$

čia p_i yra n–tojo veido i–tasis pikselis, o A:

$$A = \{\Phi_1, \Phi_2, \Phi_3, \ldots, \Phi_n\}. \tag{2.7}$$

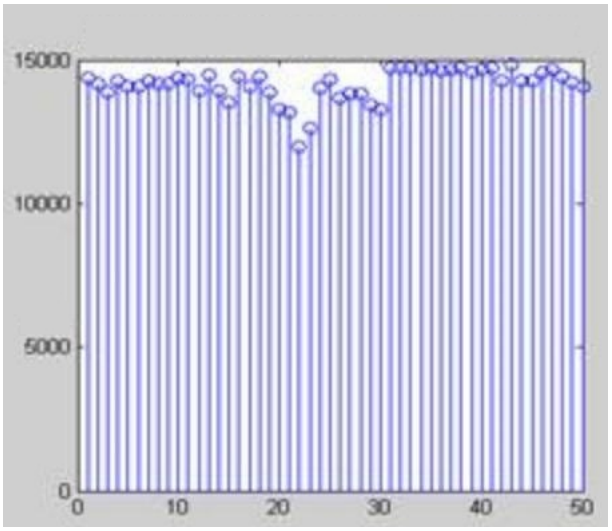
6. Suradus tikrinius vektorius (v_l ir u_l) galima atvaizduoti tikrinius veidus kurie yra pavaizduoti 2.3 paveiksle.

Siekiant kritiškai įvertinti šį algoritmą programa yra testuojama pasirinkus tam tikrą paveikslą kuris nebuvo tarp apmokymo pavyzdžių. Testavimą sudaro 4 žingsniai kurie yra aprašyti žemiau.

1. Visų pirma testuojamas veidas yra transformuojamas į tikrinių veidų komponentes. Tuomet jis yra palyginamas su suvidurkintu veidu ir jų skirtumas padauginamas iš kiekvieno L matricos tikrinio vektoriaus. Kiekviena gauta vertė apibūdina tam tikrą svorį, kurie visi kartu yra išsaugomi kaip Ω vektorius:

$$\omega_k = u_k^T(\Gamma - \Psi), \tag{2.8}$$

$$\Omega^T = [\omega_1, \omega_2, \ldots, \omega_M]. \tag{2.9}$$



2.4 pav. Testuojamo paveikslo Euklido atstumas

2. Dabar galima surasti kuri veidų klasė labiausiai atitinka (yra panašiausia) į testuojamą paveikslą. Tai atliekama minimizuojant Euklido atstumą (žr. 2.4 pav.).

3. Yra tariama, jog testuojamas paveikslas priklauso tam tikram asmeniui, jei ε_k yra mažesnis nei pasirinkta slenkstinė vertė Θ_ε. Jei ši sąlyga yra tenkinama, tuomet testuojamas paveikslas yra laikomas kaip „žinomas veidas“. Priešingu atveju, kai Euklido atstumas yra didesnis už pasirinktą slenkstį Θ_ε, tačiau mažesnis už sekantį slenkstį Θ'_ε, tuomet testuojamas paveikslas yra atpažįstamas kaip veidas kurio duomenų bazėje nėra. Galiausiai, jei atstumas yra didesnis už antrąjį slenkstį, tuomet yra laikoma jog testuojamame paveiksle nėra veido.

4. Jei yra nustatoma, jog testuojamas paveikslas yra „nežinomas veidas“, tuomet reikia nuspręsti kaip bus elgiamasi toliau – ar norima naują pavyzdį įtraukti tarp apmokymo pavyzdžių ar ne. Jei nutariama jį įtraukti, tuomet algoritmą reikia apmokyti per naujo pakartojant 1–6 apmokymo žingsnius.

Tokioje sistemoje veidų erdvė susideda iš žinomų veidų gautų apmokymo etape, o į įėjimą teikiamas veidas yra lyginamas su šia erdve ir skaičiuojamas jo atstumas nuo žinomų veidų. Pasak W. S. Yambor et al. [22], atstumo skaičiavimui PCA sistemoje geriausiai tinka Mahalanobis algoritmas, kuris duoda tikslesnius atsakymus lyginant su Euklido atstumu ar neigiamo kampo tarp paveikslo vektorių skaičiavimu. Taip pat, išmėginus įvairias šių atstumo matavimo algoritmų kombinacijas buvo pastebėta, jog tai nelemia geresnių rezultatų ir Mahalanobis algoritmas išlieka lyderio pozicijoje.

Projektuojant tokią sistemą, vertėtų atsižvelgti į du klausimus:

1. Kas bus jei į įėjimą pateiktas paveikslas yra ne veidas?

2. Kas bus jei sistemai pateiktas veidas yra sistemai nežinomas veidas?

Pirma problema gali būti lengvai išspręsta nes tikrinis veidas yra geras veidų filtras ir galima patikrinti kaip kiekvienas vaizdas koreliuoja su nurodytu vaizdu. Vaizdas su maža koreliacija gali būti atmestas kaip netinkamas. Taip pat abi šias problemas galima išspręsti paveikslą kategorizuojant pagal keturias taisykles:

1. Paveikslas arti veidų erdvės ir arti žinomų veidų → žinomas veidas;

2. Paveikslas arti veidų erdvės tačiau toli nuo žinomų veidų → nežinomas veidas;

3. Paveikslas toli nuo veidų erdvės tačiau arti žinomų veidų → ne veidas;

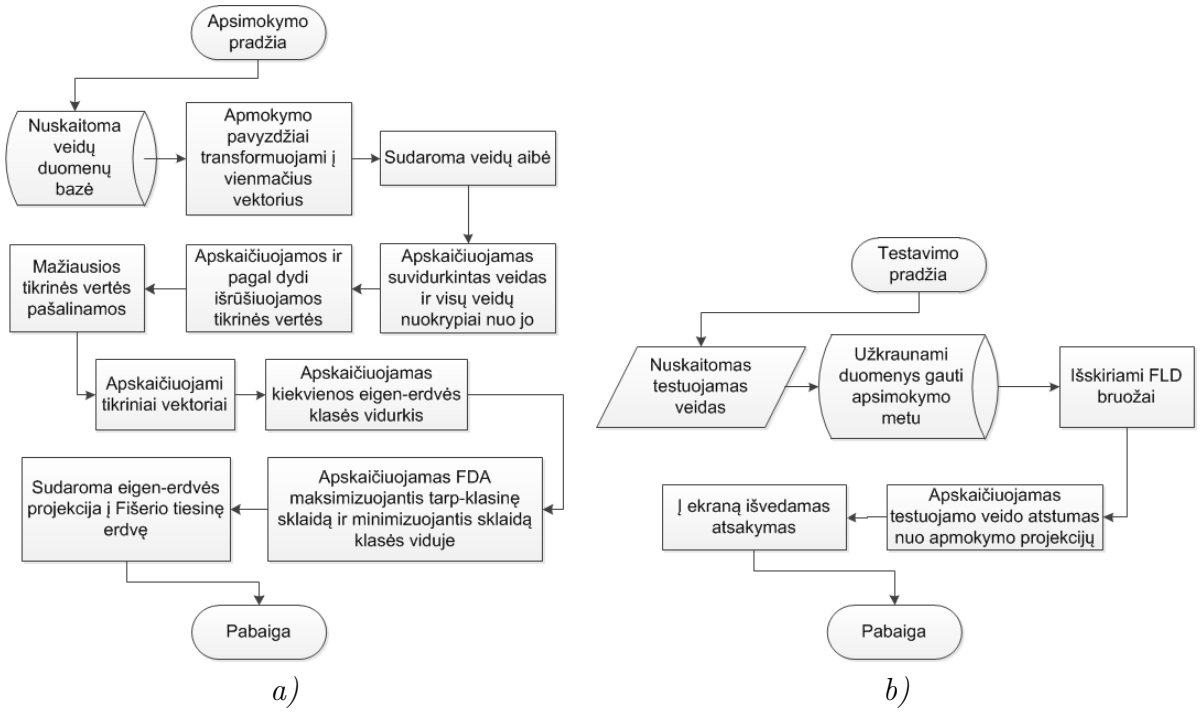
4. Paveikslas toli nuo veidų erdvės ir žinomų veidų → ne veidas.

Jei veidas yra arti veidų erdvės, tai jo atkūrimo paklaida bus labai nedidelė ir atkurtas paveikslas bus labai artimas originalui, o tuo tarpu paveikslas neturintis veido bus atkurtas su didele paklaida. Kuo apmokymo aibė yra didesnė, tuo atkūrimo paklaida mažesnė [21].

Vienas pagrindinių trūkumų atpažįstant ir aptinkant veidus tikrinių veidų metodu yra tas, jog jis yra labai priklausomas nuo apšvietimo lygio ir veido posūkio kampo. Iš to išplaukia išvada, jog norint priversti programą veikti kuo tiksliau – visi veidai turėtų būti apšviesti vienodo lygio šviesa bei pateikti algoritmui tiesiai iš priekio (pranc. *en face*). Taip pat vertėtų paminėti, jog visi algoritmui perduodami paveikslai turi būti vienodų matmenų, o veidai juose – centruoti. Tikriniai veidai taip pat susiduria su tam tikromis problemomis kai veidas yra plaukuotas arba randuotas. Kita vertus, šis algoritmas veikia gana sparčiai ir nereikalauja daug vietos duomenų kaupimui, o daugumą prieš tai paminėtų problemų galima išspręsti panaudojus eigen–bruožus (angl. *eigenfeatures*) [23]. Šie bruožai išmatuoja veido metrikas, tokias kaip atstumas tarp akių ir nosies, ir šio algoritmo kombinacija su tikriniais veidais žymiai pagerina veidų atpažinimą.

2.2. Fišerio veidų algoritmas

Fišerio veidų (angl. *fisherface*) algoritmo pavadinimas yra kildinamas nuo R.A.Fišerio kuris dar 1930 metais plėtojo tiesinio diskirminanto analizės temą. Lyginant su kitais veidų atpažinimo algoritmais šis metodas yra labiau atsparus šviesos srauto bei veido išraiškų pokyčiams [2]. Kaip matyti iš 2.6 paveikslo, žmogaus nuotraukos nekintant veido išraiškai ir taškui iš kurio yra fotografuojama gali labai skirtis priklausomai nuo apšvietimo šaltinių skaičiaus, galingumo bei jų pozicijų. Šis atsparumas yra pasiekiamas sudarant tiesinę veidų projekciją iš daugiadimensinės paveikslo erdvės į žymiai mažiau dimensijų turinčią bruožų erdvę kuri yra nejautri veido išraiškų bei apšvietimo kampo pokyčiams. Projekcijos kryptis turi būti parinkta tokia, kad ji būtų beveik statmena vidinei klasės sklaidai (angl. *within–class scatter*). Šis metodas yra išvestas iš tiesinio Fišerio



2.5 pav. Fišerio veidų *a)* apsimokymo ir *b)* atpažinimo algoritmai

diskriminanto (angl. *Fisher's linear discriminant*, toliau FLD) [24], kuris maksimizuoja tarp–klasinės (angl. *between–class scatter*) ir klasės sklaidos santykį. Bendrinė Fišerio veidų algoritmo schema yra pateikta 2.5 paveiksle.

Kaip jau minėta anksčiau, tikrinių veidų metodas taip pat yra pagrįstas tiesine paveikslų erdvės projekcija į mažiau dimensijų turinčią bruožų erdvę [25, 26, 27]. Tikrinių veidų metodas dimensijų sumažinimui naudoja PCA metodą, ko pasekoje yra maksimizuojama sklaida tarp visų klasių, pavyzdžiui visų veidų visiems paveikslams. Po tokio sklaidos maksimizavimo išlieka nepageidautinos apšvietimo ir veido išraiškų variacijos. Pasak Y.Moses, Y.Aldini ir S.Ullman, apšvietimo ir veido posūkio kampo variacija tarp paveikslų priklausančių tam pačiam veidui beveik visada yra didesnė nei paveikslo variacija dėl netinkamai nustatytos tapatybės [28].

2.6 pav. Nuotraukų skirtumai pagal veido apšvietimą [2]

Tikrinių veidų algoritmas pasinaudoja faktu, jog prie idealių sąlygų duomenų variacija klasės viduje vyksta paveikslų erdvėje, ko pasekoje klasės tampa išgaubtos ir jos gali būti atskirtos tiesiškai. Pritaikius tiesinę projekciją dimensijų mažinimui klases vis dar galima tiesiškai atskirti, tad tai ir yra pagrindinis argumentas, kodėl tiesinių metodų taikymas yra tinkamas veidų atpažinime dimensijų mažinimui.

Kadangi visi apmokymo pavyzdžiai būna žinomi ir sužymėti tam tikromis etiketėmis nusakančiomis asmens tapatybę, visai logiška tampa panaudoti šią informaciją kuriant patikimesnį metodą bruožų erdvės dimensijų mažinimui. Fišerio veidų algoritme dimensijų mažinimui yra panaudojamas klasei būdingasis tiesinis metodas bei paprastas klasifikatorius sumažintoje bruožų erdvėje – ko pasekoje yra gaunamas geresnis atpažinimo santykis lyginant su tikrinių veidų metodu. Fišerio tiesinis diskriminantas (FLD) yra vienas iš klasei būdingųjų tiesinių metodų pavyzdžių, kuris siekia sumažinti erdvės dimensijas ir padaryti ją patikimesnę klasifikacijos atžvilgiu. Šis metodas išrenka tokius W, kad santykis tarp tarp–klasinės sklaidos (2.10) ir klasės sklaidos (2.11) matricų yra maksimalus.

Tarkime, kad tarp–klasinės sklaidos matrica yra apibrėžta kaip:

$$S_B = \sum_{i=1}^{c} N_i(\mu_i - \mu)(\mu_i - \mu)^T, \tag{2.10}$$

o klasės sklaidos matrica:

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T, \tag{2.11}$$

čia μ_i yra suvidurkintas X_i klasės paveikslas, o N_i yra pavyzdžių skaičius X_i klasėje. Jei S_W yra nesinguliari, tuomet W_{opt} yra parenkama kaip matrica su ortonormaliais stulpeliais kurie maksimizuoja determinanto santykį tarp pavyzdžių projekcijų tarp–klasinės sklaidos matricos ir pavyzdžių projekcijos klasės sklaidos matricos determinanto, pavyzdžiui:

$$W_{opt} = \arg\max_W \frac{\left|W^T S_B W\right|}{\left|W^T S_W W\right|} = [w_1 \ w_2 \ \dots \ w_m], \tag{2.12}$$

čia $\{w_i \mid i = 1, 2, \dots, m\}$ yra S_B ir S_W apibendrintų tikrinių vektorių rinkinys, atitinkamai pagal m didžiausių apibendrintų tikrinių verčių $\{\lambda_i \mid i = 1, 2, \dots, m\}$, pavyzdžiui:

$$S_B w_i = \lambda_i S_W w_i, \qquad i = 1, 2, \dots, m. \tag{2.13}$$

Verta pabrėžti, jog turi egzistuoti bent jau $c - 1$ apibendrintų tikrinių verčių kurios yra nelygios nuliui ir viršutinis rėžis m yra $c - 1$, kur c yra klasių skaičius (žr. 2.12 formulę).

Veidų atpažinimo uždavinyje yra susiduriama su problema, kad klasės sklaidos matrica $S_W \in \mathbb{R}^{n \times n}$ yra visada singuliari. Tai išplaukia iš fakto, kad S_W laipsnis yra nedidesnis

nei $N - c$ ir paveikslų skaičius apmokymo rinkinyje N yra žymiai mažesnis nei pikselių skaičius kiekviename paveiksle n. tai reiškia, kad įmanoma parinkti tokią matricą W, kad projektuojamų pavyzdžių klasės sklaida būtų lygi nuliui.

Siekiant išvengti įvairių komplikacijų susijusių su S_W singuliarumu, yra pasirenkama alternatyva (2.12) formulei. Fišerio veidų algoritmas išvengia šios problemos projektuodamas paveikslų rinkinį į mažesnių dimensijų erdvę, kad galutinė klasės sklaidos matrica S_W būtų nesinguliari. Bruožų erdvės dimensijos yra sumažinamos iki $N - c$ pasitelkiant PCA ir vėliau pritaikant standartinę FLD procedūrą kuri sumažina dimensijas iki $c - 1$. Kitais žodžiais tariant, W_{opt} yra apskaičiuojamas

$$W_{opt}^T = W_{fld}^T W_{pca}^T, \tag{2.14}$$

čia:

$$W_{pca} = \arg\max_W \left| W^T S_T W \right|, \tag{2.15}$$

$$W_{fld} = \arg\max_W \frac{\left| W^T W_{pca}^T S_B W_{pca} W \right|}{\left| W^T W_{pca}^T S_W W_{pca} W \right|}. \tag{2.16}$$

W_{pca} optimizacija yra atliekama su $n \times (N - c)$ matricų su ortonormuotais stulpeliais, kai tuo tarpu W_{fld} optimizacija yra atliekama tik su $(N - c) \times m$ su ortonormuotais stulpeliais. Apskaičiuojant W_{pca} yra pašalinami tik c mažiausių principinių komponenčių.

Egzistuoja ir kitas būdas skirtas sumažinti klasės sklaidą išlaikant tarp–klasinę sklaidą – jame pirma sumažinus klasės sklaidą yra parenkamas toks W kuris maksimizuotų tarp–klasinę projekcijų sklaidą. Čia galima maksimizuoti projekcijų tarp–klasinę sklaidą atsižvelgiant į tai, jog klasės sklaida yra lygi nuliui, pavyzdžiui:

$$W_{opt} = \arg\max_{W \in \mathcal{W}} \left| W^T S_B W \right|, \tag{2.17}$$

čia $\mathcal{W}$ yra $n \times m$ matricų rinkinys su ortonormuotais stulpeliais ir sudarantis S_W branduolį.

Apibendrinant Fišerio veidų algoritmą galima teigti, jog lyginant su tikrinių veidų algoritmu jis yra žymiai atsparesnis šviesos bei veido mimikų pokyčiams, tačiau jei nuotraukose vyrauja ryškūs šešėliai – atpažinimo santykis žymiai sumažėja [2]. Vienas iš pagrindinių Fišerio metodo trūkumų yra tas, jog klasės sklaidos matrica praktiškai visuomet yra singuliari. Taip nutinka dėl to, jog paveikslo pikselių skaičius dažniausiai būna žymiai didesnis už pačių paveikslų skaičių ir tai neigiamai įtakoja atpažinimo santykį [29]. Pastaruoju metu vietoje tiesinio diskriminanto analizės (LDA) yra siūloma naudoti dvimatę tiesinę diskriminanto analizę (2DLDA) [30]. Pastaroji nuo paprasto LDA metodo skiriasi tuo, jog nereikalauja, kad paveikslas būtų transformuotas iš dvimatės erdvės į vienmatį vektorių. 2DLDA sugeba tiesiogiai iš paveikslo matricos išskirti reikiamus bruožus pagrįstus Fišerio diskriminanto analize.

2.3. 2D–DCT ir SOM taikymas veidų atpažinimui

Diskrečioji kosinuso transformacija (toliau DCT) yra labai plačiai naudojamas algoritmas, dažniausiai taikomas duomenų glaudinimui (pavyzdžiui paveikslų glaudinimui skirtas JPEG formatas [31]). DCT turi tokią savybę, jog daugumoje paveikslų svarbiausia vizuali informacija yra sukoncentruota keletoje koeficientų. Išgavus šiuos DCT koeficientus, juos galima panaudoti kaip unikalų kiekvieno paveikslo parašą veidų atpažinimui [32, 33].

Veidų nuotraukos dažniausiai turi daug perteklinės informacijos ir aukštą koreliaciją, ko pasekoje yra gaunamas skaičiavimų perteklius įtakojantis atminties išnaudojimą ir apdorojimo laiką. DCT transformuoja paveikslus iš erdvinės plokštumos į dažninę plokštumą ir, kadangi žemi dažniai yra vizualiai svarbesni už aukštuosius, jis atmeta aukštų dažnių koeficientus kvantizuodamas likusiuosius. Taip DCT sumažina duomenų apimtį neprarasdama per daug paveikslo kokybės [34].

$M \times N$ matmenų A matricos dvimatė diskrečioji kosinuso transformacija (angl. *2D Discreet Cosine Transformation*, toliau 2D–DCT) yra apibrėžiama sekančiai:

$$B_{pq} = \alpha_p \alpha_q B_{pq} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos\left(\frac{\pi(2m+1)p}{2M}\right) \cos\left(\frac{\pi(2n+1)q}{2N}\right), \begin{array}{l} 0 \leq p \leq M-1 \\ 0 \leq q \leq N-1 \end{array}. \tag{2.18}$$

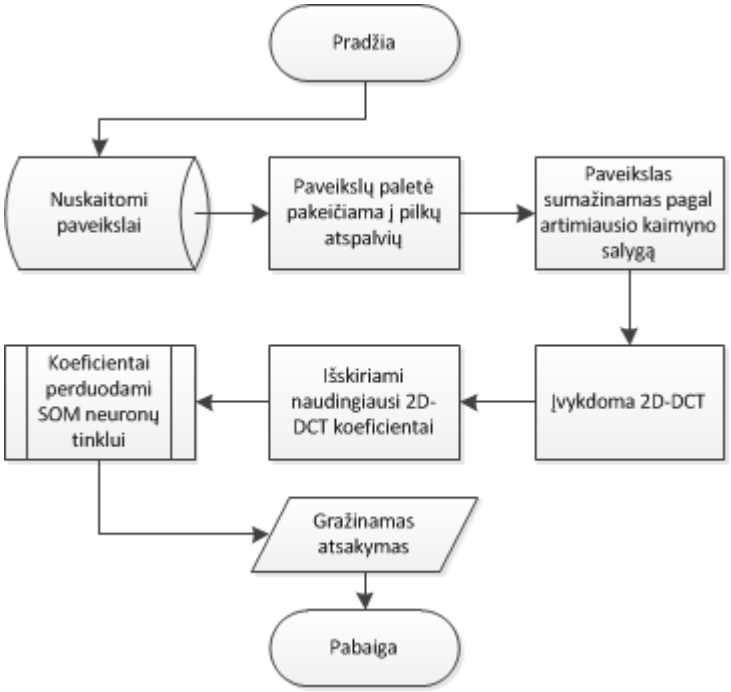
2.18 formulėje B_{pq} yra 2D–DCT koeficientų reikšmės. Taip pat vertėtų pabrėžti, jog diskrečioji kosinuso transformacija yra dvipusė, tad atvirkštinė 2D–DCT (angl. *2D Inverse Discreet Transformation*, toliau 2D–IDCT) yra apibrėžiama:

$$A_{pq} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \alpha_p \alpha_q B_{pq} \cos\left(\frac{\pi(2m+1)p}{2M}\right) \cos\left(\frac{\pi(2n+1)q}{2N}\right), \begin{array}{l} 0 \leq p \leq M-1 \\ 0 \leq q \leq N-1 \end{array}. \tag{2.19}$$

Reikšmės α_p ir α_q naudojamos 2.18 ir 2.19 formulėse yra apskaičiuojamos:

$$\alpha_p = \begin{cases} \sqrt{\frac{1}{M}}, & p = 0 \\ \sqrt{\frac{2}{M}}, & 1 \leq p \leq M-1 \end{cases} \quad \alpha_q = \begin{cases} \sqrt{\frac{1}{N}}, & q = 0 \\ \sqrt{\frac{2}{N}}, & 1 \leq q \leq N-1 \end{cases}. \tag{2.20}$$

Save organizuojantis tinklas (angl. *Self Organizing Map*, toliau SOM), dar kitaip žinomas kaip Kohoneno žemėlapis, yra vienas iš dirbtinių neuronų tinklų (toliau DNT) porūšių. Šis dirbtinių neuronų tinklas sugeba apsimokyti be priežiūros, turi topologijos įsiminimo galimybę ir jame vyksta varžymasis tarp daugybės neuronų kuris bus aktyvuotas ar iššautas. Galutiniame rezultate laimi tik vienas neuronas kuris yra iššautas – tai yra uždavinio atsakymas.



2.7 pav. Algoritmo struktūrinė veikimo schema

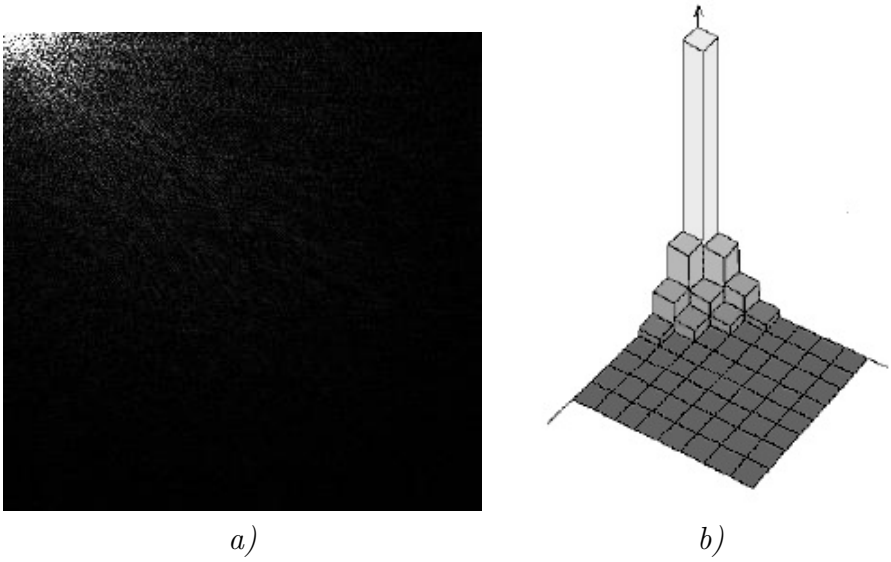
Šiame DNT, vietoje kiekvieną sykį visų neuronų svorių atnaujinimo yra atnaujinami tik laimėjusio neurono kaimyninių neuronų svoriai. Šis atnaujinimas vyksta pagal *Kohoneno taisyklę*, kuri leidžia neuronų svoriams išmokti įėjimo vektorių, ko pasekoje šis DNT yra tinkamas atpažinimo uždaviniams spręsti. Dažniausiai SOM tinklas susideda iš trijų fazių: mokymosi, treniravimosi ir testavimo.

Mokymosi fazėje neuronas kurio svoriai yra artimiausi įėjimo duomenų vektoriui yra pažymimas kaip nugalėtojas. To pasekoje visų nugalėjusio neurono kaimyninių neuronų svoriai yra atnaujinami tam tikru koeficientu atvirkščiai proporcingu Euklido atstumui. Tuo tarpu apmokymui naudojami paveikslai SOM tinklo pagalba yra priskiriami žemesnei dimensijai ir kiekvieno paveikslo svorių matrica yra išsaugomo apsimokymo duomenų bazėje. Testavimo fazėje, kai yra bandoma atpažinti tam tikrą objektą, apmokymo paveikslai naudojant svorių matricas yra rekonstruojami ir lyginami su testuojamais paveikslais matuojant Euklido atstumą.

Šiame algoritme nespalvotų paveikslų intensyvumas iš pradžių yra apdorojamas dvimate diskrečiąja kosinuso transformacija ir iš 2D–DCT koeficientų yra išskiriami bruožų vektoriai. Sekančiame etape save organizuojantis tinklas naudojant neprižiūrimą mokymąsi yra apmokomas klasifikuoti bruožų vektorius į grupes (žr. 2.7 pav.).

Nuskaičius kiekvieną paveikslą, jis yra konvertuojamas į nespalvotą ir pritaikant artimiausio kaimyno interpoliaciją (angl. *nearest–neighbor interpolation*) visų jų matmenys yra sumažinami iki 8×8 px paveikslų blokų.

Šis veidų atpažinimo algoritmas kiekvienam 8×8 px dydžio apmokymo paveikslui



a) *b)*

2.8 pav. DCT koeficientų *a)* pasiskirstymas paveiksle ir *b)* koeficientu matrica su apskaičiuotomis vertėmis

pritaiko 2D–DCT ir iš 8×8 dydžio gautų koeficientų matricų paima tik 8 iš 64 DCT koeficientų verčių. Šios vertės yra imamos iš viršutinio kairiojo paveikslo kampo, nes čia yra empiriškai svarbiausios vertės (jos atitinka apdorotų paveikslų žemų dažnių komponentes [35], žr. 2.8 pav.). Likusieji 56 koeficientai yra atmetami juos prilyginant nuliui.

Toliau, iš apdorotos 2D–DCT koeficientų matricos yra apskaičiuojama 2D–IDCT ir rekonstruojamas kiekvienas paveikslas. Galiausiai yra gaunamas rinkinys 8×8 px dydžio matricų, kur kiekviena matrica atitinka skirtingą paveikslą. Šie rekonstruoti paveikslai yra transformuojami į vektorius (šiuo atveju į 64×1 matmenų vektorius) ir pateikiami į SOM tinklo įėjimą apsimokymui. Visos šių vektorių reikšmės kinta intervale $[0, 255]$, kas reiškia pilkumo intensyvumo lygius. Pats SOM tinklas yra sukuriamas su $[64, 2]$ pirmojo sluoksnio dimensijomis, kur kiekvienam paveikslo pikseliui yra atrenkamos 64 minimalios ir 64 maksimalios intensyvumo vertės. Pagal šiuos parametrus sukurtas tinklas yra vieno sluoksnio tiesioginio sklidimo (angl. *feed–forward*) SOM žemėlapis su 128 svorio koeficientų ir konkuruojančia perdavimo funkcija. Algoritmo testavimui su šiuo SOM tinklu, kiekvienam paveikslui turi būti pakartota analogiška procedūra prieš tai aptartajai.

Apibendrinant dirbtinių neuronų tinklų taikymą veidų atpažinimo uždaviniams spręsti galima teigti, jog jie turi nemažai trūkumų. Visų pirma, siekiant optimalių rezultatų yra būtina dirbtinių neuronų tinklą labai tiksliai suderinti (t. y. tinklo sluoksnių skaičių, įėjimų skaičių, apsimokymo santykį ir t. t.) [36]. Antra, dauguma DNT susiduria su problemomis kai apmokymui naudojamų klasių skaičius smarkiai išauga, arba pavyzdžių skaičius vienai klasei (asmeniui) yra per mažas – tokiu atveju DNT nesugeba tinkamai išskirti reikalingų bruožų. Nors įvairūs neuronų tinklai sugeba labai greitai išspręsti uždavinius, tačiau kita vertus – tinklo apmokymas gali trukti labai ilgai.

2.4. Skyriaus apibendrinimas

Išanalizavus tikrinių veidų, Fišerio veidų bei 2D–DCT+SOM algoritmus galima teigti, jog pirmieji du algoritmai yra pakankamai panašūs, tik naudoja skirtinas technikas – pirmasis naudoja PCA, o antrasis FLD. Tuo tarpu 2D–DCT+SOM algoritmas yra puikus tuo, jog jam užtenka pateikti 2D–DCT koeficientus pagal kuriuos jis pats sugeba apsimokyti klasifikuoti veidus bei juos atpažinti. Žinoma, norint pasiekti puikių rezultatų – būtina šį algoritmą, tiksliau patį SOM dirbtinių neuronų tinklą labai tiksliai suderinti. Kaip jau minėta anksčiau, 2D–DCT+SOM algoritmo apsimokymo trukmė gali būti labai didelė, lyginant su pirmųjų dviejų algoritmų, kai tuo tarpu atpažinimo trukmės yra pakankamai panašios.

Fišerio veidų algoritmas, pasak literatūros šaltinių, yra pranašesnis už tikrinių veidų algoritmą tuo, jog jis nėra jautrus šviesos ar veido išraiškų pokyčiams. Tuo tarpu Tikrinių veidų algoritmas yra itin populiarus iki šių dienų. Šis algoritmas yra laikomas pirmuoju sėkmingai įgyvendintu veidų atpažinimo modelio bei iki dabar yra tobulinamas bei naudojamas palyginimui su kitais algoritmais.

Apibendrinant visus šiuos tris algoritmus galima teigti, jog juose yra pasigesta tam tikro saugiklio. Norint įgyvendinti veidų atpažinimo sistemą yra būtina numatyti kaip sistema turėtų elgtis jei į ja bus pateiktas paveikslas kuriame yra nepažįstamo žmogaus veidas arba paveikslas kuriame išvis nėra veido. Fišerio bei tikrinių veidų algoritmuose šitą problemą galima lengvai išspręsti patikrinus kaip testuojamas paveikslas koreliuoja su suvidurkintu veidu, arba tiesiog nustačius tam tikrus atstumo slenksčius. Tarkime, jei testuojamo paveikslo Euklido atstumas viršytų pirmąjį slenkstį – testuojamas paveikslas būtų identifikuojamas kaip nežinomas asmuo, o tuo tarpu viršijus ir antrąjį slenkstį – paveikslas būtų prilyginamas neturinčiam savyje veido.

3. TESTAVIMO DUOMENŲ ATRANKA

Siekiant gauti kuo tikslesnius rezultatus, praeitame skyriuje aptarti algoritmai bus tiriami trijose veidų duomenų bazėse. Dvi veidų duomenų bazės – jau egzistuojančius ir laisvai internete prieinamos duomenų bazės. Pirmoji – AT&T duomenų bazė, antroji – Grimasų, o trečioji duomenų bazė buvo sudaryta šio tyrimo meto.

AT&T duomenų bazė [37], dar kitaip žinoma kaip „ORL veidų duomenų bazė“, susideda iš veidų rinkinio, kuris buvo sudarytas 1992–1994 metais AT&T laboratorijoje Kembridžo universitete. Ši duomenų bazė buvo sudaryta kartu su „Kalbos, Regos ir Robotikos grupe“ veidų atpažinimo uždaviniams tirti. Šioje duomenų bazėje viso yra 40 subjektų, kurių nuotraukos padarytos kintant laikui, apšvietimui bei veido išraiškai (atsimerkęs–užsimerkęs, šypsosi–nesišypso ir pan.) bei detalėms (subjektas su akiniais–be akinių). Visos nuotraukos buvo padarytos esant tolygiam tamsiam fonui fotografuojant asmenį iš priekio. Šioje duomenų bazėje visi paveikslai yra pateikiami PGM formatu 256–iais pilkumo atspalviais pikseliui, o kiekvieno paveikslo raiška – 92×112 px. Visi paveikslai yra surūšiuoti 40–tyje katalogų, kurių pavadinimai yra sudaryti sX formatu (čia X žymi subjekto numerį). Kiekviename kataloge yra 10 skirtingų subjekto nuotraukų, kurių pavadinimas sudarytas „Y.pgm“ formatu, čia Y žymi paveikslo numerį nuo 1 iki 10.

Grimasų duomenų bazė [38] (laisbai prieinama tyrimų tikslams) buvo sudaryta dr. Libor Spacek kompiuterinės regos laboratorijoje Esekso universitete. Ši duomenų bazė tėra dalis veidų duomenų bazės, kuri susideda iš „faces94“, „faces95“, „faces96“ ir „grimace“ rinkinių. Pastarieji du rinkiniai yra patys sudėtingiausi dėl keleto priežasčių – juose kinta fonas, veidų mastelis bei veido išraiškos. Visos nuotraukos šioje duomenų bazėje yra pateikiamos JPG formatu 24 bit RGB paletėje, jų raiška yra 180×200 px, viso duomenų bazėje yra 18 subjektų ir kiekvienas jų turi po 20 save apibūdinančių nuotraukų. Visi subjektai yra surūšiuoti kataloguose pavadintuose jų vardais, o nuotraukų pavadinimai yra sudaryti sekančiu formatu: VARDAS_exp.#.jpg. Čia VARDAS atitinka katalogo pavadinimą, o # nurodo nuotraukos eilės numerį, kuris kinta nuo 1 iki 20. Viso šiame rinkinyje yra 360 nuotraukų, kuriose yra tiek vyriškos, tiek moteriškos lyties subjektų. Visi subjektai yra įvairių rasių, 18–20 metų amžiaus, dalis jų dėvi akinius ar yra su barzda.

Paskutinioji duomenų bazė buvo sudaryta mano paties. Ši duomenų bazė susideda iš 49 nuotraukų, kurios priklauso 14 subjektu. Vidutiniškai kiekvienam subjektai tenka po 3 jį apibūdinančias fotografijas. Dalis šių fotografijų buvo surinktos internete, o kita dalis – fotografuojant pažįstamus asmenis. Visose šiuose fotografijose kinta fono spalva, veido išraiškos bei apšvietimas. Visos nuotraukos yra pateiktos JPG formatu 24 bit RGB paletėje, o jų išmatavimai yra 180×200 px. Kiekvieno subjekto nuotraukų pavadinimai yra pateikti VARDAS_###.jpg formatu. Čia VARDAS atitinka subjekto vardą, o ### nuotraukos numerį priskirtą tam subjektui. Šis numeris kinta ribose nuo 001 iki 006.

4. VEIDŲ ATPAŽINIMO TYRIMO REZULTATAI

Šiame skyriuje mėginama įsigilinti į visus algoritmus aptartus 2–ąjame skyriuje, suprasti jų veikimo principus ir įgyvendinti juos *MATLAB* terpėje. Įgyvendinus šiuos algoritmus bus ištirti jų atpažinimo santykiai bei išanalizuotos algoritmų apsimokymo bei atpažinimo spartos priklausomybės nuo pavyzdžių kiekio bei dydžio. Galiausiai visi tyrimo rezultatai bus apibendrinti, aptarti kiekvieno algoritmo pranašumai bei trūkumai ir optimalūs jų parametrai. Remiantis šio skyriaus išvadomis bus išrinktas algoritmas, kuris yra labiausiai tinkamas įgyvendinimui *Android* operacinėje sistemoje.

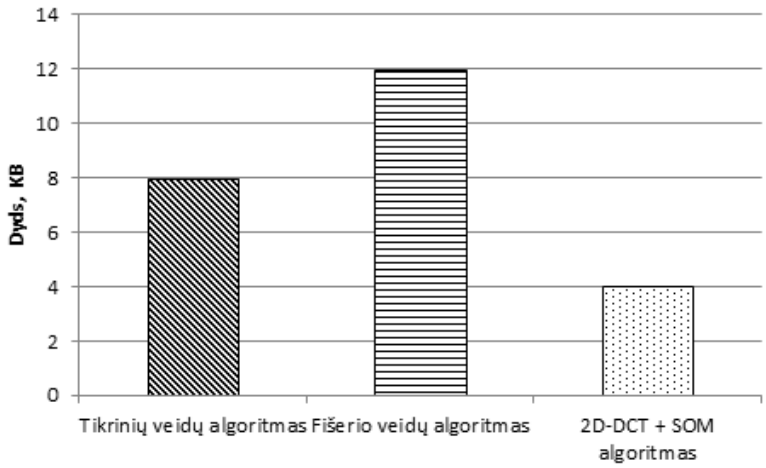
Visi šie tyrimai yra atlikti trijuose veidų duomenų bazėse aptartuose praeitame skyriuje. Taip pat vertėtų pabrėžti, jog siekiant kuo tikslesnių rezultatų, visuose šiuose tyrimuose testavimo metu nebuvo naudotas nei vienas paveikslas kuris buvo naudojamas apmokymui.

4.1. Algoritmų pirminio programos teksto palyginimas

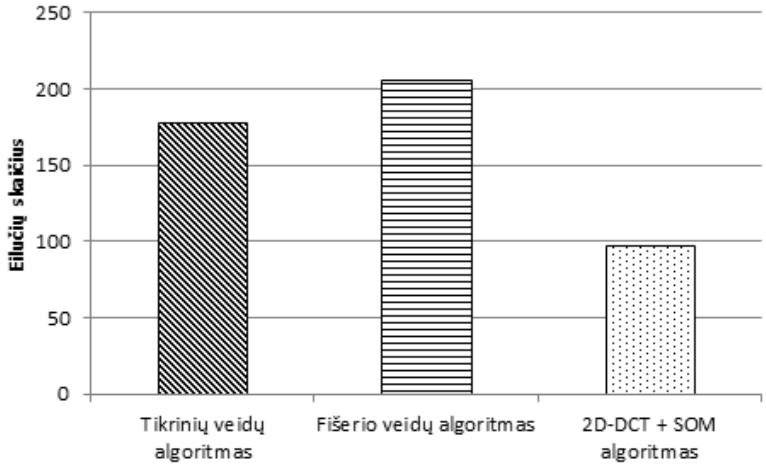
Įgyvendinus visus tris pasirinktus algoritmus *MATLAB* aplinkoje galima palyginti jų sudėtingumą bei užimamą vietą kietajame diske. Kadangi *MATLAB* aplinkoje pirminis programos tekstas nėra kompiliuojamas, o yra interpretuojamas paeiliui kaip skriptas – galima daryti išvadą kad programos užimama vieta kietajame diske tiesiogiai priklauso nuo simbolių skaičiaus pirminiame kode. Kaip matyti iš 4.1–4.3 grafikų, 2D–DCT+SOM algoritmas užima mačiausiai vietos kietajame diske (tik 4 KB) bei yra trumpiausias tekstiniu požiūriu. Šis algoritmas yra dvigubai lengvesnis nei tikrinių veidų algoritmas, ir trigubai lengvesnis už Fišerio veidų algoritmą. Tuo tarpu Fišerio algoritmo įgyvendinimas pareikalavo daugiausiai pastangų. Šiam algoritmui įgyvendinti prireikė parašyti 206 eilutes kodo (atitinkamai 8903 simboliai).

Kadangi ištyrus šiuos tris algoritmus vienas jų bus įgyvendintas *Android* operacinėje sistemoje, šių algoritmų sudėtingumą vertinti vien pagal simbolių skaičių yra netinkama. *MATLAB* aplinka yra skirta atlikti įvairiems tyrimams , kai tuo tarpu *Android* operacinė sistema yra pagrįsta *JAVA* programavimo kalba. Perkeliant algoritmą į *Android* operacinę sistemą galima susidurti su įvairių funkcijų stygiumi naujoje aplinkoje, tad vertėtų atsižvelgti ir į tai kokios specifinės funkcijos yra naudojamos šiuose algoritmuose.

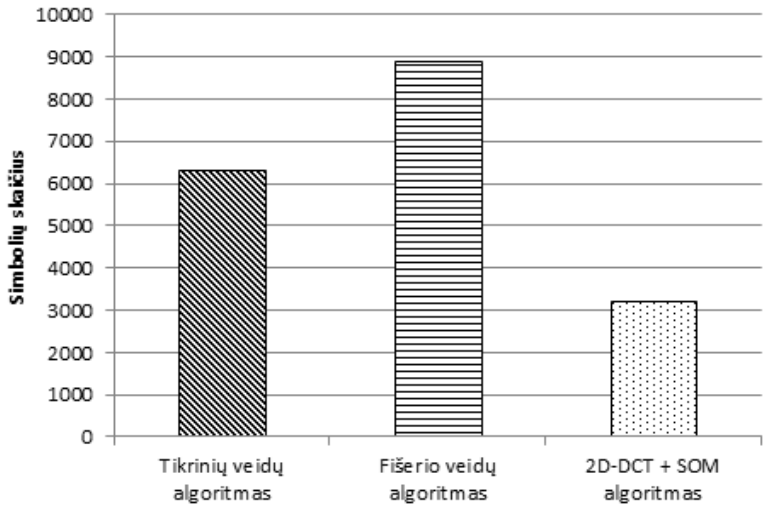
4.1 lentelėje yra pateiktos nestandartinės funkcijos kurias naudoja kiekvienas algoritmas. Šių funkcijų įgyvendinimas *Android* operacinėje sistemoje gali sukelti tam tikrų keblumų. Kaip matyti iš šios lentelės, dauguma funkcijų kartojasi visuose algoritmuose. Mažiausiai nestandartinių funkcijų reikalauja 2D–DCT algoritmas, o daugiausiai – Fišerio veidų algoritmas. Tikrinių veidų algoritmas tiek savo apimtimi, tiek ir įgyvendinimo sudėtingumu, yra tarpinis variantas tarp likusių dviejų algoritmų.



4.1 pav. Nekompiliuoto kodo dydis kilobaitais



4.2 pav. Eilučių skaičius pirminiame programos tekste



4.3 pav. Simbolių skaičius pirminiame programos tekste

4.1 lentelė Specifinės funkcijos algoritmuose

Fišerio veidų algoritmas	**Tikrinių veidų algoritmas**	**2D–DCT+SOM algoritmas**
rgb2gray()	*rgb2gray()*	*rgb2gray()*
norm()	*norm()*	*dct2()*
eig()	*eig()*	*idct2()*
mean()	*mean()*	*newsom()*
reshape()	*reshape()*	*sim()*
Operacijos su matricomis	*Operacijos su matricomis*	
std()	*fliplr()*	
	repmat()	
	circshift()	

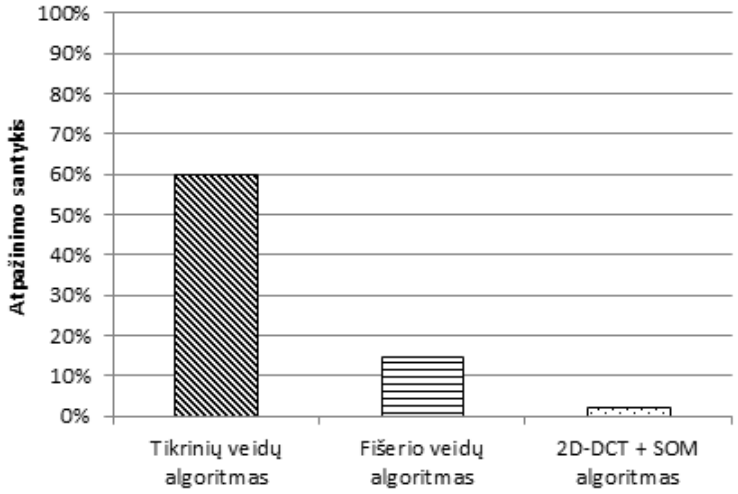
4.2. Algoritmų tikslumas skirtingose duomenų bazėse

Turbūt pats svarbiausias kriterijus kalbant apie veidų atpažinimo algoritmus yra jų tikslumas. Šio skyriaus grafikuose yra pateikti visų trijų algoritmų atpažinimo tikslumai skirtingose duomenų bazėse. Kaip matyti iš 4.4 grafiko, prasčiausi rezultatai yra gaunami AT&T duomenų bazėje — čia 2D–DCT+SOM algoritmo atpažinimo santykis tėra lygus 3 %, kas greičiausiai buvo įtakota to, kad veidai šioje duomenų bazėje yra nufotografuoti kintant apšvietimo lygiui bei jų padėčiai. Šioje duomenų bazėje geriausias rezultatas yra pasiektas tikrinių veidų algoritmo.

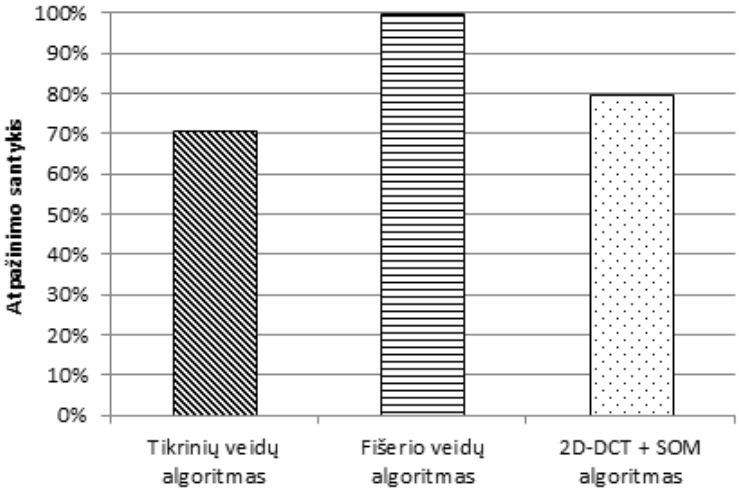
Grimasų veidų duomenų bazėje (4.5 grafikas) visų algoritmų atpažinimo santykis yra kur kas geresnis. Čia mažiausias atpažinimo santykis yra 71 % (tikrinių veidų algoritmas), 2D–DCT algoritmas pasiekė 80 %, o Fišerio veidų algoritmas visus asmenis identifikavo 100 % tikslumu.

Tuo tarpu mano sudarytoje duomenų bazėje yra gaunami geriausi rezultatai. Šioje duomenų bazėje visi algoritmai pasiekė didesnį nei 85 % atpažinimo santykį – geriausiai pasirodė 2D–DCT algoritmas, kurio atpažinimo santykis buvo lygus 96 %, tikrinių veidų – 85 %, o Fišerio veidų – 90 %.

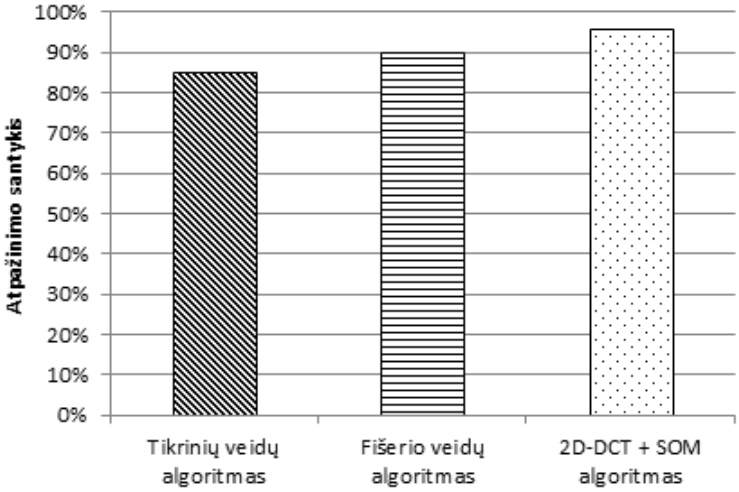
Apibendrinant 4.4–4.6 grafikus galima teigti, jog visose trijose veidų duomenų bazėse stabiliausiai veikė tikrinių veidų algoritmas – jo atpažinimo santykis svyravo nuo 60 % iki 85 %, kai tuo tarpu kitų algoritmų atpažinimo santykis krisdavo net iki 3 % teisingo atpažinimo ribos.



4.4 pav. Algoritmų tikslumas AT&T duomenų bazėje



4.5 pav. Algoritmų tikslumas Grimasų duomenų bazėje



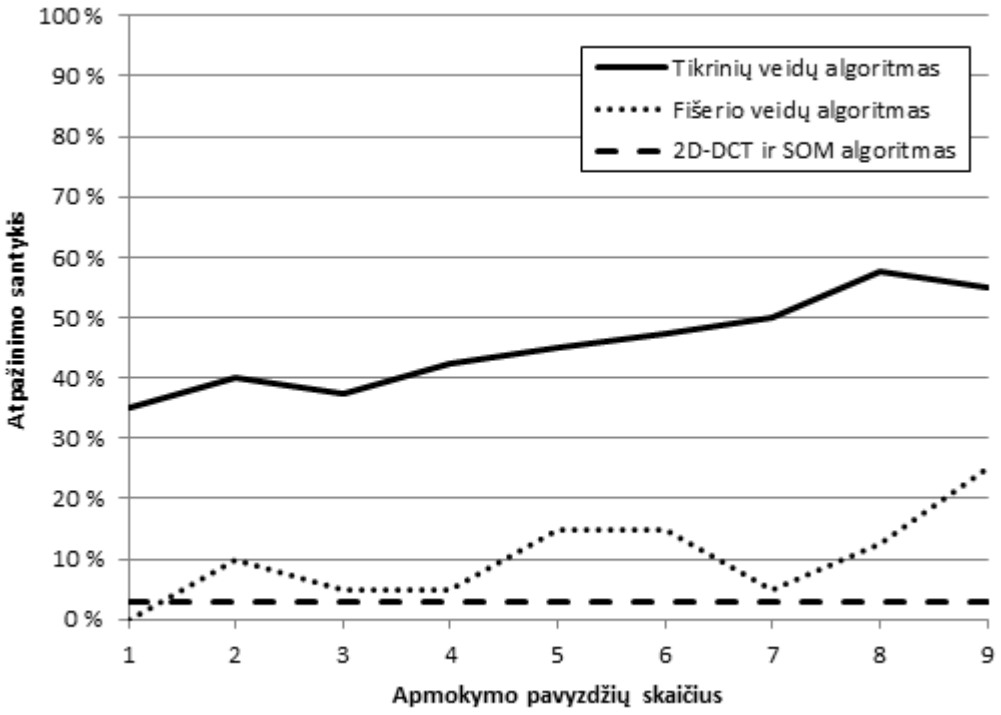
4.6 pav. Algoritmų tikslumas Mano duomenų bazėje

4.3. Algoritmų tikslumo priklausomybė nuo pavyzdžių skaičiaus vienam asmeniui

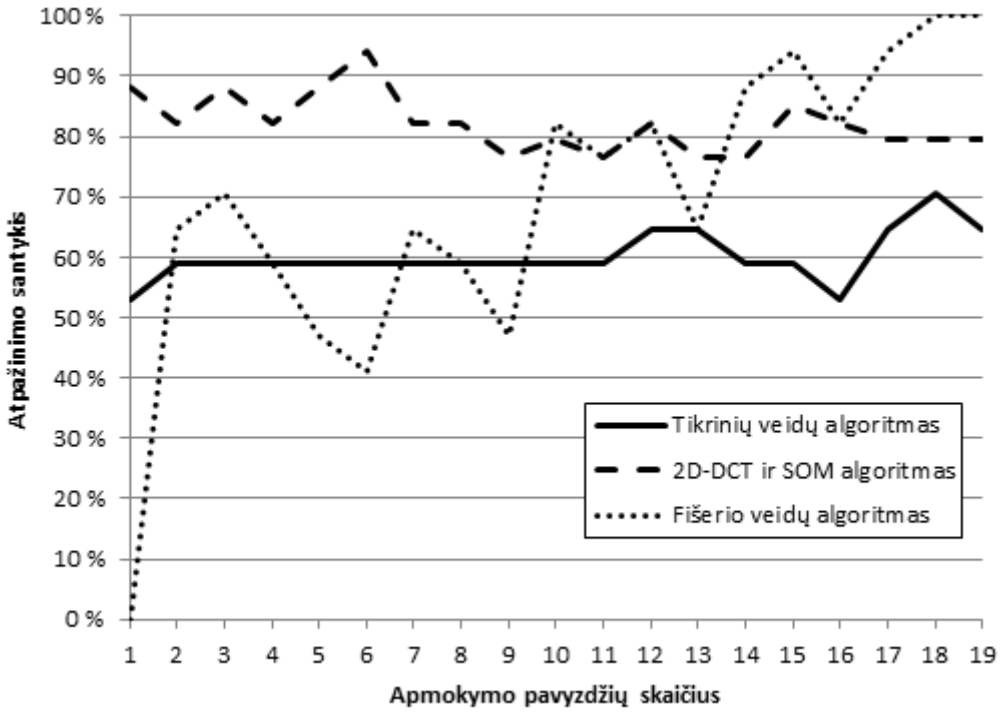
Sekančiame etape buvo ištirta kaip šių trijų algoritmų atpažinimo santykį įtakoja nuotraukų skaičius vienam asmeniui duomenų bazėje. Tai taip pat yra gan svarbus parametras, nes didelis pavyzdžių skaičius reikalauja daugiau resursų – reikalinga daugiau vietos kietajame diske, bei galima daryti prielaidą, kad ir algoritmo apmokymas turėtų trukti ilgiau bei reikalauti daugiau resursų iš centrinio procesorinio įrenginio. Dėl nepakankamo pavyzdžių skaičiaus vienam asmeniui Mano duomenų bazėje, buvo pasirinktos tik AT&T ir Grimasų duomenų bazės.

Iš 4.7 grafiko matyti, jog AT&T duomenų bazėje didinat nuotraukų priklausančių tam pačiam asmeniui skaičių nuo 1 iki 9 atpažinimo santykis vienodai kyla tikrinių ir Fišerio veidų algoritmuose. Tuo tarpu iš 4.8 grafiko galima pastebėti, jog tolygus kylimas Grimasų duomenų bazėje yra pastebimas tik Fišerio veidų algoritme. Pagal tikrinių veidų algoritmo kreivę galima teigti, jog šioje duomenų bazėje vieno asmens pavyzdžių skaičius įtakoja algoritmą tik tuomet, kai tų pavyzdžių skaičius peržengia 10 nuotraukų. Tuo tarpu 2D–DCT+SOM algoritmui pavyzdžių skaičius didelės įtakos atpažinimo santykiui neturi nei vienoje duomenų bazėje.

Iš 4.7–4.8 grafikų galima padaryti išvadą, jog geriausi rezultatai yra gaunami tuomet, kai vienas asmuo turi daugiau nei 15 save apibūdinančių nuotraukų, o esant mažam



4.7 pav. Algoritmų atpažinimo santykio priklausomybė nuo pavyzdžių kiekio AT&T duomenų bazėje



4.8 pav. Algoritmų atpažinimo tikslumo priklausomybė nuo pavyzdžių kiekio Grimasų duomenų bazėje

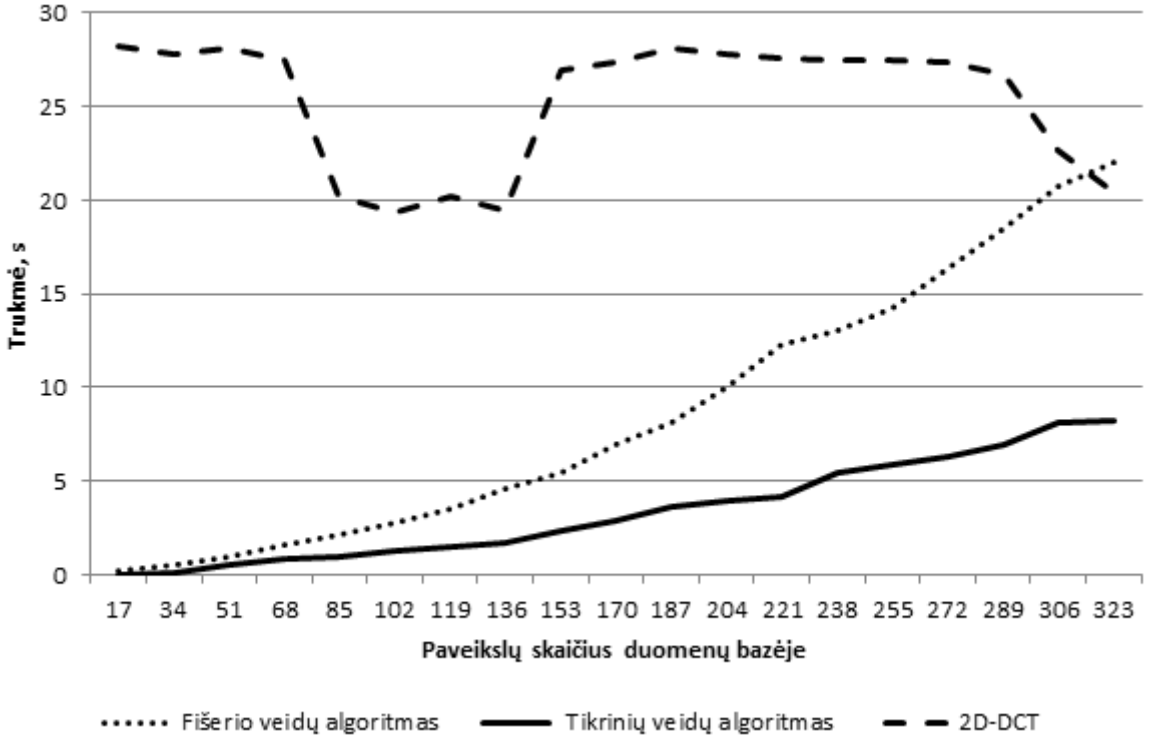
pavyzdžių skaičiui (nuo 1 iki 9) abejose veidų duomenų bazėse geriausiai veikė tikrinių veidų algoritmas. Vertėtų pabrėžti, jog pavyzdžių skaičius nėra esminis parametras – iš šių dviejų grafikų matyti, jog vienoje duomenų bazėje atpažinimo santykis yra daugiau priklausomas nuo pavyzdžių skaičiaus nei kitoje, tad svarbiausia, kad tie pavyzdžiai būtų geros kokybės.

4.4. Trukmės priklausomybė nuo paveikslų kiekio

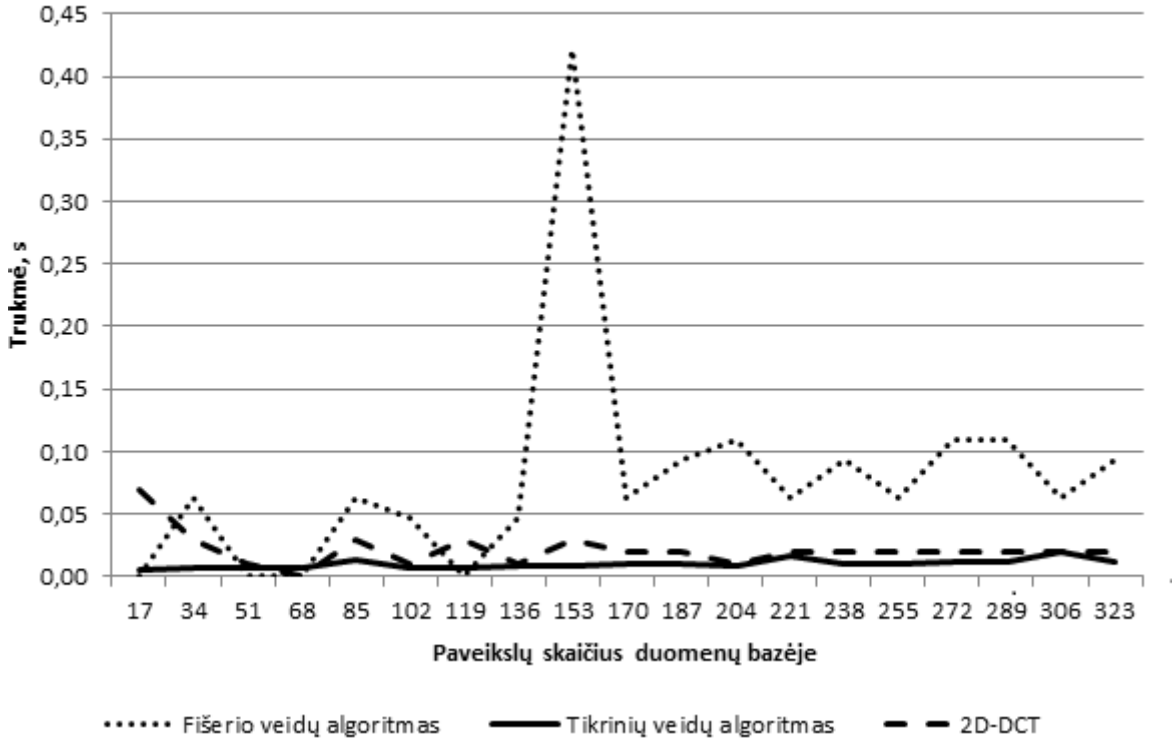
Praeitame poskyryje išsiaiškinus optimalų pavyzdžių skaičių vienam asmeniui, iškyla klausimas kaip pavyzdžių skaičius įtakoja algoritmų apsimokymo bei atpažinimo trukmes. Siekiant atsakyti į šį klausimą buvo ištirta laiko priklausomybė nuo nuotraukų kiekio.

4.9 grafike yra pateikta apsimokymo trukmės priklausomybė nuo pavyzdžių skaičiaus, kur bendras nuotraukų skaičius duomenų bazėje kinta nuo 17 iki 323. Kaip matyti iš grafiko, pavyzdžių skaičius mažiausiai įtakoja 2D–DCT+SOM algoritmo apsimokymo trukmę, kuri palyginus su likusiais dviem algoritmais yra didžiausia. Tikrinių ir Fišerio veidų algoritmų apsimokymo trukmė yra tiesiogiai proporcinga nuotraukų skaičiui, o sparčiausiai apsimoko tikrinių veidų algoritmas (du kartus greičiau už Fišerio veidų algoritmą).

Už apsimokymo trukmę kur kas svarbesnis parametras yra atpažinimo trukmė. 4.10 grafike yra pateikta visų trijų algoritmų atpažinimo trukmės priklausomybė nuo apmokymo pavyzdžių skaičiaus. Iš šio grafiko galima pastebėti, jog pavyzdžių skaičius įtakoja



4.9 pav. Apsimokymo trukmės priklausomybė nuo paveikslų skaičiaus duomenų bazėje



4.10 pav. Atpažinimo trukmės priklausomybė nuo paveikslų skaičiaus duomenų bazėje

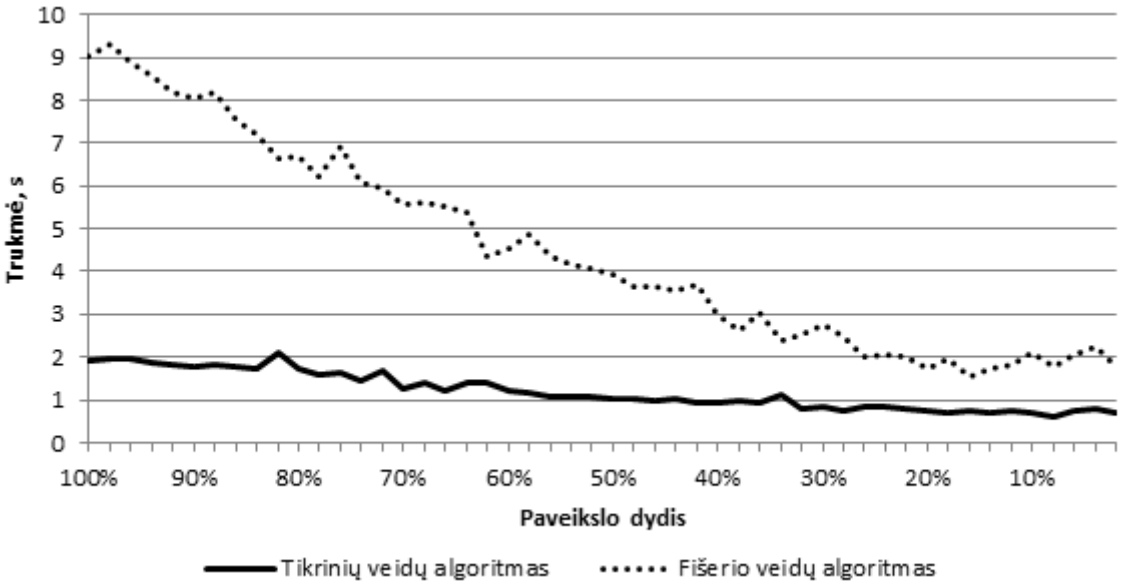
tik Fišerio veidų algoritmą, kai tuo tarpu likusių dviejų algoritmų atpažinimo trukmė praktiškai nekinta. Tikrinių veidų algoritmo kreivė šiame grafike yra labiausiai statiška ir šis algoritmas atpažinti veidus sugeba sparčiausiai.

4.5. Trukmės priklausomybė nuo paveikslų dydžio

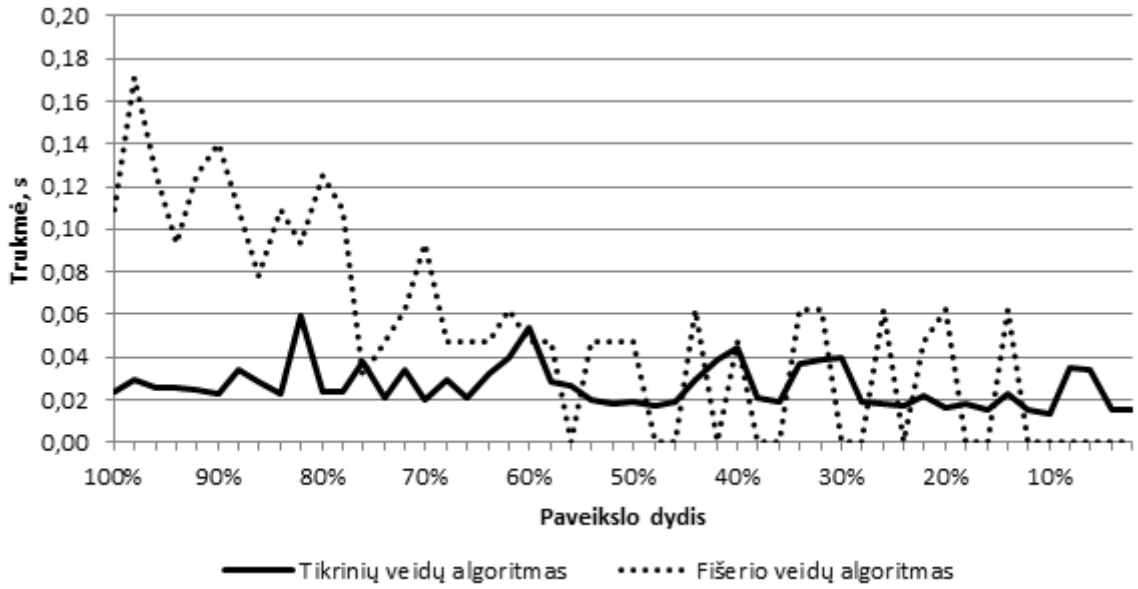
Šių trijų algoritmų spartą įtakoja ne tik paveikslų skaičius, bet ir jų dydis pikseliais. Nustačius optimalų paveikslo dydį, bus galima paspartinti algoritmo veikimą arba parinkti tokį paveikslo dydį, kad algoritmas sugebėtų veikti realių laiku. Šio tyrimo metu paveikslo dydis buvo mažinamas 2 % žingsniu nuo originalaus dydžio iki 2 % paveikslo dydžio ir išnagrinėta kaip šis pokytis įtakoja apsimokymo bei atpažinimo trukmes. 2D–DCT+SOM algoritmas buvo netirtas, nes šis algoritmas apsimokymui ir taip sumažina paveikslą iki minimalios ribos siekiant spartumo.

Iš 4.11 grafiko matyti, jog paveikslo dydis tiesiogiai įtakoja apsimokymo trukmę. Esant originaliam paveikslo dydžiui tikrinių veidų algoritmas sugeba apsimokyti 4,5 karto greičiau už Fišerio veidų algoritmą, o mažinant paveikslo dydi šis skirtumas mažėja iki 2 kartų. Tiek šis, tiek ir 4.9 grafikas dar kartą patvirtina, jog sparčiausiai apsimokyti sugeba tikrinių veidų algoritmas.

4.12 grafike yra pateikta atpažinimo trukmės priklausomybė nuo paveikslo dydžio. Iš šio grafiko galima pastebėti, jog paveikslo dydis tikrinių veidų algoritmo atpažinimo trukmę (kaip ir apsimokymo trukmę) įtakoja mažiau nei Fišerio veidų algoritmą. Taip pat, iš šio grafiko galima teigti, kad Fišerio veidų apsimokymo trukmė stabilizuojasi kai paveikslas yra mažesnis nei 60 % originalaus jo dydžio (120 × 108 px), kai tuo tarpu tikrinių veidų algoritmo atpažinimo trukmė prie visų paveikslų dydžių praktiškai nekito.



4.11 pav. Apsimokymo trukmės priklausomybė nuo paveikslų dydžio



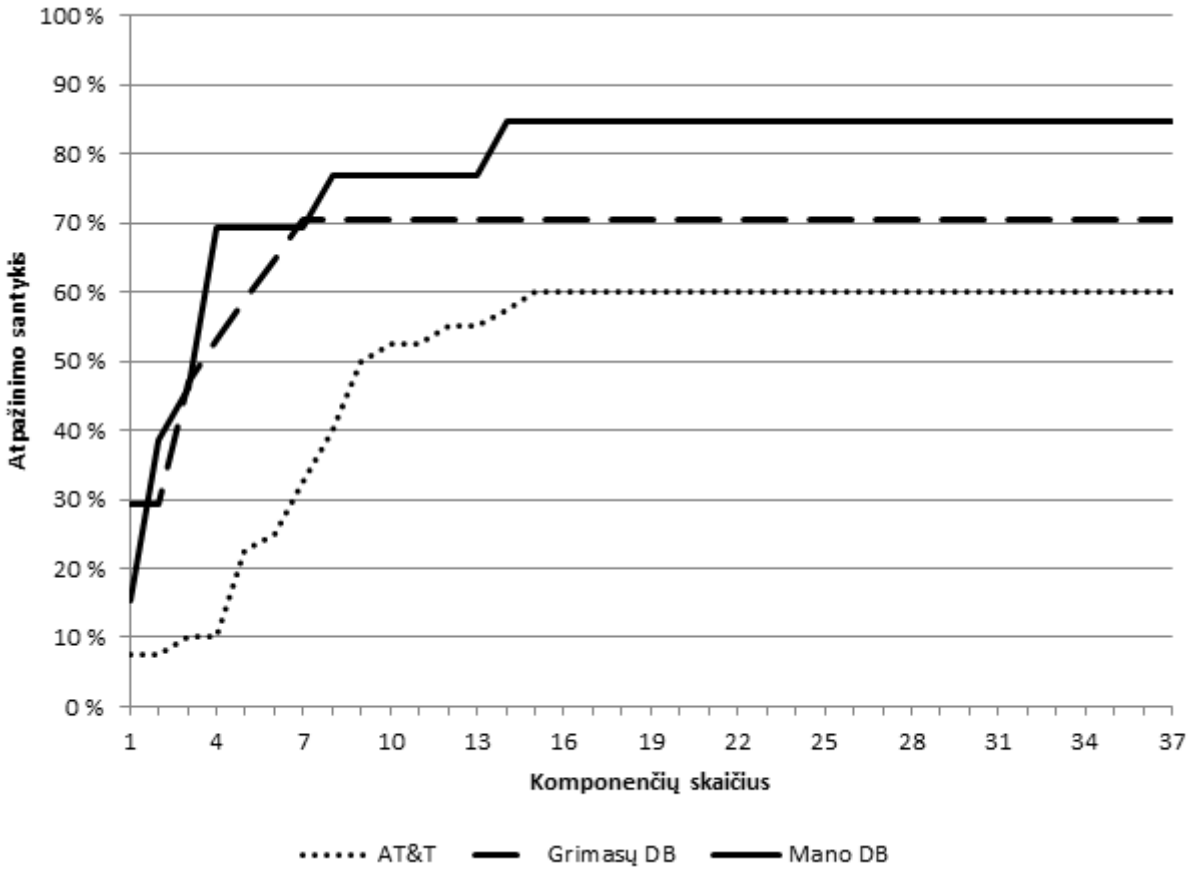
4.12 pav. Atpažinimo trukmės priklausomybė nuo paveikslų dydžio

4.6. Tikrinių veidų algoritmo priklausomybė nuo komponenčių skaičiaus

Tikrinių veidų algoritme pagrindinis parametras lemiantis atpažinimo tikslumą yra komponenčių skaičius naudojamas kiekvieno veido tikriniams vektoriams sudaryti. Naudojant per mažai komponenčių atpažinimo tikslumas bus prastas, o naudojant per daug komponenčių – gali nukentėti algoritmo sparta.

Siekiant nustatyti optimalų komponenčių skaičių buvo atliktas tyrimas visose duomenų bazėse. Kaip matyti iš 4.14 grafiko, pasiekti geriausią rezultatą Grimasų duomenų bazėje pavyko turint tik 7 komponentes, o likusiose dvejuose duomenų bazėse prireikė 14 ir 15 komponenčių. Literatūroje yra teigiama, jog optimalus komponenčių skaičius yra lygus 20, tačiau kaip matyti iš tyrimo, šiose duomenų bazėse pilnai pakanka ir 15 komponenčių. Naudojant daugiau nei 15 komponenčių sistemoje nepastebimi jokie tikslumo pokyčiai.

Išsiaiškinus optimalų komponenčių skaičių lemiantį atpažinimo tikslumą iškyla klausimas kaip komponenčių kiekis įtakoja algoritmo veikimo sparą. 4.14 grafike yra pateikta apsimokymo trukmės priklausomybė nuo komponenčių skaičiaus visose duomenų bazėse, čia abscisių ašyje yra pateiktas komponenčių skaičius, o ordinačių ašyje — laikas kurį sistema sugaišta apsimokymo procese. Kaip matyti iš šio grafiko, naudojamų komponenčių skaičius neturi jokios įtakos apmokant algoritmą. Vienas nežymus laiko šuolis AT&T veidų duomenų bazėje ties 6 – 11 komponentėmis yra matomas dėl tam tikrų pašaliniu kompiuterio apkrovos trukdžių apmokymo metu. Analogiški rezultatai yra gaunami ir algoritmo testavimo procese – komponenčių skaičius algoritmo testavimo spartai neturi visiškai jokios įtakos.



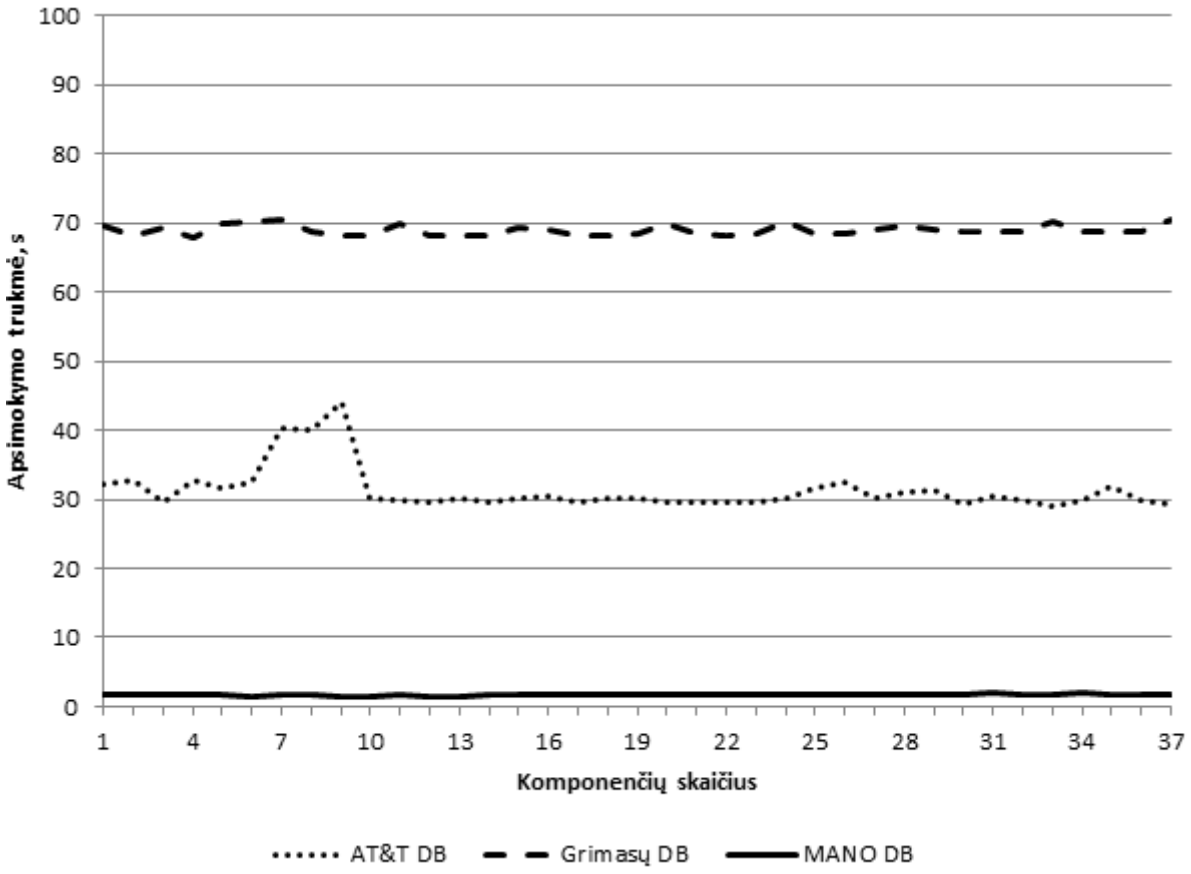
4.13 pav. Tikrinių veidų algoritmo atpažinimo santykio priklausomybė nuo komponenčių skaičiaus

4.7. Skyriaus apibendrinimas

Šiame skyriuje buvo įgyvendinti ir ištirti tikrinių, Fišerio veidų bei 2D–DCT algoritmai trijose veidų duomenų bazėse. Visų šių algoritmų pirminis programos tekstas yra pateiktas D priede. Iš 4.2 lentelės matyti, jog visi algoritmai sugeba puikiai atpažinti veidus. Geriausias rezultatas buvo gautas Fišerio veidų algoritmo, kuris teisingai atpažino visus testuos veidus Grimasų veidų duomenų bazėje. Tuo tarpu kiti algoritmai geriausius rezultatus pasiekė Mano duomenų bazėje – tikrinių veidų algoritmas sugebėjo teisingai atpažinti 85 % testuojamų veidų, o 2D–DCT+SOM algoritmas net 95 %. Palyginus al-

4.2 lentelė Algoritmų atpažinimo santykiai

Algoritmas	**Atpažinimo santykis**	**Duomenų bazė**
Tikrinių veidų	85%	Mano DB
Fišerio veidų	100%	Grimasų DB
SOM + 2D–DCT	95%	Mano DB



4.14 pav. Tikrinių veidų algoritmo apsimokymo trukmės priklausomybė nuo komponenčių skaičiaus

goritmų gautus rezultatus visose trijose duomenų bazėse, geriausi atpažinimo santykiai buvo pasiekti tikrinių veidų algoritmu.

Taip pat, tikrinių veidų bei Fišerio algoritmuose buvo pastebėta, jog didinant apmokymo pavyzdžių skaičių gerėja ir atpažinimo santykis. Tuo tarpu 2D–DCT+SOM algoritme buvo pastebėta, jog didinant pavyzdžių skaičių atpažinimo santykis nežymiai mažėja. Geriausias atpažinimo santykis pasiekiamas kai vienas asmuo turi daugiau nei 15 save apibūdinančių nuotraukų

Išsiaiškinus kiekvieno algoritmo atpažinimo santykius buvo ištirta ir algoritmų apsimokymo bei atpažinimo spartos priklausomybės nuo apmokymui naudojamų pavyzdžių skaičiaus bei jų dydžio. Iš gautų grafikų matyti, jog apsimokymo trukmė yra tiesiogiai proporcinga naudojamo paveikslo dydžiui bei jų skaičiui – kuo paveikslų daugiau ar jų matmenys didesni – tuo ir apsimokymas trunka ilgiau. Tuo tarpu atpažinimo trukmei šie parametrai įtakos praktiškai neturi. Siekiant optimizuoti algoritmo apsimokymo trukmę, abu šiuos parametrus reikia derinti tarpusavyje – norint didinti vieną, derėtų atitinkamai mažinti kitą. Paveikslų kiekis ir raiška mažiausiai įtakoja tikrinių veidų algoritmą, kuris sugeba veidus atpažinti sparčiausiai – jis užtrunka apie 10 ms. Tuo tarpu veidų atpažinimas 2D–DCT+SOM algoritme trunka apytiksliai 20 ms, o Fišerio veidų algoritme jis gali

trukti netgi virš 200 ms.

Apibendrinant visus šiuos algoritmus galima teigti, jog sparčiausias ir tiksliausias algoritmas yra Tikrinių veidų. Jis sugeba puikiai identifikuoti asmenis netgi esant mažam apmokymo pavyzdžių skaičiui. Taip pat, šio tyrimo metu buvo išsiaiškinta, jog tikrinių veidų algoritmui optimalus komponenčių skaičius tėra viso labo 15. Šis algoritmas tiek savo sudėtingumu, tiek ir pirminio programos teksto apimtimi yra tarpinis variantas tarp likusių dviejų algoritmų. Šiame algoritme yra naudojama mažiau specifinių funkcijų nei Fišerio veidų algoritme, bei jis nereikalauja bibliotekų skirtų darbui su dirbtinių neuronų tinklais.

5. ALGORITMO ĮGYVENDINIMAS *ANDROID* OPERACINĖJE SISTEMOJE

Šiame skyriuje bus aptartas veidų atpažinimo algoritmo įgyvendinimas *Android* operacinėje sistemoje. Kadangi iš praeito skyriaus aišku, jog tinkamiausias algoritmas yra tikrinių veidų – tad šis algoritmas ir bus įgyvendinamas.

Android operacinė sistema yra paremta *Linux* branduoliu su *JAVA* kalbos programavimo sąsaja. Tai reiškia, jog *Android SDK* (pilnas pavadinimas angl. *Android Software Development Kit*) yra skirtas *JAVA* programavimo kalbai, nors yra įmanoma programą rašyti ir C/C++ programavimo kalbomis naudojant *Android NDK* (pilnas pavadinimas angl. *Android Native Development Kit*).

Kiekviena *Android* programa veikia atskirame *Linux* procese, kuris yra pradedamas jei nors viena kodo dalis reikalauja vykdymo ir užbaigiamas gavus rezultatą arba sistemai pritrūkus resursų. Kiekvienas šių procesų turi individualią *JAVA* virtualią mašiną (*JAVA VM*), tad programos kodas visuomet yra vykdomas izoliuotoje nuo kitų programų aplinkoje.

Pagrindiniai terminai, susiję su *Android* operacine sistema ir vartojami šiame skyriuje yra veikla (angl. *activity*), sąsaja (angl. *view*) ir tikslinė veikla (angl. *intent*). Veikla – tai tarsi vienas programos sluoksnis reprezentuojantis vieną ekrano vaizdą. Tarkime, viena veikla gali būti skirta telefonų knygos kontaktams atvaizduoti, o norint peržiūrėti konkretų kontaktą – būtų iškviečiama nauja veikla, kurios vaizdas ekrane kardinaliai (arba ne) skirtųsi. Sąsaja – tai kiekvienos veiklos vartotojo sąsaja, kuri yra dėliojama iš atskirų elementų su nustatytu išsidėstymu (angl. *ViewGroups*). Tikslinė veikla – tai asinchroninės žinutės, kurių pagalbą galima iškviesti kitas veiklų tipo programas ar servisus bei pareikalauti iš jų atitinkamo atsakymo.

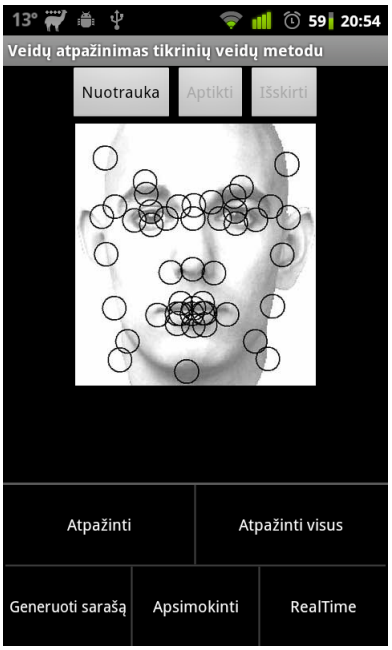
Šio skyriaus tikslas yra įgyvendinti prieš tai pasirinktą veidų atpažinimo algoritmą *Android* 2.3.3 (API 10) operacinėje sistemoje. Algoritmui perduodamas veidas bus gautas iš telefone esančios foto–kameros arba, pasirinktinai, iš telefono atminties kortelės. Taip pat, prieš atpažįstant veidą, yra būtina jį išskirti iš nuotraukos. Ši dalis taip pat bus įgyvendinta mobiliojoje platformoje.

5.1. Vartotojo sąsajos kūrimas

Kiekvienos veiklos vartotojo sąsaja yra apibūdinama per maketus, kurie yra saugomi XML tipo bylose „/res/Layouts“ projekto direktorijoje. Šiuose maketuose galime nesunkiai pridėti naujų elementų, pavyzdžiui mygtukų, teksto laukų ir panašiai, kuriuos vėliau, nepriklausomai nuo programos kodo, galima pertvarkyti. Pertvarkymo metu lengvai galima keisti tiek jų išsidėstymą, tiek ir kiekvieno elemento individualius nustatymus.

Programos kodas 5.1 Programos resursų apibrėžimas

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <string name="app_name">Veidu atpazinimas tikriniu veidu metodu
      </string>
4   <string name="foto">Nuotrauka</string>
5   <string name="detect">Aptikti</string>
6   <string name="extract">Isskirti</string>
7   <string name="save">Issaugoti i DB</string>
8 </resources>
```



5.1 pav. Programo vartotoja sąsaja

Žinoma, visi vartotojo sąsajos elementai gali būti aprašyti ir pačiame programiniame kode, tačiau maketų naudojimas yra žymiai pranašesnis metodas – jis įgalina programuotoją žymiai greičiau ir lengviau pertvarkyti visą vartotojo sąsają pagal naujus reikalavimus. Jei kuris nors maketo objektas turi būti pasiekiamas per programinį kodą, tuomet tam objektui makete turi būti priskirtas unikalus identifikatorius *android:id=„pavadinimas“*, pagal kurį programiniame kode jis bus iškviestas funkcija *findViewById(R.id.pavadinimas)*.

Kaip ir kiekvienos veiklos sąsajos, taip ir visi programos resursai (teksto eilutės, pavadinimai, paveikslėliai ir pan.) turėtų būti saugomi atitinkamose „/res/“ direktorijose į kurias galėtų kreiptis vartotojo sąsaja, o ne tiesiogiai įprogramuoti į pačią vartotojo sąsają. Tai yra geriausia praktika, kuria vadovaujantis galima nesunkiai atlikti įvairiausius pakitimus ar pritaikyti programą kitai kalbai. Toks resursų apibrėžimo pavyzdys yra pateiktas 5.1 kodo ištraukoje.

Programos kodas 5.2 Veiksmų priskyrimas vartotojo sąsajos objektams

```
1 View.OnClickListener myListener = new View.OnClickListener() {
2   public void onClick(View v) {
3     // Veiksmai, kuriuos reikia atlikti
4   }
5 };
```

5.2 pav. Būdai skirti nuotraukai gauti

Šiai programai sudaryta minimali vartotojo sąsaja yra pateikta 5.1 paveiksle. Kaip matyti iš šio paveikslo, vartotojo sąsaja susideda iš pavadinimo, paveikslėlio ir mygtukų. Visi šie elementai, kaip minėta anksčiau, yra apibrėžti „/res/“ direktorijoje. Paveikslo apačioje galima pastebėti meniu, kuris yra aprašytas atskirame vartotojo sąsajos makete.

Android operacinėje sistemoje, norint jog tam tikras vartotojo sąsajos objektas, pavyzdžiui mygtukas ar tam tikras meniu punktas, atliktu kokius nors veiksmus jį palietus – yra naudojamos tokios funkcijos kaip *View.OnClickListener()*, *View.OnLongClickListener()*, *View.OnTouchListener()* ir t. t.. Visi veiksmai, kurie turi būti atlikti numatytu atveju, yra aprašomi *onClick()*, *onLongClick()* ir t. t. metoduose. Paprasčiausias pavyzdys yra pateiktas 5.2 kodo ištraukoje.

5.2. Pradinės nuotraukos gavimas

Sukūrus programos pagrindinio lango dizainą buvo pereita prie sekančio žingsnio – vaizdo, kuris bus apdorojamas, gavimo. Šioje programoje, kaip matyti iš 5.2 paveikslo yra

Programos kodas 5.3 Deklaracija, jog programa naudos vaizdo kamera bei išorinę duomenų laikmeną

```
1 <uses-permission android:name="android.permission.CAMERA" />
2 <uses-feature android:name="android.hardware.camera" />
3 <uses-permission android:name="android.permission.
    WRITE_EXTERNAL_STORAGE" />
```

įgyvendinti trys metodai skirti gauti pradiniam vaizdui. Pirmieji du metodai vaizdui gauti naudoja telefone įmontuotą vaizdo kamerą, o paskutinysis – nuotrauką įkelia iš galerijos (atminties kortelės). Pirmieji du metodai tarpusavyje skiriasi tuo, jog pirmasis naudoja gamyklinę foto–kameros programą, kuri yra iškviečiama kaip tikslinė veikla, kai tuo tarpu antrasis metodas iškviečia naują foto–kameros veiklą kuri yra dalis šios programos.

Gamyklinės foto–kameros aplikacijos naudojimas yra kur kas paprastesnis – užtenka ją iškviesti ir po iškvietimo yra gaunamas rezultatas, tačiau įgyvendinant naują foto–kameros veiklą galima ją individualiai pritaikyti savo reikmėms. Šiuo atveju kameros programa buvo perrašyta pridedant jai papildomą funkcionalumą – veidų aptikimą ir jų sužymėjimą ekrane realiu laiku.

Visų pirma, prieš gaunant pradinį vaizdą iš foto–kameros, yra būtina *AndroidManifest.xml* byloje apibrėžti reikalaujamas teises bei ypatybes (žr. 5.3 kodo ištrauką). Pirmojoje eilutėje yra nurodoma, jog programa naudos foto–kamerą, o antrojoje – jog naudos kameros ypatybes, pavyzdžiui automatinį fokusavimą, apšvietimą ir panašiai. Trečioji kodo ištraukos eilutė apibrėžia, jog programa gali rašyti duomenis į išorinę telefono atminties laikmeną.

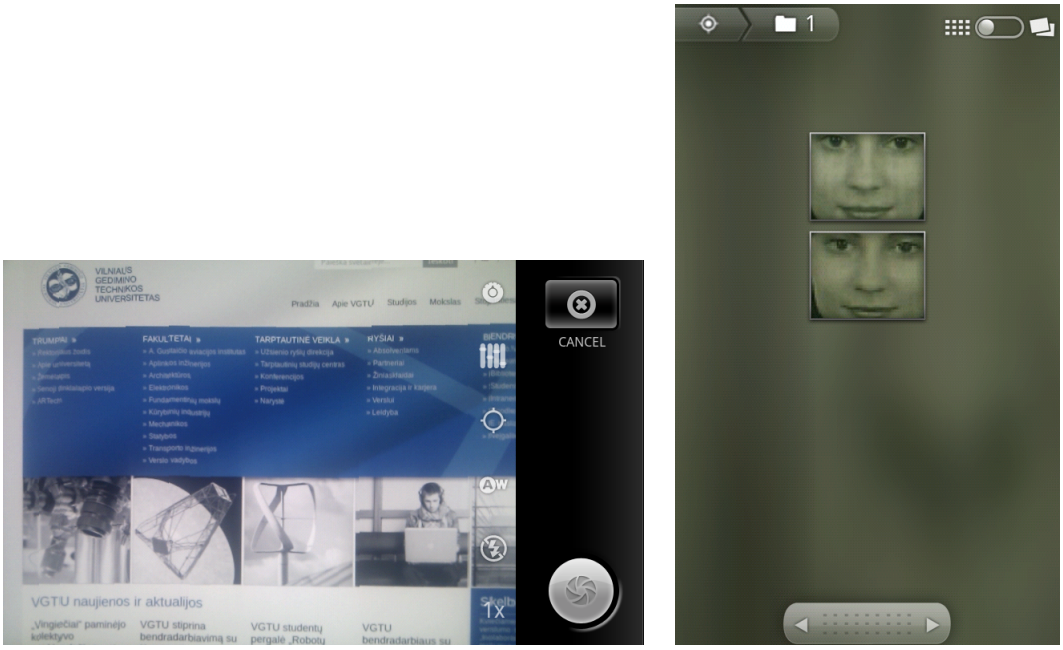
Tiek gamyklinės kameros veiklos, tiek ir galerijos veiklos iškvietimas vyksta per tikslines veiklas. Iškvietus tikslines veiklas, jos perima kontrolę iš pagrindinės programos, o baigusios darbą grąžina kontrolę kartu su rezultatu. 5.3 paveiksle yra pavaizduotos jau iškviesto gamyklinės foto–kameros be galerijos tikslinės veiklos. Tikslinių veiklų iškvietimo procedūra susideda iš šių pagrindinių žingsnių:

1. Objekto, iškviečiančio tikslinę veiklą sukūrimo. Objektui turi būti nurodoma, ką tikimasi iš jo gauti kaip rezultatą. Foto nuotraukoms gauti reikia nurodyti

   MediaStore.ACTION_IMAGE_CAPTURE tipą.

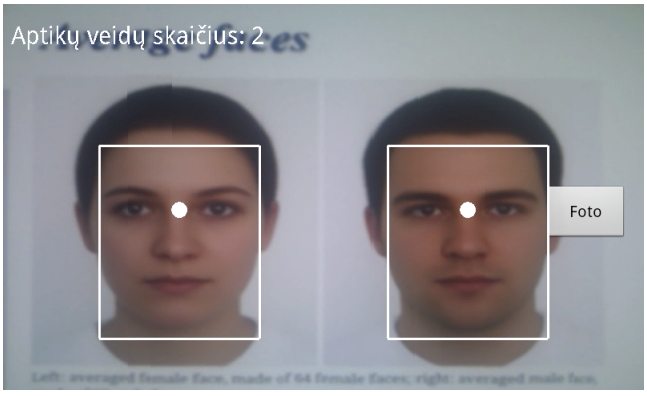
2. Tikslinės veiklos paleidimo. Ji yra paleidžiama iškvietus *startActivityForResult()* metodą, kuriam startavus nauja tikslinė veikla perima kontrolę.

3. Rezultatų gavimas. Kai vartotojas baigia naudoti tikslinę veiklą, yra iškviečiamas *onActivityResult()* metodas, kuris grąžina gautus duomenis.



a) *b)*

5.3 pav. Tikslinių veiklų iškvietimas: *a)* gamyklinės foto–kameros ir *b)* galerijos



5.4 pav. Naujos vaizdo kameros aplikacijos su veidų aptikimo funkcija nuotrauka

Naują foto–kameros aplikaciją yra verta kurti tik tuo atveju jei yra planuojama įgyvendinti išskirtinę vartotojo sąsają ar naujas funkcijas, šiuo atveju – veidų aptikimą vaizdo sraute. Kaip matyti iš 5.4 paveikslo, naujoji vaizdo kameros aplikacija realiu laiku sugeba aptikti veidus ir juos sužymėti. Vartotojui yra pateiktas apdorotas vaizdo kadras, kuriame aptikti veidai yra pažymėti stačiakampiu, o ekrano kampe yra pateikiama informacija apie aptiktų veidų kiekį. Naujos vaizdo kameros aplikacijos kūrimas susideda iš šių pagrindinių etapų:

1. Kameros aptikimas ir jos rezervavimas įrenginyje. Visų pirma yra patikrinama ar prietaisas turi vaizdo kamerą ir kiek jų turi, vėliau yra reikalaujama priėjimo prie pasirinktos vaizdo kameros.

2. Sukuriama atvaizdavimo klasė *Preview*. Ši klasė išplečia *SurfaceView* klasę ir įgyvendiną *SurfaceHolder* sąsają. Ji yra skirta atvaizduoti kintantį vaizdo srautą gaunamą iš vaizdo kameros.

3. Visi programos objektai susiejami su atitinkamais vartotojo sąsajos elementais ir nustatomi objektų klausytojai. Taip pat, objektų klausytojams yra priskiriami atitinkami veiksmai, kurie turi būti atlikti įvykus nustatytam įvykiui, pavyzdžiui mygtuko paspaudimui.

4. Apdorojamas kiekvienas vaizdo kadras gaunamas iš video kameros. Įvykdžius numatytus apdorojimo procesus šis kadras yra atvaizduojamas vartotojui.

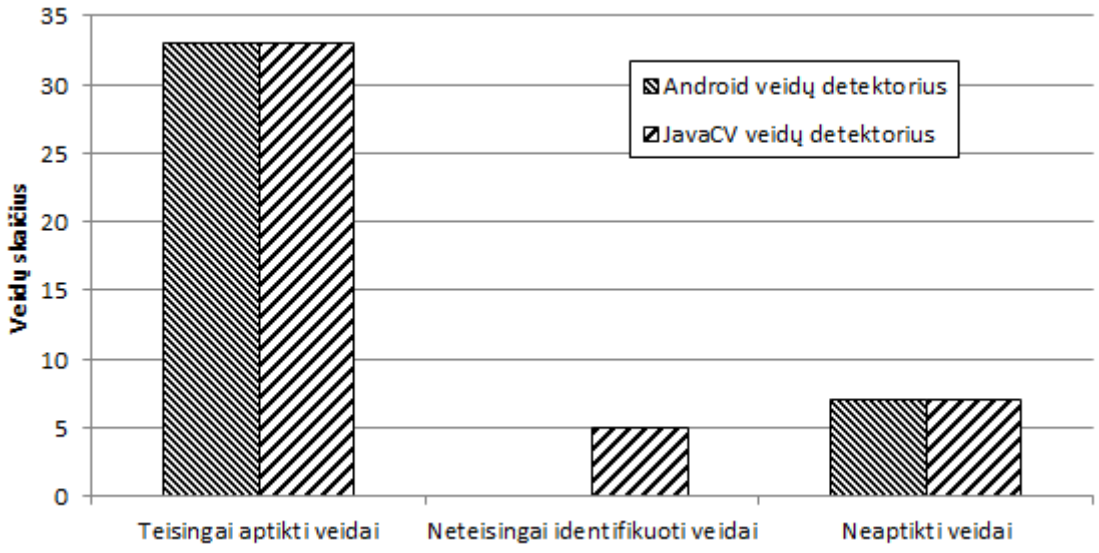
5. Užfiksuojamas ir išsaugojamas vaizdas. Programiniame kode yra aprašoma kaip bus užfiksuojama nuotrauka ir kur ji bus išsaugoma.

6. Atlaisvinama kamera ir grąžinamas rezultatas. Baigus darbą, programa privalo atlaisvinti kamerą, jog ja galėtu naudotis kitos programos.

Kadangi dažniausiai nuotraukos būna labai didelės raiškos (3 Mpx ir daugiau), programa sėkmingai gavusi rezultatą jį suglaudina, jog apdorojimas vyktų sparčiau. Suglaudintas rezultatas yra atvaizduojamas pagrindiniame programos lange, kad su juo būtų galima atlikti tolimesnius veiksmus. Suglaudinimo santykis yra apibrėžtas kaip konstanta programos kode *RATIO* kintamajame. Detaliau šis kintamasis bus aptartas sekančiame poskyryje.

5.3. Veidų aptikimas ir išskyrimas

Android operacinėje sistemoje, nuo pat pirmos jos versijos yra įgyvendintas veidų aptikimas nuotraukose. Jei veidus norima aptikti nuotraukoje, daug sunkumų nekyla –



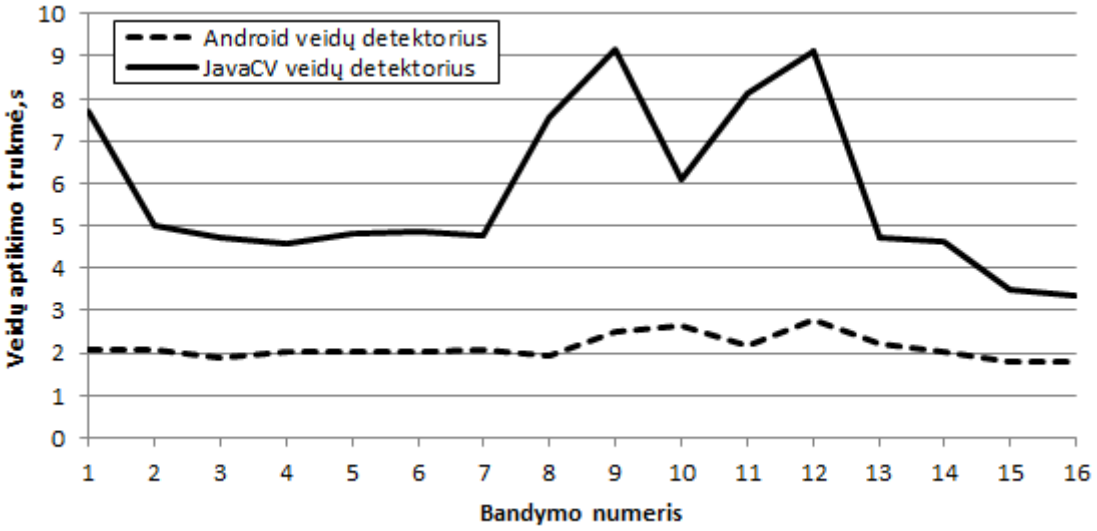
5.5 pav. Veidų detektoriaus tikslumo palyginimas *Android* ir *JavaCV* bibliotekose [3]

sudėtingiau yra juos aptikti vaizdo sraute kuris yra gaunamas iš kameros. Tokiu atveju kiekvieną vaizdo kadrą reikia analizuoti atskirai, ko pasėkoje yra susiduriama su operatyviosios atminties stygiumi.

Žinoma, veidus nuotraukose aptikti galima ir kitais metodais. Francesco Florio atlikto tyrimo metu buvo palyginti *Android* ir *JavaCV* bibliotekos veidų detektoriai [3]. Kaip matyti iš 5.5 paveikslo, ištestavus abu veidų detektorius su 40 skirtingų paveikslų – skirtumai tarp abiejų metodų praktiškai nepastebimi. Abu metodai sugebėjo teisingai aptikti 33 veidus iš 40 ir abu metodai neaptiko 7 veidų. Vienintelis neigiamas *JavaCV* metodo aspektas – jog jis kartais aptinka veidus ten kur jų išties nėra. Kaip matyti iš šio grafiko – jis net 5 bandymuose klaidingai aptiko veidus.

Nors abu šie metodai sugeba pakankamai tiksliai aptikti veidus, pagrindinis skirtumas yra pastebimas lyginant laiko tarpą reikalingą veidams aptikti šiuose metoduose. 5.6 paveiksle galima matyti, jog visais atvejais *Android* veidų detektorius veikia žymiai sparčiau. Šiam detektoriui apytiksliai užtenka 2 s aptikti visiems veidams, kai tuo tarpu *JavaCV* metodui gali prireikti nuo 3 s iki 9 s. Kadangi laikas reikalingas veidams aptikti yra labai svarbus parametras – *JavaCV* metodą galima pripažinti netinkančiu.

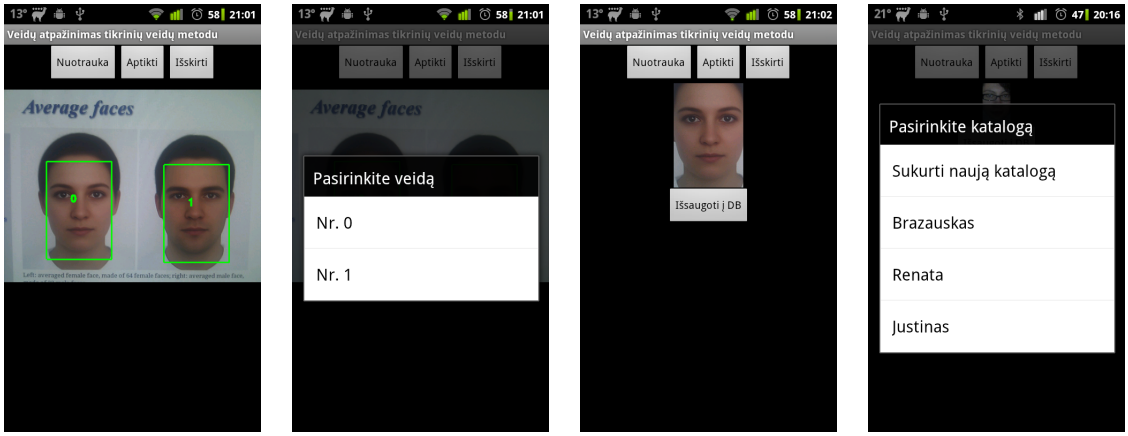
Vaizdo kadre ieškant veidų *Android* veidų detektoriumi, visų pirma yra sukuriamas *FaceDetector* objektas, kuriam kaip parametrai yra nurodomi analizuojamo vaizdo aukštis, plotis ir maksimalus veidų skaičius, kurį gali aptikti detektorius. Šiame objekte iškvietus metodą *findFaces()* ir jam nurodžius analizuojamą vaizdą, yra grąžinami tokie parametrai, kaip aptiktų veidų skaičius bei juos apibūdinantys bruožai. Visi šie parametrai yra saugomi masyve, kur kiekvienas masyvo narys apibūdina skirtingą veidą, tad *for()* cikle galima nesunkiai išanalizuoti kiekvieną veidą ir atnaujinti vaizdą sužymint jame veidus.



5.6 pav. Veidų aptikimo trukmės palyginimas *Android* ir *JavaCV* bibliotekose [3]

Programos kodas 5.4 Veido posvyrio erdvėje klaidingas deklaraivmas

```
1 const float X2F = 1.0f / 65536.0f;
2 _env->SetFloatField(face, gFaceOffsets.confidence, faceData.
    confidence);
3 _env->SetFloatField(face, gFaceOffsets.midpointx, faceData.
    midpointx);
4 _env->SetFloatField(face, gFaceOffsets.midpointy, faceData.
    midpointy);
5 _env->SetFloatField(face, gFaceOffsets.eyedist, faceData.eyedist);
6 _env->SetFloatField(face, gFaceOffsets.eulerx, 0);
7 _env->SetFloatField(face, gFaceOffsets.eulery, 0);
8 _env->SetFloatField(face, gFaceOffsets.eulerz, 0);
```



a) *b)* *c)* *d)*

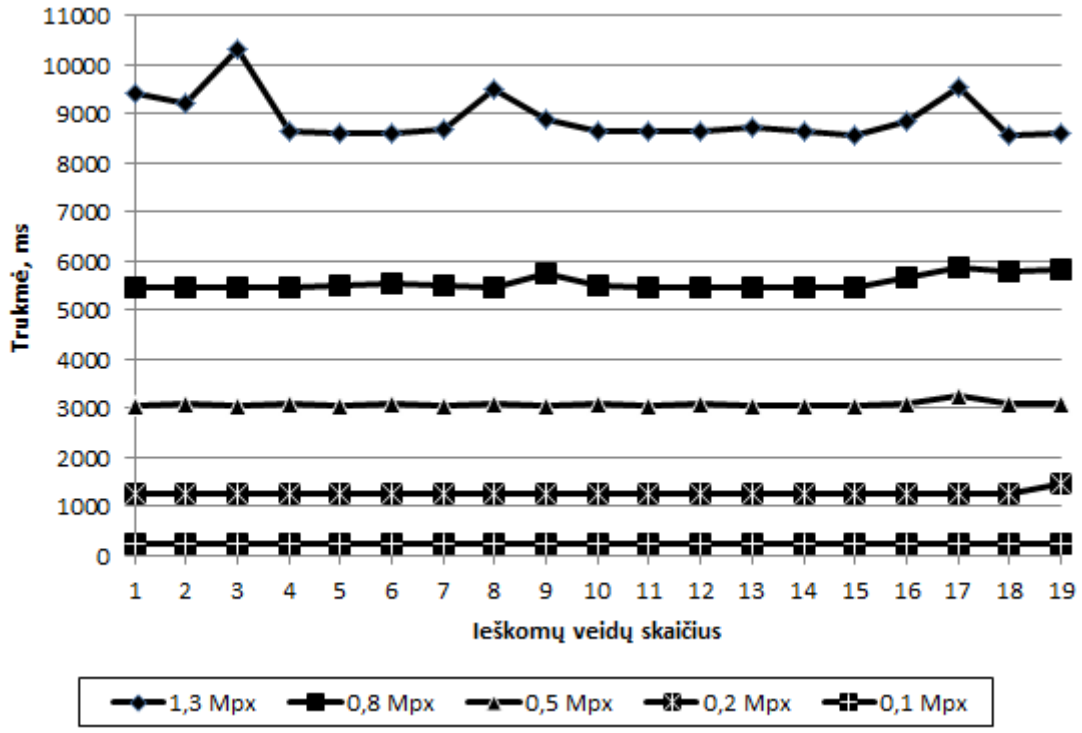
5.7 pav. Gautos nuotraukos apdorojimas: a) veidų aptikimas; b) pasirinkimas kurį veidą išskirti, c) išskirtas veidas ir d) veido išsaugojimas duomenų bazėje

Veidų detektorius kiekvieną rastą veidą apibūdina šiais parametrais: veido pasvirimas (x, y ir z ašyse), atstumas tarp akių, tarpuakio koordinates ir veido patikimumą. Veido patikimumas kinta ribose tarp 0 ir 1 bei apibūdina tikimybę, jog aptiktas objektas iš tiesų yra veidas. Dažniausiai užtenka, jog patikimumas būtų didesnis nei 30 %. Gaila, tačiau kaip matyti iš 5.4 kodo ištraukos, *Android* operacinėje sistemoje veido posvyrio kampas x, y ir z ašyse yra niekuomet negrąžinamas. Tai yra klaida palikta įgyvendinant šią operacinę sistemą ir iki šiol neištaisyta.

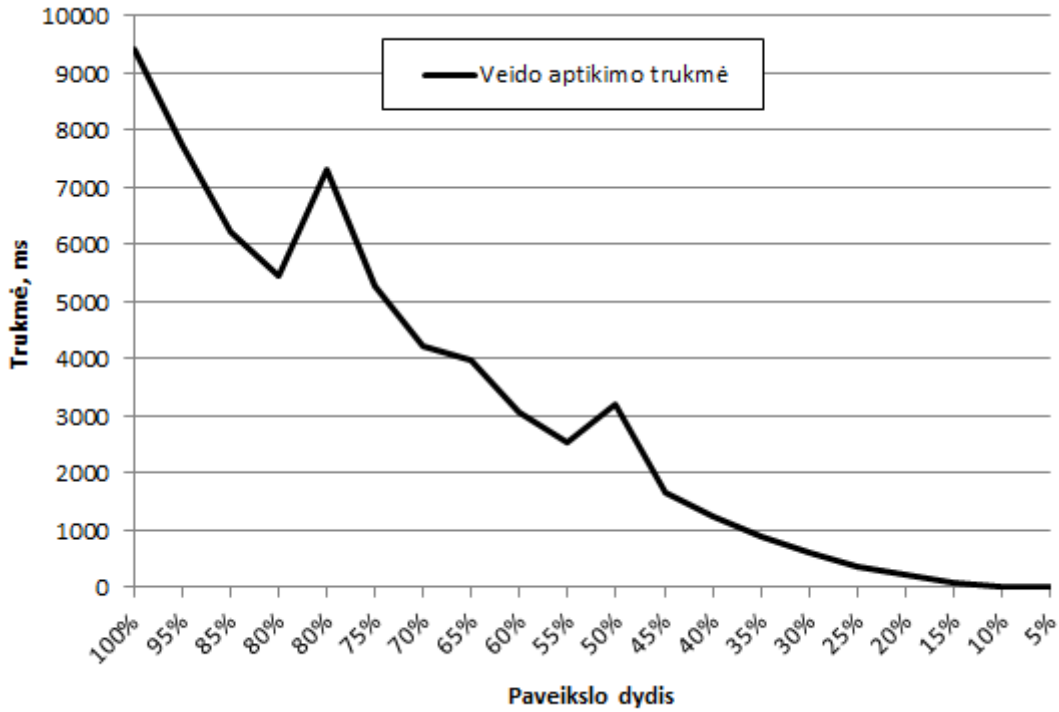
Kaip matyti iš 5.7 paveikslo, praeitame etape gavus nuotrauką galima pradėti ją apdoroti. 5.7 *a)* paveiksle yra pateikti visi aptikti veidai, kurie yra pažymėti stačiakampiu. Šiuos veidus galima išskirti nuspaudus mygtuką *Išskirti* – visų pirma vartotojas yra klausiamas kurį veidą reikia išskirti (žr. 5.7 *b* pav.), o vėliau išskirtas veidas yra atvaizduojamas pagrindiniame programos lange (žr. 5.7 *c* pav.). Galiausiai, išskirtas veidas gali būti išsaugotas duomenų bazėje (žr. 5.7 *d* pav.).

Įgyvendinus veidų aptikimą nuotraukose gali iškilti klausimas kaip maksimalus ieškomų veidų kiekis, kuris yra nurodomas detektoriui, įtakoja veidų detektoriaus spartą. Šis parametras buvo ištirtas 5 skirtingų dydžių paveiksluose keičiant maksimalų ieškomų veidų skaičių nuo 1 iki 19, o gauti rezultatai yra pateikti 5.8 grafike. Tyrimo metu buvo naudojamas vienas paveikslas su 50 veidų keičiant jo dydį Mpx. Iš šio grafiko galima padaryti išvadą, jog ieškomų veidų skaičius detektoriaus spartai neturi. Kai paveikslų dydis artėja prie 1 Mpx ribos, nepriklausomai nuo ieškomų veidų skaičiaus proceso laikas pradeda drastiškai augti, o keičiant maksimalų ieškomų veidų skaičių atsiranda nežymūs trukmės svyravimai.

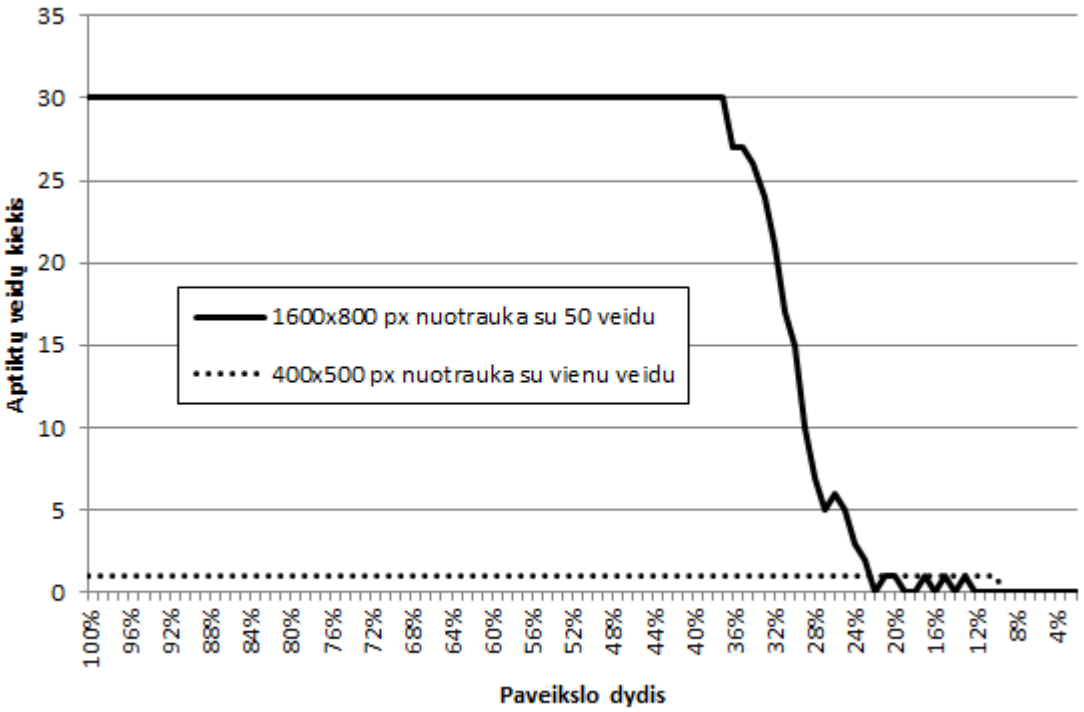
Iš 5.9 grafiko matyti, jog detektoriaus sugaištamas laikas ieškant veidų yra tiesiogiai proporcingas paveikslo dydžiui. Šiame grafike matyti kaip kinta detektoriaus sugaištamas



5.8 pav. Veidų aptikimo skirtingų matmenų paveiksluose trukmės priklausomybė ieškomų veidų skaičiaus



5.9 pav. Veidų aptikimo trukmės priklausomybė nuo paveikslo dydžio



5.10 pav. Aptinkamų veidų kiekio priklausomybė nuo paveikslo dydžio

laikas apdorojant paveikslą, keičiant jo matmenis. Šiame grafike abscisių ašyje yra pateiktas paveikslo dydis išreikštas procentais nuo jo originalaus dydžio, čia 100 % atitinka 1,3 Mpx.

Išsiaiškinus, jog ieškomų veidų kiekis neįtakoja veidų detektoriaus trukmės, kuri yra tiesiogiai proporcinga paveikslo dydžiui – iškyla klausimas koks yra optimalus paveikslo dydis, kuriame detektorius sugeba aptikti veidus. 5.10 grafike yra pateikti tyrimo rezultatai. Šio tyrimo metu 1,3 Mpx ir 0,2 Mpx dydžio paveikslų dimensijos buvo mažinamos 1 % žingsniu ir stebima kaip veidų detektorius sugeba aptikti veidus. Kaip matyti iš grafiko, kai paveiksle yra labai daug veidų, ties 27 % paveikslo dydžio riba aptiktų veidų kiekis pradeda staigiai mažėti iki 23 % ribos, ties kuria detektorius nebesugeba aptikti veidų. Tuo tarpu paveikslą su vienu veidu galima mažinti net iki 10 % ribos. Pagal šiuos rezultatus galima teigti, jog paveikslas, jei jame 50 ir daugiau veidų, privalo būti bent jau 324×648 px dydžio. Tuo tarpu jei paveiksle yra tik vienas veidas, kuris užima didžiąją vaizdo dalį, paveikslą galima mažinti net iki 40×50 px ribos. Kai paveikslas yra mažesnis už šią ribą – detektorius nebesugeba aptikti veido.

5.4. Veidų atpažinimas

Įgyvendinant tikrinių veidų algoritmą *Android* operacinėje sistemoje buvo renkamasi tarp kelių bibliotekų skirtų vaizdų apdorojimui: *OpenCV4Android*, *JavaCV* ir *OpenCV*.

Programos kodas 5.5 *Application.mk* bylos turinys

```
1  # Kad butu galima naudoti STD bibliotekas
2  APP_STL := stlport_static
3
4  # OpenCV bibliotekos
5  APP_STL := gnustl_static
6  APP_CPPFLAGS := -frtti -fexceptions
7
8  # Nurodomas procesoriaus tipas
9  # armeabi      -- AVD
10 # armeabi-v7a -- HTC Desire
11
12 # APP_ABI := armeabi
13 APP_ABI := armeabi-v7a
```

OpenCV4Android ir *JavaCV* yra *OpenCV* bibliotekos atmainos. *JavaCV* yra ta pati biblioteka pritaikyta *JAVA* programavimo kalbai, kai tuo tarpu *OpenCV4Android* – *OpenCV* biblioteka dalinai perkelta į *Android API*. *OpenCV* biblioteka (pilnas jos pavadinimas angl. *Open Source Computer Vision library*, liet. *atvirojo kodo kompiuterio regos biblioteka*) buvo sukurta dar 1999 metais Intel korporacijos, o nuo 2008 metų yra plėtojama „Willow Garage“. Šią biblioteką sudaro daugiau nei 2500 optimizuotų algoritmų, o jos funkcijos yra pilnai pritaikytos C, C++ bei Python programavimo kalboms.

Detaliau išanalizavus šias tris bibliotekas buvo prieita išvados jog tikrinių veidų algoritmas turi būti įgyvendintas *OpenCV* bibliotekoje. Taip buvo nuspręsta atsižvelgus į tai, jog C++ sąsaja yra spartesnė už *JAVA* sąsaja (šiuo atveju *JavaCV*), o *OpenCV4Android* biblioteka dar nėra užbaigta ir turi daugybę klaidų.

Kadangi *OpenCV* biblioteka naudoja C++ kalbos sąsają – ją *Android* operacinėje sistemoje galima naudoti tik per *Android NDK*. *Android NDK* yra įrankių rinkinys kuris leidžia *Android* operacinėje sistemoje naudoti prigimtinį programos kodą C arba C++ programavimo kalbomis bei veikia per JNI (pilnas pavadinimas angl. *JAVA Native Interface*).

Naudojant *Android NDK*, yra būtina egzistuojančiame *Android* projekte sukurti naują direktoriją pavadinimu *jni*, kurioje bus saugomas programos kodas parašytas prigimtine programavimo kalba. Paleidus *ndk-build* komandą automatiškai yra sugeneruojamos *Android.mk* bei *Application.mk make* bylos.

Kaip matyti iš 5.5 ir 5.6 programos kodo ištraukų, šiose bylose yra nurodoma kurią bylą reikia kompiliuoti, kokias bibliotekas įtraukti ir kokiam procesoriui bus optimizuota programa. Kadangi programa buvo testuojama HTC Desire telefone, šio telefono procesoriaus tipas (*armeabi-v7a*) ir nurodytas *Application.mk* byloje. Tuo tarpu iš *JAVA* pusės (*Android SDK*) prieš iškviečiant prigimtines funkcijas – visų pirma reikia užkrauti JNI

Programos kodas 5.6 *Android.mk* bylos turinys

```
1  LOCAL_PATH := $(call my-dir)
2
3  include $(CLEAR_VARS)
4
5  include /home/jb/OpenCV-2.3.1/share/OpenCV/OpenCV.mk
6  OPENCV_CAMERA_MODULES:=off
7
8  LOCAL_MODULE    := recognize-jni
9  LOCAL_SRC_FILES := recognize-jni.cpp
10
11 # Kad butu galima loginti is C++ aplinkos
12 LOCAL_LDLIBS  := -lGLESv1_CM -ldl -llog
13
14 include $(BUILD_SHARED_LIBRARY)
```

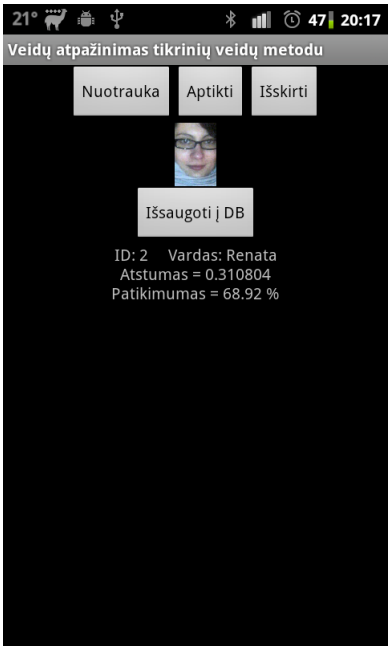
Programos kodas 5.7 Prigimtinių funkcijų deklaravimas

```
1  // Uzkrauti NATIVE biblioteka
2  static {
3      System.loadLibrary("recognize-jni");
4  }
5
6  // JNI klases
7  public native String  stringFromJNI();
8  public native JavaCls jniRecognizeRT();
```

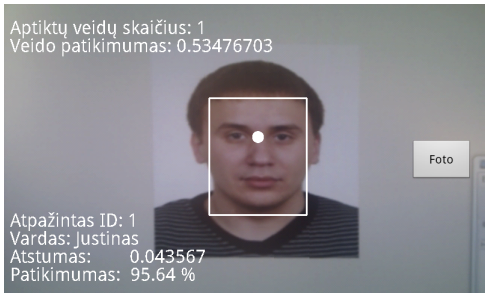
biblioteką bei deklaruojant kiekviena JNI klasę (žr. 5.7 programos kodo ištrauką).

Sukompiliavus programą ir ją paleidus išmaniajame telefone, po veidų atpažinimo operacijos yra matomas vaizdas, kuris yra pateiktas 5.11 ir 5.12 paveiksluose. 5.11 paveiksle matyti, jog atpažinus veidą iš nuotraukos vartotojui yra pateikiamas unikalus atpažinto asmens identifikacijos numeris, jo vardas, atstumas tarp testuojamo paveikslo ir identifikuoto asmens bei patikimumas jog asmuo yra atpažintas teisingai. Tuo tarpu 5.12 paveiksle, kur veidas yra atpažįstamas realiu laiku, yra pateikiama identiška informacija papildyta aptiktų veidų skaičiumi bei patikimumu jog aptiktas objektas iš tiesų yra veidas (kairiajame viršutiniame nuotraukos kampe). *Android* operacinėje sistemoje tikrinių veidų algoritmas buvo įgyvendintas remiantis Robin Hewitt publikuotu straipsniu [39].

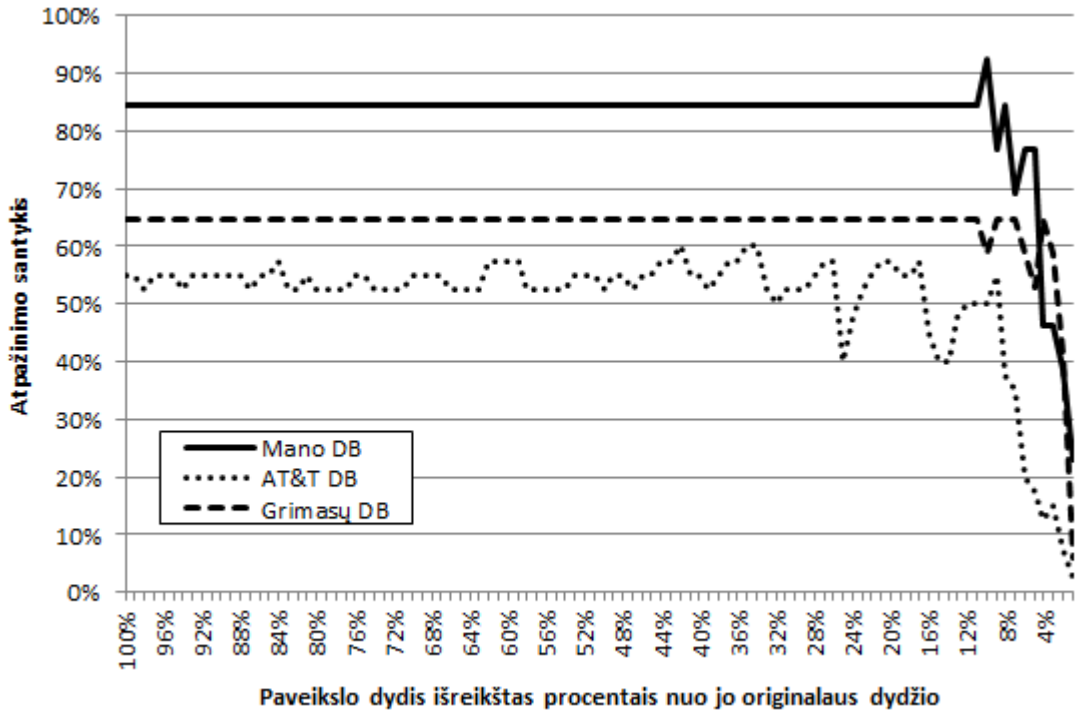
Android operacinėje sistemoje atpažįstant veidus iškyla klausimas kokio dydžio turėtų būti aptiktas, iškirptas ir atpažinimui paruoštas veidas, kada jis per didelis ir kada jis per mažas. 5.13 grafike yra pateiktos tikrinių veidų algoritmo atpažinimo santykio priklausomybe kai apmokymui ir testavimui yra naudojami skirtingų dydžių paveikslai. Testavimas buvo atliktas trijose veidų duomenų bazėse keičiant paveikslų dydį vieno pro-



5.11 pav. Veido atpažinimas iš nuotraukos



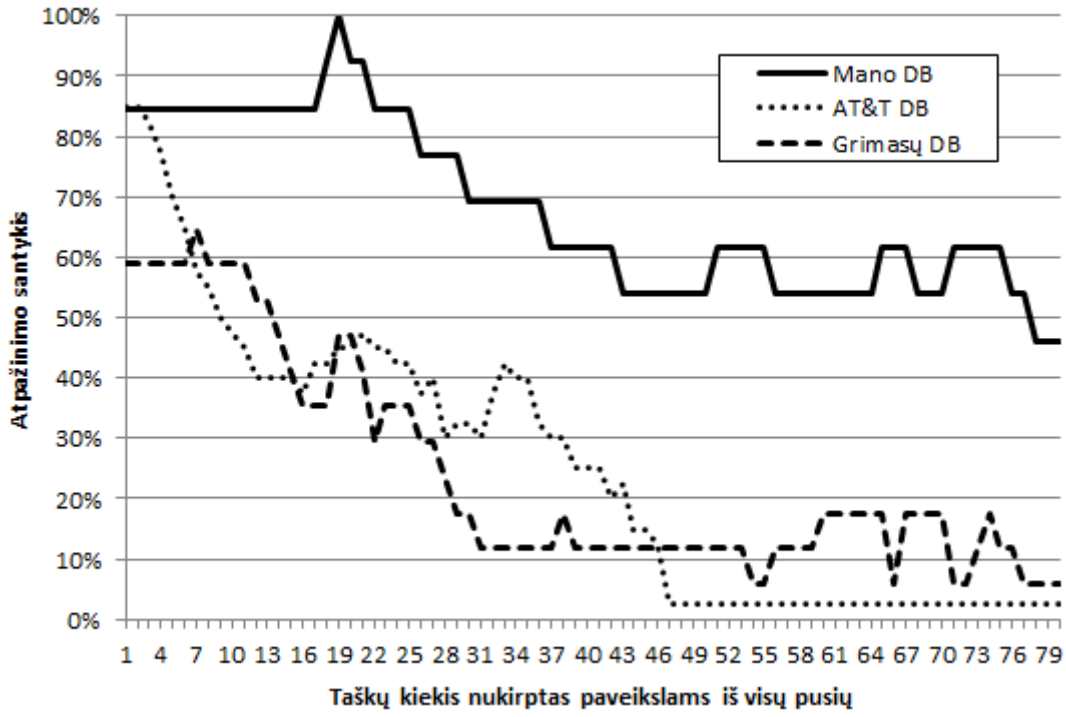
5.12 pav. Veido atpažinimas realiu laiku



5.13 pav. Veido atpažinimo tikslumo priklausomybė nuo paveikslo dydžio

cento žingsniu. Kaip matyti iš grafiko, grimasų ir mano duomenų bazėse kritinis taškas yra ties 10 % riba, kas atitinka 20×18 px paveikslo dydį. Tuo tarpu AT&T duomenų bazėje kritinis taškas pasireiškia kur kas anksčiau – ties 26 % riba. Tai paaiškinti galima tuo, jog šioje duomenų bazėje pradinis paveikslo dydis yra mažesnis, o 26 % riba atitinka 30×24 px. Pasiekus šias ribas yra pastebimas spartus atpažinimo santykio nuosmukis visose veidų duomenų bazėse.

Išsiaiškinus jog veidų atpažinimui yra būtina naudoti didesnius nei 30×30 px paveikslus iškyla klausimas kaip atpažinimui naudojamame paveiksle turi būti pozicionuojamas veidas ir kaip tai įtakoja atpažinimo santykį. Gal būt algoritmas veikia tiksliau kai veidas paveiksle yra centruotas, užpildo visą vaizdą ar palieka tuščius kraštus? Siekiant atsakyti į šį klausimą algoritmas dar kartą buvo ištestuotas trijose veidų duomenų bazėse šį sykį nukerpant paveikslams kraštus. Nukerpamų kraštų plotis šio tyrimo metu buvo didinamas nuo 1 px iki 80 px. Kaip matyti iš 5.14 paveikslo – grimasų bei AT&T duomenų bazėje didinant nukerpamų kraštų plotį atpažinimo santykis pastoviai mažėja. Taip nutiko dėl to, jog šiuose duomenų bazėse, priešingai nei mano duomenų bazėje, veidai yra necentruoti ir skirtingo dydžio. To pasekoje pasitaiko jog yra nukerpama veido dalis. Tuo tarpu mano duomenų bazėje, kai nukerpamos juostos plotis yra tarp 16 ir 22 px – yra pasiekiamas atpažinimo pikas. Šiame pike atpažinimo santykis pasiekia 100 %. Iš to galima padaryti išvadą, jog jeigu veidas yra centruotas – geriausiai atpažinimo algoritmui perduoti veidą



5.14 pav. Veido nuotraukos karpymas iš visų pusių

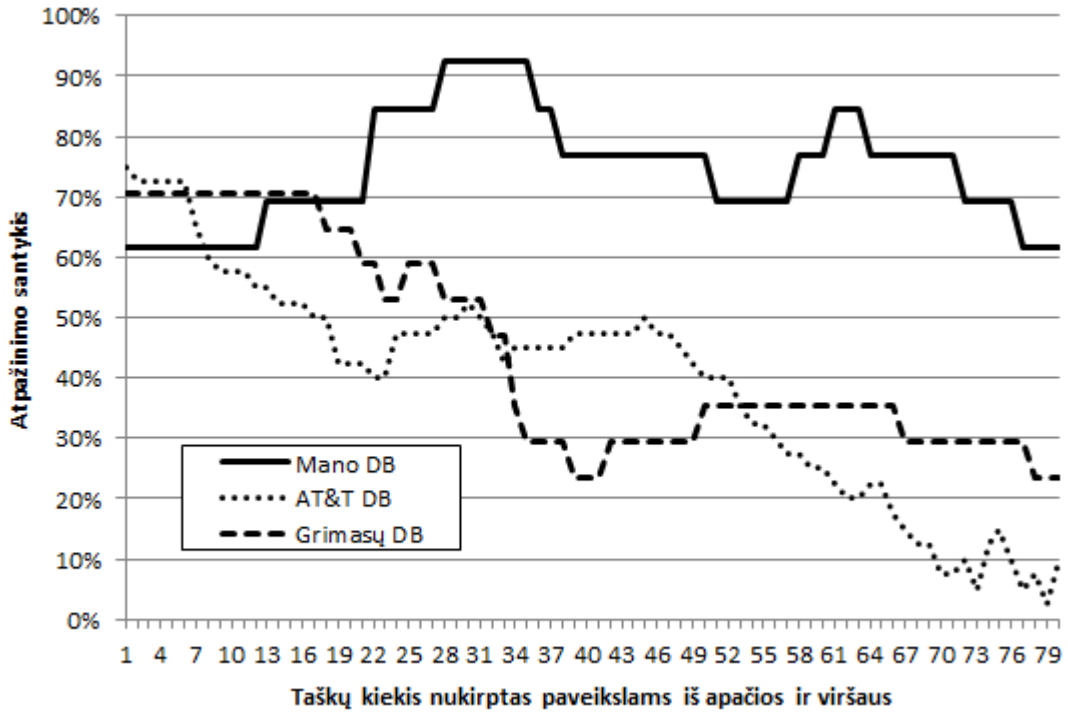
be kraštų su fonu, bei su nukirptu viršugalviu bei kaklu.

5.15 grafike pavaizduota kaip kinta atpažinimo tikslumas kai paveikslams yra nukerpamas tik viršus ir apačia. Kadangi, kaip minėta prieš tai, AT&T ir grimasų duomenų bazėse veidai yra necentruoti – juose didinant nukerpamų juostų plotį atpažinimo santykis mažėja. Tuo tarpu mano veidų duomenų bazėje kuomet nukerpamos juostos plotis yra tarp 30–35 px – vėlgi yra pasiekiamas atpažinimo pikas lygus 92% tikslumui. Iš šių rezultatų galima padaryti išvadą, jog svarbiausia yra kad veidas būtų centruotas ir iš nuotraukos būtų pašalintas perteklinis vaizdas esantis virš ir po veidu.

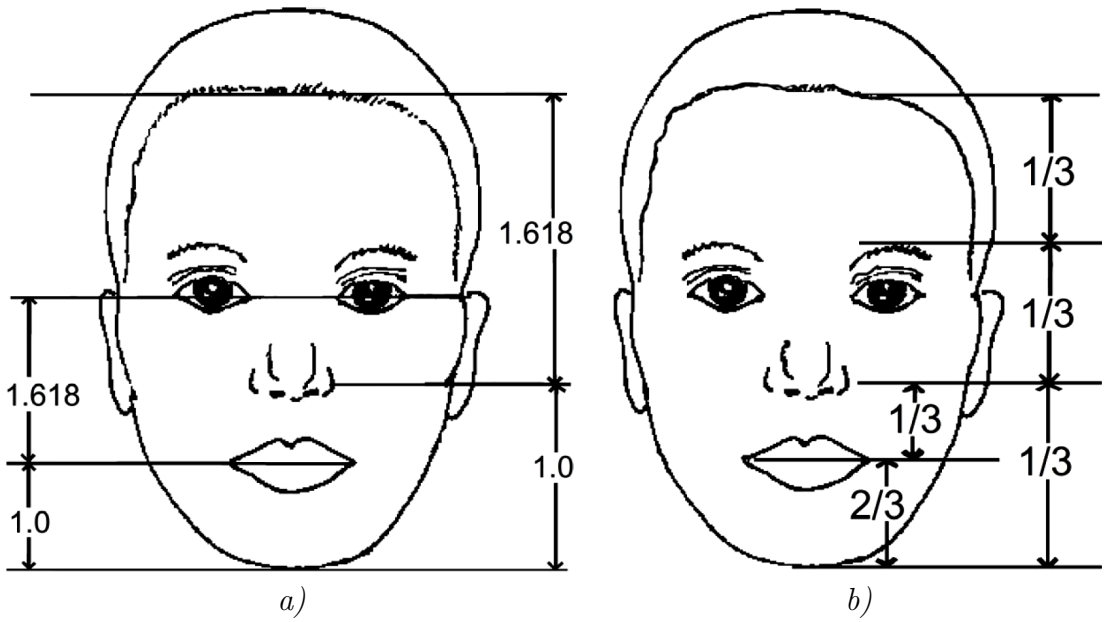
Literatūros šaltiniuose [40, 41, 42, 43] yra rekomenduojama veidą iškirpti vadovaujantis trečdalių arba auksinio santykio (angl. *golden ratio*) taisykle. Pagal auksinio santykio taisyklę iškirpto veido matmenų santykis turėtų būti:

$$\frac{aukstis}{plotis} = \frac{1+\sqrt{5}}{2}. \tag{5.1}$$

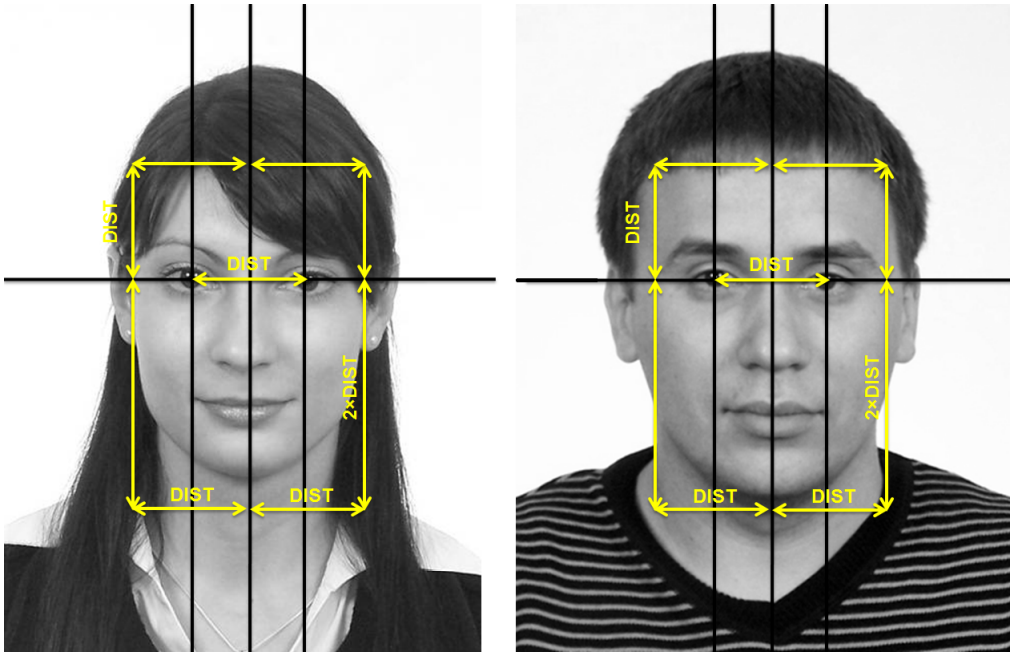
Pasak veido auksinės proporcijos taisyklės – kiekvienas veidas gali būti suskirstytas pagal auksinę proporciją brėžiant horizontalias linijas ties plaukais, nosimi ir smakru, arba ties akimis, lupomis ir smakru. Toks veido suskirstymas yra pateiktas 5.16 *a)* paveiksle. Tuo tarpu trečdalių metodas (žr. 5.16 *b)* pav.) teigia, jog kiekvienas veidas gali būti suskirstytas į lygius trečdalius brėžiant linijas ties plaukų linija, antakiais, nosimi bei smakru. Tuo tarpu atstumas tarp smakro ir nosies gali būti suskaidytas į dvi dalis – $^1/_3$



5.15 pav. Veido nuotraukos karpymas iš visų pusių



5.16 pav. Vertikalaus veido: *a)* auksinės proporcijos [4, 5] ir *b)* trečdalių proporcijos [6, 7]



5.17 pav. Iškirpamo veido ribos

tarp nosies ir lūpų, bei ⅔ tarp lūpų ir smakro.

Išanalizavus turimas veidų nuotraukas, pagal šias proporcijas buvo sudarytas veido iškirpimo modelis pateiktas 5.17 paveiksle. Šiame modelyje veido kraštinės yra apskaičiuojamos iš dviejų parametrų (tarpuakio taško ir atstumo tarp akių) pagal 5.2–5.5 formules:

$$kaire(X) = tarpuakis(X) - atstumasTarpAkiu \tag{5.2}$$

$$virsus(Y) = tarpuakis(Y) - atstumasTarpAkiu \times 2 \tag{5.3}$$

$$desine(X) = tarpuakis(X) + atstumasTarpAkiu \tag{5.4}$$

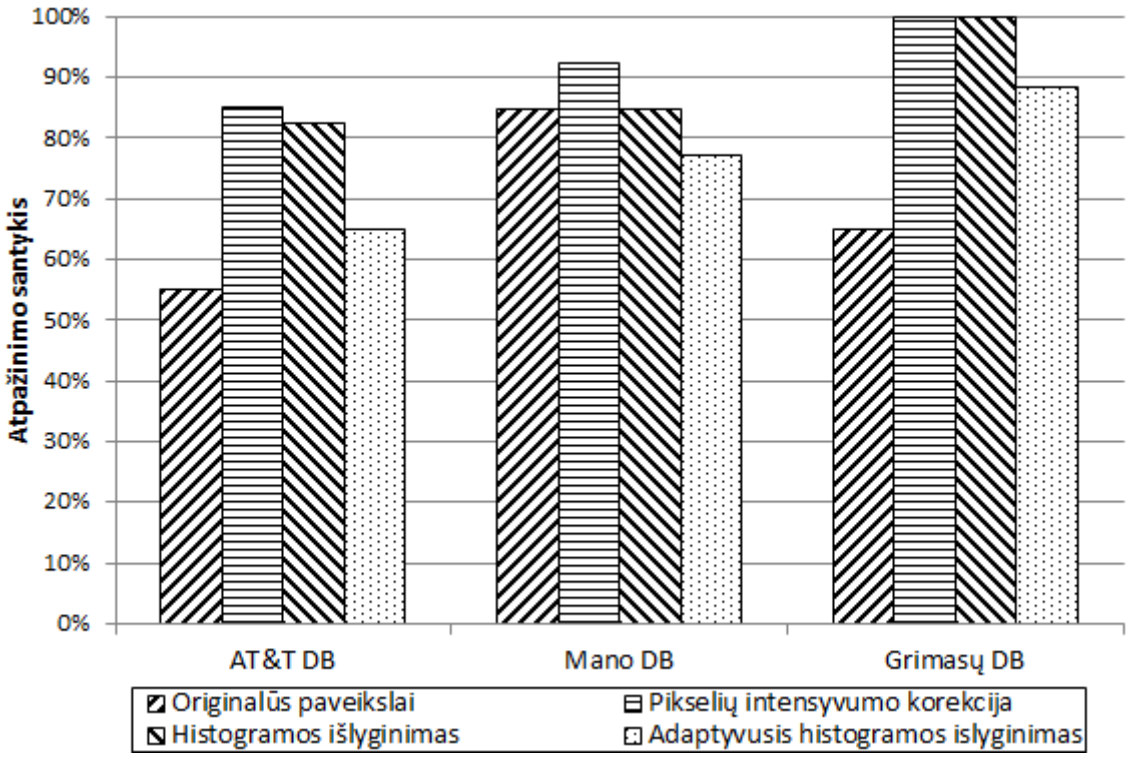
$$apacia(Y) = tarpuakis(Y) + atstumasTarpAkiu \times 2 \tag{5.5}$$

Šiose formulėse iš tiesų yra apskaičiuojamos dviejų taškų koordinatės (X_1, Y_1) ir (X_2, Y_2), pagal kurias yra brėžiamos keturios linijos (žr. 5.8 kodo ištrauką). Atitinkamai pagal šį veido modelį veido aukštis yra lygus $3\times atstumasTarpAkiu$, o plotis – $2\times atstumasTarpAkiu$.

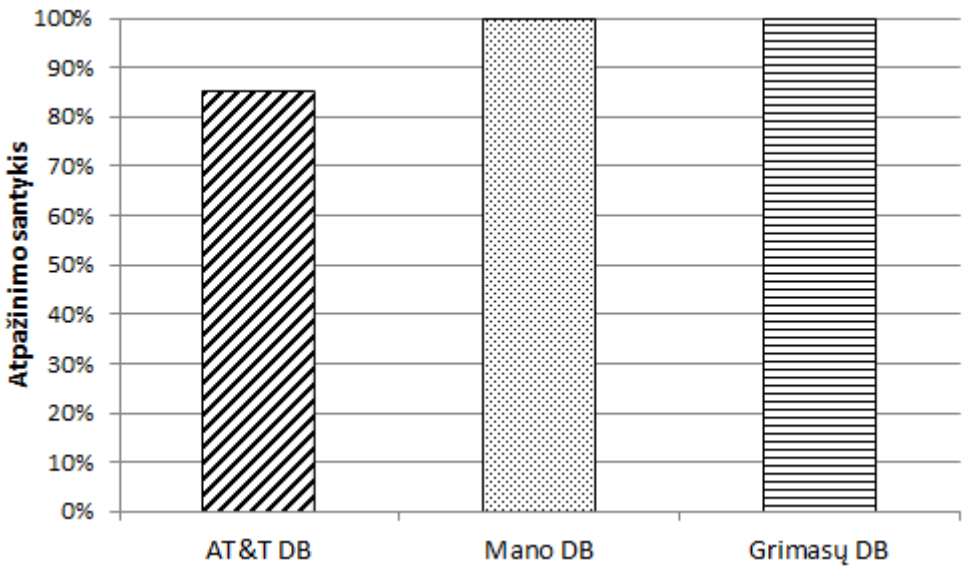
Šio tyrimo metu buvo pastebėta, jog iškirptam veidui pritaikius intensyvumo korekciją yra gaunami kur kas geresni atpažinimo rezultatai. Kaip matyti iš 5.18 paveikslo AT&T duomenų bazėje atpažinimo santykis pagerėjo 30 %, mano duomenų bazėje – 7,7 %, o grimasų duomenų bazėje 35,3 %. Kartu su intensyvumo korekcija buvo ištirtas ir histogramos išlyginimo poveikis. Paveikslui pritaikius tiek paprastą histogramos išlyginimą, tiek ir adaptyvųjį histogramos išlyginimą, taip pat yra pastebimas atpažinimo santykio padidėjimas. Kaip matyti iš grafiko, adaptyvusis histogramos suvienodinimas nėra toks efektyvus – mano duomenų bazėje po šio metodo pritaikymo atpažinimo santykis nusmu-

Programos kodas 5.8 Aptikto veido iškirpimas

```
// Kaire linija
canvas.drawLine((eyeMidPoint[i].x - eyeDist[i]),
                (eyeMidPoint[i].y + eyeDist[i]),
                (eyeMidPoint[i].x - eyeDist[i]),
                (eyeMidPoint[i].y - eyeDist[i] * 2),
                newPaintForFaces);
// Desine linija
canvas.drawLine((eyeMidPoint[i].x + eyeDist[i]),
                (eyeMidPoint[i].y + eyeDist[i]),
                (eyeMidPoint[i].x + eyeDist[i]),
                (eyeMidPoint[i].y - eyeDist[i] * 2),
                newPaintForFaces);
// Virsutine linija
canvas.drawLine((eyeMidPoint[i].x - eyeDist[i]),
                (eyeMidPoint[i].y + eyeDist[i]),
                (eyeMidPoint[i].x + eyeDist[i]),
                (eyeMidPoint[i].y + eyeDist[i]),
                newPaintForFaces);
// Apatine linija
canvas.drawLine((eyeMidPoint[i].x - eyeDist[i]),
                (eyeMidPoint[i].y - eyeDist[i] * 2),
                (eyeMidPoint[i].x + eyeDist[i]),
                (eyeMidPoint[i].y - eyeDist[i] * 2),
                newPaintForFaces);
```



5.18 pav. Nuotraukų esančių duomenų bazėse apdorojimas skirtingais metodais



5.19 pav. Tikrinių veidų algoritmo įgyvendinto *Android* operacinėje sistemoje atpažinimo santykiai

ko 7,7 %. Tuo tarpu paprastas histogramos išlyginimas nedaug nusileidžia intensyvumo korekcijai, tačiau jos poveikis atpažinimo santykiui turi didžiausią teigiamą įtaką.

Pagal gautus rezultatus buvo nuspręsta, jog atpažįstant veidus realiu laiku paveikslo dydis turi būti 1024 × 768 px dydžio, o peržiūros lango išmatavimai – 320 × 240 px. Video kameros apšvietimo, baltos spalvos balanso, fokusavimo ir scenos parametrai buvo nustatyti *auto* padėtį, kas reiškia jog jie automatiškai prisiderina prie kintančio vaizdo. Atpažįstant veidus realiu laiku buvo pasirinkta vaizdo kadre ieškoti tik vieno veido, o foto–nuotraukoje – 6 veidų. Kiekvienas iškirpto veido dydis yra nustatomas į 90 × 60 px raišką ir išsaugomas. Tokia paveikslo raiška buvo pasirinkta pagal prieš tai atliktus tyrimus, siekiant išlaikyti veido proporcijas ir taikomosios programos našumą bei neprarasti tikslumo per daug priartėjus prie kritinės ribos. Pagal šiuos parametrus vieno veido paruošimas atpažinimui trunka 5–10 ms, o vieno veido atpažinimas užtrunka nuo 50 ms iki 70 ms. Vidutinė viso proceso trukmė 60 ms, kas leidžia realiu laiku apdoroti beveik 17 kadrų per sekundę. Pagrindinis kriterijus kuris įtakoja vieno veido atpažinimo trukmę yra tas, jog *Android* operacinėje sistemoje vaizdas gaunamas iš video kameros yra koduotas YUV420SP spalvų palete, tad norint dirbti su šiuo vaizdu iš pradžių tenka jį dekoduoti į RGB spalvų paletę. Tiek ir apmokant algoritmą, tiek ir atpažįstant veidus visi paveikslai yra konvertuojami į pilkų atspalvių paletę (paveiksluose panaikinamos spalvos).

Pagal visus šiuos parametrus įgyvendinus tikrinių veidų algoritmą *Android* operacinėje sistemoje buvo gauti rezultatai, kurie yra pateikti 5.19 grafike. Iš šio grafiko matyti, jog algoritmas sugeba 100 % tikslumu atpažinti veidus Mano bei Grimasų duomenų bazėse, o AT&T duomenų bazėje buvo pasiektas 85 % atpažinimo santykis.

5.5. Skyriaus apibendrinimas

Šiame skyriuje tikrinių veidų algoritmas buvo įgyvendintas *Android* operacinėje sistemoje. Vartotojo sąsaja ir paveikslų paruošimas buvo įgyvendintas *Android SDK JAVA* programavimo kalba, o pats veidų atpažinimas per *Android NDK* sąsają – kitais žodžiais tariant žemesniame lygmenyje C++ programavimo kalba. Šios programos struktūrinė schema yra pateikta B priede, o pirminis programos tekstas pateiktas E priede. Įgyvendinant veidų atpažinimo programą buvo nuspręsta veidų atpažinimą įgyvendinti tiek iš statinio vaizdo (foto–nuotraukos), tiek ir vaizdo sraute. Veido atpažinimui iš statinio vaizdo nuotrauką galima pasirinkti trimis metodais – iškviečiant gamyklinę vaizdo kamerą, iškviečiant galeriją kurioje yra saugomi visi vaizdai arba iškviečiant specialiai sukurtą vaizdo kameros programą, kuri realiu laiku seka aptiktus veidus ir grąžina foto–nuotrauką. Tuo tarpu veidų atpažinimas vaizdo sraute buvo įgyvendintas kaip nauja vaizdo kameros aplikacija, kuri kiekviename kadre ieško veido, o jį aptikusi išskiria ir perduoda veidų atpažinimo algoritmui. Įgyvendinus veidų atpažinimą realiu laiku su tyrimų metu gautais optimaliais parametrais algoritmas sugeba apdoroti net 17 kadrų per sekundę.

Veidų aptikimui vaizde buvo pasirinktas *Android* veidų detektorius kuris yra daugiau nei 200 % spartesnis už *JavaCV* veidų detektorių, o veidų atpažinimui yra naudojama „Atviroji kompiuterinės regos biblioteka (*OpenCV*)“, kuri mano nuomone yra tinkamiausia šiai užduočiai atlikti. Kadangi *Android* operacinės sistemos dokumentacijoje nepateiktas detalaus veidų detektoriaus aprašymas, pagal šio detektoriaus veikimo tikslumą bei spartą galima daryti prielaidą jog jis naudoja Viola–Jones algoritmą.

Kadangi veidų detektorius vienu metu gali ieškoti daugybės veidų – atlikus tyrimą buvo išsiaiškinta jog ieškomų veidų kiekis veidų aptikimo spartai įtakos neturi. Nesvarbu ar vaizde yra ieškoma 1 veido ar 20 veidų – jų aptikimo trukmė nesiskiria. Šį parametrą labiausiai įtakoja vaizdo, kuriame ieškoma veidų, dydis – vaizdą sumažinus 5 kartus (nuo 1600×800 px iki 320×160 px) veidų aptikimo trukmė sumažėja 50 kartų (nuo 10000 ms iki 200 ms). Taip pat, šiame darbe buvo išsiaiškinta, jog kuo paveiksle yra daugiau veidų – tuo jis turi būti didesnis norint juos visus sėkmingai aptikti. Pavyzdžiui, kai paveiksle yra 50 ir daugiau veidų, veidų detektorius sugeba sėkmingai juos aptikti tik tuomet, kai paveikslas yra didesnis nei 320×160 px, o tuo tarpu jei paveiksle yra tik vienas veidas užimantis beveik visą plotą – paveikslo dimensijas galima mažinti netgi iki 40×50 px.

Kalbant apie veidų atpažinimą, tai paveikslų dydžio įtaką atpažinimo tikslumui jaučiama tik tuomet, kai paveikslo matmenys peržengia 20 % jo originalaus dydžio (AT&T, Mano ir Grimasų duomenų bazėse). Kitais žodžiais tariant – norint sėkmingai atpažinti veidus paveikslai privalo būti didesni nei 30×30 px. Už paveikslų, kuriuose yra veidas, dydį – kur kas svarbesnis parametras yra veido centravimas. Atlikus tyrimą kaip šis parametras įtakoja veidų atpažinimo tikslumą buvo pastebėta, jog nukerpant nereikalingus

paveikslų karštus atpažinimo santykis išauga net iki 100 %, o kai kuriose duomenų bazėse yra pastebimas net 30 % teigiamas šuolis. Pagal šiuos rezultatus buvo nuspręsta iš nuotraukos iškirpti veidą, kurio aukščio ir pločio santykis yra $^2/_3$, o rėmelio koordinatės yra apskaičiuojamos pagal akių koordinačių taškus bei atstumą tarp jų.

Kiekvienam iškirptam veidui pritaikius histogramos išlyginimą ar pikselių intensyvumo korekciją taip pat buvo pastebėtas žymus atpažinimo santykio pagerėjimas (Grimasų duomenų bazėje atpažinimo santykis išaugo 35 %). Didžiausią įtaką atpažinimo santykio pagerėjimui turi pikselių intensyvumo korekcija, po kurios buvo gauti geresni rezultatai lyginant su paprastuoju bei adaptyviuoju histogramos išlyginimu. Galiausiai, pagal visus šiuos parametrus įgyvendinus algoritmą *Android* operacinėje sistemoje buvo pasiekti puikūs veidų atpažinimo rezultatai. Tikrinių veidų algoritmas AT&T duomenų bazėje visus veidus atpažįsta 85 % tikslumu, o Mano ir Grimasų duomenų bazėse visi veidai buvo atpažinti 100 % tikslumu.

APIBENDRINIMAS. IŠVADOS

Šiose magistro tezėse buvo atlikta analitinė veidų atpažinimo algoritmų apžvalga po kurios buvo pasirinkti trys veidų atpažinimo algoritmai (tikrinių veidų, Fišerio veidų ir 2D–DCT+SOM) detalesnei analizei. Šie trys algoritmai buvo nuodugniai išanalizuoti bei įgyvendinti *MATLAB* aplinkoje. Įgyvendinus algoritmus buvo atlikta testavimo duomenų atranka, ištirti įvairūs algoritmų parametrai ir pagal juos nustatytas optimalus algoritmas. Šis optimalus algoritmas buvo įgyvendintas *Android* operacinėje sistemoje, aptartos problemos kylančios jį įgyvendinant,buvo pasiūlyti metodai kuriais vadovaujantis veidų atpažinimo santykis yra didesnis bei optimalūs parametrai su kuriais algoritmas veikia sparčiau neprarandant tikslumo.

Analitinėje literatūros apžvalgoje buvo išanalizuoti veidų aptikimo ir atpažinimo procesai, spalvų įtaka veidų atpažinimui. Pasak naujausių tyrimų – spalvotuose paveiksluose, lyginant su nespalvotais, veidus galima atpažinti tiksliau, tačiau tokiu atveju nukenčia algoritmo sparta. Labai didelę įtaką algoritmo veidų atpažinimo tikslumui turi jam teikiamų paveikslų kokybė. Siekiant optimalių rezultatų, paveikslas turi būti tinkamos raiškos, o veidas jame turi būti fokusuotas, centruotas ir simetriškas. Yra pastebėta, jog veidas nuotraukoje gali būti pasuktas į kairę ar dešinę pusę nedaugiau nei 15°, o jei posūkio kampas didesnis – atpažinimo santykis sparčiai mažėja. Visi veidų atpažinimo algoritmai gali būti suskaidyti į dvi pagrindines grupes – lokaliųjų ir globaliųjų požymių. Lokalių požymių grupei būtų galima priskirti LBP ir Gaboro metodus, o globalių požymių – tikrinių ir Fišerio veidų metodus. Kuri veidų atpažinimo algoritmų grupė yra geresnė vienareikšmiškai atsakyti negalima. Dalis mokslininkų teigia, jog lokalūs požymiai kur kas kokybiškiau aprašo veidą, tačiau šie požymiai reikalauja sudėtingesnių skaičiavimų ir dažnai nukenčia algoritmo sparta. Visuose veidų atpažinimo algoritmuose siekiant nustatyti testuojamo veido panašumą su apmokyme naudotais pavyzdžiais yra būtina juos kažkaip palyginti. Šiam palyginimui galima pritaikyti tikėtinumo santykio logaritmą, kuris aprašo veidų tarpusavio panašumą. Pagal šį santykį algoritmas gali nuspręsti ar testuojamas veidas jam yra pažįstamas ir kaip jis turi elgtis toliau.

Atlikus analitinę apžvalgą detaliai analizei buvo pasirinkti trys veidų atpažinimo algoritmai: tikrinių veidų, Fišerio veidų bei 2D–DCT+SOM algoritmas. Visi šie algoritmai buvo detaliai išstudijuoti ir įgyvendinti *MATLAB* aplinkoje. Išanalizavus šių algoritmų veikimo principus galima teigti, jog tikrinių veidų ir Fišerio veidų algoritmai yra pakankamai panašūs – pirmasis globalių požymių išskyrimui naudoja principinių komponenčių analizę (PCA), o antrasis – Fišerio tiesinį diskriminantą (FLD). Tuo tarpu 2D–DCT+SOM algoritmas savo veikimo principu yra visiškai kitoks – jis kaip požymius išskiria dvimatės diskrečiosios kosinuso transformacijos (2D–DCT) koeficientus, kurie yra teikiami save organizuojančiam dirbtinių neuronų tinklui (SOM) tolimesnei analizei. 2D–DCT+SOM

algoritmas yra puikus tuo, jog jis pasitelkdamas dirbtinių neuronų tinklą pats sugeba apsimokyti ir identifikuoti asmenis, tačiau norint gauti puikius rezultatus, šį neuronų tinklą yra būtina labai tiksliai sukonfigūruoti, kas kartais gali tapti ypač sudėtingu uždaviniu. Taip pat, save organizuojantis neuronų tinklas pasižymi labai didele apsimokymo trukme, kas tam tikrais atvejais gali būti nepriimtina. Tuo tarpu tikrinių veidų algoritmas sugeba sparčiai apsimokyti ir nereikalauja daug vietos kietajame diske, tačiau jis yra pakankamai jautrus veido posūkio kampui ir apšviestumui. Kaip tik šių problemų neturi Fišerio veidų algoritmas, tačiau jam sunkiai sekasi atpažinti veidus kurie yra padengti šešėlio, bei jis susiduria su klasės sklaidos matricos singuliarumo problema. Visi šie trys algoritmai turi tiek teigiamų, tiek ir neigiamų savybių, tad norint išrinkti geriausią algoritmą, kuris bus įgyvendintas *Android* operacinėje sistemoje, buvo atlikti papildomi tyrimai.

Siekiant objektyviai įvertinti ir ištirti visus šiuos algoritmus, buvo nuspręsta visus tyrimus atlikti trijose skirtingose veidų duomenų bazėse. Atlikus paiešką internete buvo pasirinktos dvi laisvai prieinamos veidų duomenų bazės – AT&T (dar kitaip žinoma kaip „ORL veidų duomenų bazė“) bei Grimasų duomenų bazė. Trečioji duomenų bazė buvo sudaryta mano paties. Šios trys duomenų bazės yra pakankamai skirtingos – jose skiriasi subjektų kiekis, nuotraukų raiška, sąlygos kuriomis buvo fotografuoti subjektai ir t. t. Didžiausia duomenų bazė yra AT&T – joje yra sukaupti duomenys apie 40 subjektų, Grimasų duomenų bazėje – 18 subjektų, o mano duomenų bazėje – 14 subjektų. AT&T duomenų bazę galima būtų apibūdinti kaip sudėtingiausią – joje kinta veidų išraiškos, apšvietimas, bei veidų detalės (akiniai, barzda, ūsai), tačiau visose nuotraukose fonas yra vienodas. Grimasų duomenų bazėje yra pastovus apšvietimas, tačiau kinta fonas, veido mastelis bei išraiškos. Tuo tarpu mano duomenų bazėje kinta veido apšvietimas, fonas, veido išraiškos, tačiau visi veidai yra vienodai centruoti ir vienodo mastelio.

Įgyvendinus algoritmus *MATLAB* aplinkoje bei surinkus testavimui tinkamus duomenis buvo įvertintas algoritmų tikslumas. Aukščiausias rezultatas (100 % atpažinimo tikslumas) buvo pasiektas Fišerio veidų algoritmo Grimasų duomenų bazėje, tuo tarpu tikrinių veidų algoritmas (85 %), kaip ir 2D–DCT+SOM algoritmas (95 %), aukščiausią atpažinimo santykį pasiekė mano duomenų bazėje. Apibendrinant šių trijų algoritmų rezultatus visose duomenų bazėse galima teigti, jog tikrinių veidų algoritmas veikė stabiliausiai – jo atpažinimo santykis svyravo 10 % ribose, kai tuo tarpu Fišerio veidų algoritmo atpažinimo santykis kito 75 % ribose, o 2D–DCT+SOM algoritmo – 77 % ribose. Pastarieji du algoritmai nesusitvarkė su sudėtingiausia AT&T duomenų baze, čia jų atpažinimo santykiai krito net iki 15 % ir 3 % ribos atitinkamai.

Ištyrus atpažinimo priklausomybę nuo apmokymo pavyzdžių skaičiaus vienam asmeniui buvo pastebėta, jog šis parametras labiausiai įtakoja tikrinių ir Fišerio veidų algoritmus, kai tuo tarpu 2D–DCT+SOM algoritmo atpažinimo santykis kinta nežymiai. Didinant apmokymo pavyzdžių skaičių iki maksimalaus galimo kiekio duomenų bazėje –

tolygiai didėja ir atpažinimo santykis. Esant maksimaliam pavyzdžių skaičiui yra gaunamas 20–30 % prieaugis lyginant su vienu pavyzdžių asmeniui. Atlikus šį tyrimą buvo pastebėta, jog Fišerio veidų algoritmas nesugeba atpažinti veidų, jei apmokymo duomenų bazėje kiekvienas subjektas turi tik po vieną save apibūdinantį pavyzdį. Tuo tarpu likusių dviejų algoritmų atpažinimo santykis iškart siekdavo apie 50 %.

Kadangi veidų atpažinimo algoritmuose yra svarbi ir apsimokymo bei atpažinimo trukmė – ji taip pat buvo ištirta keičiant paveikslų skaičių duomenų bazėje ir jų raišką. Iš gautų rezultatų matyti, jog apsimokymo trukmę paveikslų skaičius labiausiai įtakoja Fišerio veidų algoritme – keičiant paveikslų skaičių nuo 17 iki 323, apsimokymo trukmė išaugą nuo 140 ms iki 21966 ms. Tikrinių veidų algoritmo apsimokymo trukmę šis parametras taip pat įtakoja, tačiau ne taip ženkliai – apytiksliai 40 % mažiau nei Fišerio veidų algoritmą, tuo tarpu 2D–DCT+SOM algoritmo apsimokymo trukmė praktiškai nekinta. Tokie pat rezultatai yra gaunami tiriant paveikslų raiškos įtaką. Mažinant paveikslo raišką, Fišerio veidų algoritme apsimokymo trukmė mažėja nuo 9 s iki 2 s, kai tuo tarpu tikrinių veidų algoritme ji kinta nuo 2 s iki 1 s. Kaip ir apsimokymo, taip ir atpažinimo trukmę paveikslų dydis bei jų kiekis labiausiai įtakoja Fišerio veidų algoritmą, o tikrinių veidų algoritme atpažinimo trukmė keičiant šiuos parametrus praktiškai nekito. Duomenų bazėje esant virš 300 paveikslų Fišerio veidų algoritmo atpažinimo procesas užtrunka 120 ms, o 2D–DCT+SOM bei tikrinių veidų – 20 ms. Mažinant paveikslų raišką buvo pastebėta, jog šiai sumažėjus 30 % Fišerio veidų algoritmo atpažinimo sparta pamažu artėja prie tikrinių veidų spartos.

Kadangi tikrinių veidų algoritmas yra pagrįstas principinių komponenčių analize – buvo ištirta kaip šių komponenčių kiekis įtakoja atpažinimo tikslumą bei apsimokymo ir atpažinimo spartą. Nors daugumoje literatūros šaltinių rekomenduojamas komponenčių kiekis yra 20, tyrimo metu buvo išsiaiškinta, jog šiose veidų duomenų bazėse pilnai pakanka 15 komponenčių. Ir toliau didinant jų kiekį atpažinimo santykio pokyčiai nepastebėti. Apsimokymo ir atpažinimo trukmė nuo komponenčių kiekio nepriklauso.

Išanalizavus šiuos tris algoritmus buvo padaryta išvada, jog tinkamiausias algoritmas įgyvendinimui *Android* operacinėje sistemoje yra tikrinių veidų, nes šis algoritmas:

1. Prie skirtingų sąlygų šis algoritmas veikia stabiliausiai;
2. Jis puikiai veikia kai subjektai turi mažą kiekį save apibūdinančių nuotraukų (šis parametras yra svarbus, nes atmintis mobiliajame įrenginyje gali būti ribota);
3. Sparčiausiai apsimoko ir atpažįsta veidus;
4. Šio algoritmo apsimokymo ir atpažinimo trukmė yra atspariausia paveikslų kiekio bei raiškos pokyčiui;
5. Savo pirminio programos teksto apimtimi jis yra lengvesnis už Fišerio veidų algoritmą ir nereikalaują papildomų bibliotekų skirtų darbui su dirbtinių neuronų tinklais;

6. Vietoje rekomenduojamų 20 principinių komponenčių, pasiekti maksimalius rezultatus jam užtenka 15 komponenčių.

Įgyvendinant tikrinių veidų algoritmą *Android* operacinėje sistemoje pradinių vaizdų gavimui buvo pasirinkti trys metodai – gamyklinės foto–kameros iškvietimas, telefono nuotraukų galerijos iškvietimas ir naujos vaizdo kameros su papildomu funkcionalumu įgyvendinimas. Įgyvendinti naują vaizdo kamerą buvo nuspręsta norint ją kaip įmanoma labiau pritaikyti šiai užduočiai – ji realiu laiku apdoroja kiekvieną vaizdo kadrą jame aptikdama visus veidus. Aptikti veidai kadre yra sužymima kvadratais ir vartotojui yra pateikiama informacija apie aptiktų veidų kiekį. Atlikti veidų aptikimo funkciją buvo pasirinktas *Android* veidų detektorius, kuris yra pranašesnis už *JavaCV* detektorių. Palyginus šiuos du veidų detektorius paaiškėjo, jog *Android* veidų detektorius yra kur kas spartesnis (apytiksliai 2,5 karto) ir sugeba tiksliau aptikti veidus (*JavaCV* detektorius nuotraukose kartais klaidingai identifikuodavo veidus). Įgyvendinus *Android* veidų detektorių buvo išsiaiškinta, jog nuotraukoje ieškomų veidų kiekis jų aptikimo spartai įtakos neturi. Pagrindinis veidų aptikimo spartą lemiantis kriterijus – nuotraukos raiška. Kai raiška viršija 640×320 px ribą, veidų aptikimo trukmė pradeda sparčiai augti (prie 320×160 px raiškos aptikimas trunka 219 ms, o prie 1600×800 px – apie 9000 ms) šie du parametrai yra tiesiogiai priklausomi vienas nuo kito. Ištyrus veidų aptikimo tikslumo priklausomybę nuo paveikslų raiškos buvo pastebėta, jog jei paveiksle yra daug veidų (pvz. 50), minimali paveikslo raiška, norint juos visus aptikti, turi būti 324×648 px. Tuo tarpu jei paveiksle yra tik vienas veidas – raišką galima mažinti net iki 40×50 px.

Įgyvendinant tikrinių veidų algoritmą *Android* operacinėje sistemoje buvo nuspręsta naudoti *OpenCV* biblioteką, kuri lyginant su *JavaCV* ir *OpenCV4Android* bibliotekomis yra populiariausia, sparčiausia ir patikrinta laiko. Tikrinių veidų algoritmas naudojantis šia biblioteka buvo įgyvendintas C++ programavimo kalba per *Android NDK* sąsają. Įgyvendinus algoritmą ir ištyrus atpažinimo santykio priklausomybę nuo paveikslų raiškos buvo pastebėta, jog kritinė raiškos riba yra 30×30 px – esant mažesnei paveikslų raiškai veidų atpažinimo santykis pradeda sparčiai mažėti. Išanalizavus naudojamas duomenų bazes bei atlikus tyrimą buvo pastebėta, jog veidų pozicija nuotraukoje labai įtakoja atpažinimo santykį – tinkamai išcentravus veidus nuotraukose (apkarpant nuotraukų kraštus) yra pastebimas stulbinantis atpažinimo santykio pagerėjimas (tam tikrose duomenų bazėse atpažinimo santykis išauga net iki 100 % ar 92 %). Pagal gautus rezultatus buvo nuspręsta veidų detektoriaus aptiktą veidą iš nuotraukos iškirpti vadovaujantis auksinio santykio taisykle – iškerpamo veido aukščio ir pločio santykis yra ⅔, kas beveik atitinka auksinę proporciją ($\approx 1{,}618$). Ištyrus kaip iškirptų veidų kokybė veikia atpažinimo santykį buvo pastebėta, jog paveikslams pritaikius pikselių intensyvumo korekciją ar histogramos išlyginimą yra gaunami kur kas geresni rezultatai. Didžiausią įtaką atpažinimo

santykiui daro pikselių intensyvumo korekcija (atpažinimo santykio padidėjimas 35,3 %), o mažiausią – adaptyvusis histogramos išlyginimas. Pritaikius adaptyvųjį histogramos išlyginimą Grimasų duomenų bazėje yra pastebimas atpažinimo santykio pagerėjimas iki 16 %, o mano duomenų bazėje – nuosmukis 7,7 %.

Pagal šiuose tyrimuose gautus rezultatus tikrinių veidų algoritmas *Android* operacinėje buvo įgyvendintas naudojant 320×240 px peržiūros langą, o visi veidai jam buvo teikiami koduojant juos 8 *bit* pilkų spalvų palete ir 90×60 px raiškos. Pagal šiuos parametrus veidų detektoriui vieno kadro apdorojimas ir paruošimas atpažinimo operacijai trunka 5–10 ms, o atpažinimo operacija apie 60 ms (esant tokiai trukmei per sekundę pavyksta apdoroti 17 kadrų). Patobulinus tikrinių veidų algoritmą, jog prieš apsimokymo ir atpažinimo operaciją kiekvienas veidas būtų iškirptas optimaliai bei jiems būtų pritaikoma pikselių intensyvumo korekciją ir histogramos išlyginimas – AT&T duomenų bazėje buvo pasiektas 85 % atpažinimo santykis, o likusiose dvejuose veidų duomenų bazėse visi veidai buvo identifikuoti teisingai.

LITERATŪRA

[1] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face recognition with local binary patterns. In Tomás Pajdla and Jirí Matas, editors, *Computer Vision - ECCV 2004*, volume 3021 of *Lecture Notes in Computer Science*, pages 469–481. Springer Berlin / Heidelberg, 2004.

[2] Peter N. Belhumeur, João P. Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection, 1997.

[3] Francesco Florio. SSD: Small Screen Design. About face detection on android. http://www.smallscreendesign.com/author/francesco/, 2011. Žiūrėta 2012 m. gegužę.

[4] H Gunes and M Piccardi. Assessing facial beauty through proportion analysis by image processing and supervised learning. *International Journal of Human-Computer Studies*, 64(12):1184–1199, 2006.

[5] Y. Jefferson. Facial beauty-establishing a universal standard. *IJO*, 15(1):9, 2004.

[6] H L Obwegeser and L J Marentette. Profile planning based on alterations in the positions of the bases of the facial thirds. *J Oral Maxillofac Surg*, 44(4):302–11, 1986.

[7] C. Sforza, A. Laino, R. Alessio, C. Dellavia, G. Grandi, and V. F. Ferrario. Three-dimensional facial morphometry of attractive children and normal children in the deciduous and early mixed dentition. *Angle Orthod*, 77(6):1025–33, 2007.

[8] A. J. O'Toole, P. J. Phillips, F. Jiang, J. Ayyad, N. Penard, and H. Abdi. Face recognition algorithms surpass humans matching faces over changes in illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(9):1642–1646, September 2007.

[9] IDTECK. What is face recognition? http://www.idteck.com/support/w_face.asp, 2009. Žiūrėta 2011 m. birželį.

[10] H. K. Ekenel, J. Stallkamp, H. Gao, M. Fischer, and R. Stiefelhagen. FACE RECOGNITION FOR SMART INTERACTIONS, 2008.

[11] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face Recognition: A Literature Survey, 2000.

[12] Ashish Tiwari. Pattern classification and analysis, face recognition, eigen-faces with 99 pca coefficients. http://courses.media.mit.edu/2004fall/mas622j/04.projects/students/Tiwari/Tiwari.ppt, 2004. Žiūrėta 2011 m. birželį.

[13] Yingchun Li, Guangda Su, and Yan Shang. Generating Optimal Face Image in Face Recognition System.

[14] P. Jonathon Phillips, Patrick Grother, Ross J. Micheals, Duane M. Blackburn, Elham Tabassi, Mike Bone, North Fairfax Dr, United Kingdom, P. Jonathon Phillips, Patrick Grother, Ross J. Micheals, Duane M. Blackburn, Elham Tabassi, and Mike Bone. FACE RECOGNITION VENDOR TEST 2002: EVALUATION REPORT, 2003.

[15] B. Karimi and A. Krzyzak. A Study on Significance of Color in Face Recognition using Several Eigenface Algorithms. In *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*, pages 1309–1312. IEEE, 2007.

[16] P. Cavanagh and Y.G. Leclerc. Shape from shadows. *Journal of Experimental Psychology: Human Perception and Performance*, 15(1):3, 1989.

[17] L. Sirovich and M. Kirby. Low-dimensional procedure for the characterization of human faces. *JOSA A*, 4(3):519–524, 1987.

[18] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[19] R. Kam-Art, T. Raicharoen, and V. Khera. Face recognition using feature extraction based on descriptive statistics of a face image. In *IEEE Machine Learning and Cybernetics, International Conference on*, volume 1, pages 193–197, 2009.

[20] Lindsay I Smith. A tutorial on principal components analysis. Technical report, Cornell University, USA, February 26 2002.

[21] K. Kim. Face recognition using principle component analysis. *Vision and AI Research Group Korean Graduate Students in CS@ UMD*, 2001.

[22] B.A. Draper, W.S. Yambor, and J.R. Beveridge. Analyzing pca-based face recognition algorithms: Eigenvector selection and distance measures. *Empirical Evaluation Methods in Computer Vision*, 2002.

[23] C. Smith III N. D. Rishe A.B. Selivonenko R. Steinhoff, A. Gallon. Improving eigenface face recognition by using image registration preprocessing methods. In *2nd International Conference On Cybernetics and Information Technologies, Systems and Applications, CITSA 2005*.

[24] Ronald Aylmer Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[25] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuro Science*, 3(1):71–86, 1991.

[26] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–590, Hawai, June 1992.

[27] L. Sirovich and M. Kirby. Low dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America*, 4(3):519–524, 1987.

[28] Y. Moses, Y. Adini, and S. Ullman. Face recognition: The problem of compensating for changes in illumination direction. In *ECCV*, pages A:286–296, 1994.

[29] Jyoti S. Bedre and Shubhangi Sapkal. Article: Comparative study of face recognition techniques: A review. *IJCA Proceedings on Emerging Trends in Computer Science and Information Technology (ETCSIT2012) etcsit1001*, ETCSIT(1):12–15, April 2012. Published by Foundation of Computer Science, New York, USA.

[30] Chong Lu, Senjian An, Wanquan Liu, and Xiaodong Liu. An innovative weighted 2dlda approach for face recognition. In Paisarn Muneesawang, Feng Wu, Itsuo Kumazawa, Athikom Roeksabutr, Mark Liao, and Xiaoou Tang, editors, *Advances in Multimedia Information Processing - PCM 2009*, volume 5879 of *Lecture Notes in Computer Science*, pages 110–118. Springer Berlin / Heidelberg, 2009.

[31] E. C. Ifeachor and B. W. Jervis. *Digital signal processing: a practical approach*. Prentice Hall, New York, NY, USA, 2nd edition, 2002.

[32] Y. Xiao L. Ma and K. Khorasani. A new facial expression recognition technique using 2d dct and k-means algorithm. International Conference on Image Processing, pages 1269–1272, 2004.

[33] L. Ma and K. Khorasani. Facial Expression Recognition Using Constructive Feedforward Neural Networks. *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, 34(3):1588–1595, 2004.

[34] Jawad Nagi. Design of an efficient high-speed face recognition system. 2007.

[35] Abdallah S. Abdallah, A. Lynn Abbott, and Mohamad Abou El-nasr. A new face detection technique using 2d dct and self organizing feature map.

[36] Ming-Hsuan Yang, David J. Kriegman, and Narendra Ahuja. Detecting faces in images: A survey. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 24(1):34–58, 2002.

[37] AT&T Laboratories Cambridge and The Digital Technology Group. At&t veidų duomenų bazė. http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html, 2011. Žiūrėta 2011 m. gegužę.

[38] Dr Libor Spacek. Grimasų veidų duomenų bazė. http://cswww.essex.ac.uk/mv/allfaces/index.html, 2011. Žiūrėta 2011 m. gegužę.

[39] Robin Hewitt. Seeing with opencv. *SERVO Magazine*, 2007.

[40] Erik Hjelmås and Boon Kee Low. Face detection: A survey. *Computer Vision and Image Understanding*, 83(3):236–274, 2001.

[41] V. Govindaraju. *A Computational Theory for Locating Human Faces in Photographs.* 1992.

[42] L.G. Farkas and I.R. Munro. *Anthropometric facial proportions in medicine.* Thomas, 1987.

[43] Ye-Peng Guan. Unsupervised human height estimation from a single image. *Journal of Biomedical Science and Engineering*, 2009.

[44] Marie-Pierre Dubuisson and Anil K. Jain. A Modified Hausdorff Distance for Object Matching, 1994.

[45] Oliver Jesorsky, Klaus J. Kirchberg, and Robert W. Frischholz. Robust Face Detection Using the Hausdorff Distance. In , pages 90–95. Springer, 2001.

[46] James L. Crowley and Joëlle Coutaz. Vision for man machine interaction. In *Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction*, pages 28–45, London, UK, UK, 1996. Chapman & Hall, Ltd.

[47] Sanjay Kr. Singh, D. S. Chauhan, Mayank Vatsa, and Richa Singh. A Robust Skin Color Based Face Detection Algorithm, Tamkang. *Journal of Science and Engineering*, 6:227–234, 2003.

[48] S. Shaposhnikov D. Comley R. Anishenko and X. Gao. Facial image segmentation based on mixed colour space. In *In proc.: XV International Conference on Neurocybernetics (ICNC'09).*

[49] Dirk-Jan Kroon. NUMERICAL OPTIMIZATION OF KERNEL BASED IMAGE DERIVATIVES Dirk-Jan Kroon University of Twente , Enschede. *Area*, (December), 2009.

[50] Or Vincent and O Folorunso. A Descriptive Algorithm for Sobel Image Edge Detection. *Most*, pages 97–107, 2009.

[51] A Albiol, L Torres, and E J Delp. An unsupervised color image segmentation algorithm for face detection applications. *IEEE Image Processing IP*, 2:681–684, 2001.

[52] M. Fan Z. Padilla. Automatic Face Detection Using Color Based Segmentation and Template/Energy Thresholding. *Stanford University*, 2003.

[53] H.C. Vijay Lakshmi and S. Patil Kulakarni. Segmentation Algorithm for Multiple Face Detection in Color Images with Skin Tone Regions using Color Spaces and Edge Detection Techniques. *International Journal of Computer Theory and Engineering*, 2010.

[54] Vladimir Vezhnevets, Vassili Sazonov, and Alla Andreeva. A Survey on Pixel-Based Skin Color Detection Techniques. In *IN PROC. GRAPHICON-2003*, pages 85–92, 2003.

[55] H. Wu, Q. Chen, and M. Yachida. Facial Feature Extraction and Face Verification. In *Proceedings of the International Conference on Pattern Recognition (ICPR '96) Volume III-Volume 7276 - Volume 7276*, ICPR '96, pages 484–, Washington, DC, USA, 1996. IEEE Computer Society.

[56] A. Gunduz and Hamid Krim. Facial feature extraction using topological methods. In *ICIP (1)*, pages 673–676, 2003.

[57] Paul Viola and Michael Jones. Robust Real-Time Object Detection. In *Second International Workshop on Statistical and Computational Theories of Vision – Modeling, Learning, Computing, and Sampling*, 2001.

[58] A. Bastys. Veidų atpažinimas. https://kedras.mif.vu.lt/bastys/academic/ATE/veidai/veidoAtp.htm#pi, 2009. Žiūrėta 2011 m. birželį.

[59] Lale Akarun. Face recognition using principal components analysis. http://www.cmpe.boun.edu.tr/courses/cmpe360/spring2007/files/PCA/project_spec.pdf, 2007. Žiūrėta 2011 m. birželį.

[60] Yao-Jiunn Chen and Yen-Chun Lin. Simple face-detection algorithm based on minimum facial features. *IECON 2007 33rd Annual Conference of the IEEE Industrial Electronics Society*, pages 455–460, 2007.

[61] A. Kinage K.S. Bhirud Chaudhari, S. Kale. Face feature detection and normalization based on eyeball center and recognition. In *Future Computer and Communication (ICFCC), 2010 2nd International Conference.*

[62] Viola-jones object detection framework. http://en.wikipedia.org/wiki/Viola-Jones_object_detection_framework, 2011. Žiūrėta 2011 m. birželį.

[63] Egidijus Kubickas. Kalbančiojo lūpų formos registravimas. Master's thesis, Šiaulių universitetas, June 2007.

[64] Ming hsuan Yang. Face recognition using kernel methods, 2001.

[65] Gregory Shakhnarovich and Baback Moghaddam. Face recognition in subspaces. pages 141–168, 2004.

[66] Chengjun Liu and Harry Wechsler. Enhanced fisher linear discriminant models for face recognition. pages 17–20, 1998.

[67] E. Hjelmas and J. Wroldsen. Recognizing faces from the eyes only. 1999.

[68] Wen-Yi Zhao. Face recognition and modeling tutorial (part i). In *8th European Conference on Computer Vision*, May 2004.

[69] Thomas Vetter and Sami Romdhani. Face modelling and recognition tutorial (part ii). In *8th European Conference on Computer Vision*, May 2004.

[70] Santiago Serrano. Drexel University Robotics Laboratory. Eigenface tutorial. http://www.pages.drexel.edu/~sis26/Eigenface%20Tutorial.htm. Žiūrėta 2011 m. birželį.

[71] J.Krueger, M.B.Robinson, D.Kochelek, and M.Escarra. Obtaining the eigenface basis. http://cnx.org/content/m12531/latest/, 2011. Žiūrėta 2011 m. birželį.

[72] Patala Arun Kumar. Eigenface algorithm for face recognition. http://home.iitk.ac.in/~parun/cs365/hw1.html, May 2011. Žiūrėta 2011 m. gegužę.

[73] Wikipedia.org. Tikriniai veidai. http://en.wikipedia.org/wiki/Eigenface, May 2011. Žiūrėta 2011 m. gegužę.

[74] Derzu Omaia, JanKees v.d. Poel, and Leonardo V. Batista. 2d-dct distance based face recognition using a reduced number of coefficients. *Graphics, Patterns and Images, SIBGRAPI Conference on*, 0:291–298, 2009.

[75] Valentin Suciu. Facial recognition using opencv. *Journal of Mobile, Embedded and Distributed Systems*, 4(1), 2012.

[76] G.Dave, X.Chao, and K.Sriadibhatla. Face recognition in mobile phones. Department of Electrical Engineering, Stanford University, Stanford, USA, 2010.

PRIEDAI

A. SANTRUMPOS

2D – dvimatis, dviejų matmenų (angl. *two–dimensional*)

LDA – tiesinio diskriminanto analizė (angl. *Linear Discriminant analysis*)

FLD – Fišerio tiesinis diskriminantas (angl. *Fisher Linear Discriminant*)

2DLDA – dvimačio tiesinio diskriminanto analizė (angl. *two–dimensional linear discriminant analysis*)

DCT – diskrečioji kosinuso transformacija (angl. *discrete cosine transform*)

2D–DCT – dvimatė diskrečioji kosinuso transformacija (angl. *two–dimensional discrete cosine transform*)

2D–iDCT – dvimatė atvirkštinė diskrečioji kosinuso transformacija (angl. *two–dimensional inverse discrete cosine transform*)

SOM – Save organizuojantis neuronų tinklas, kartais dar vadinamas Kohoneno žemėlapiu (angl. *Self–organizing map*)

DNT – dirbtinis neuronų tinklas

NN – neuronų tinklas (angl. *neural network*)

NDK – Prigimtinis programinės įrangos kūrimo įrankių rinkinys (angl. *native development kit*)

SDK – programinės įrangos kūrimo įrankių rinkinys (angl. *software development kit*)

API – taikomojo programavimo sąsaja (angl. *Application programming interface*)

JNI – *Java* programavimo kalbos prigimtinė sąsaja (angl. *Java Native Interface*)

LBP – lokalūs binariniai šablonai (angl. *Local Binary Patterns*)

LR – panašumo santykis, tikimybė (angl. *Likelihood Ratio*)

LLR – logaritminis panašumo santykis, logaritminė tikimybė (angl. *Log Likelihood Ratio*)

FFT – greitoji Furje transformacija (angl. *Fast Fourier transform*)

PCA – principinių komponenčių analizė (angl. *Principal component analysis*)

EFM – patobulintas Fišerio tiesinio diskriminanto modelis (angl. *Enhanced Fisher linear discriminant Model*)

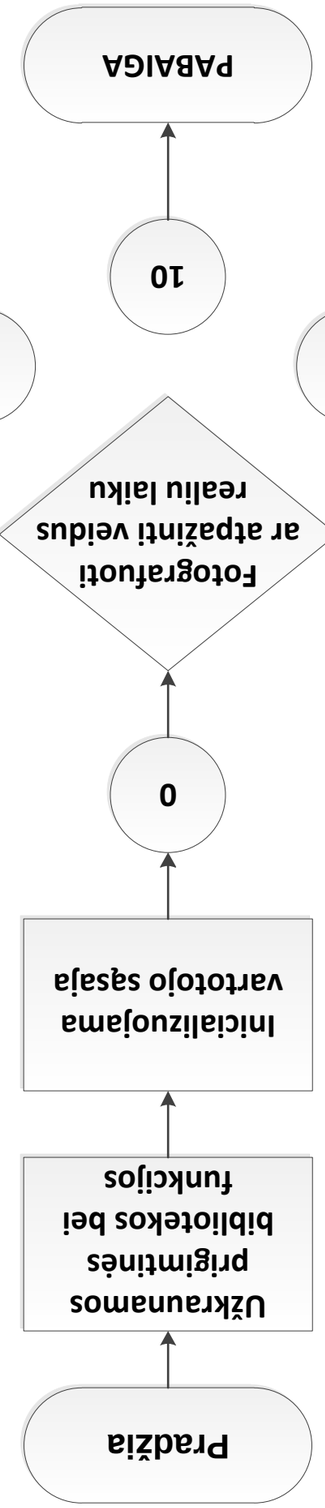
SVM – atraminių vektorių klasifikatoriai (angl. *Support vector machine*)

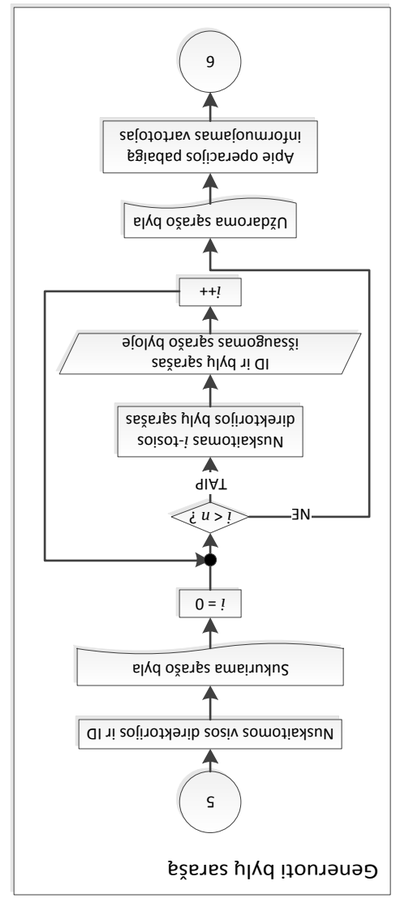
VM – virtuali mašina (angl. *Virtual machine*)

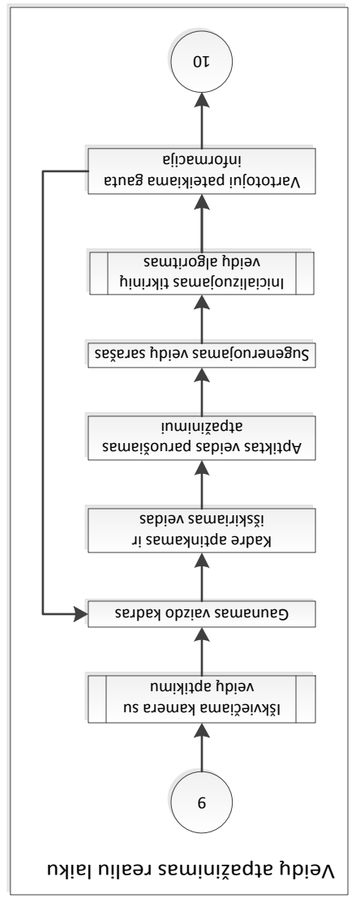
CV – kompiuterinė rega (angl. *Computer vision*)

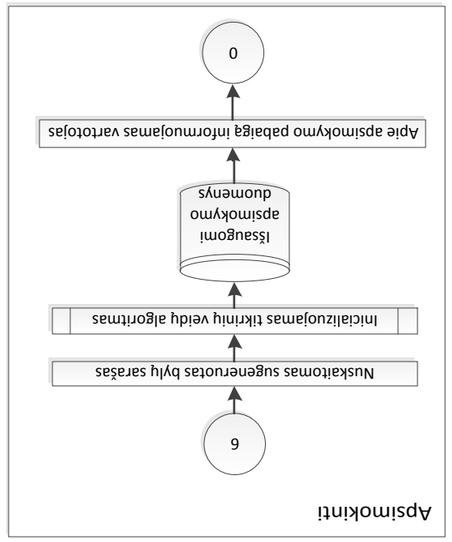
OS – operacinė sistema

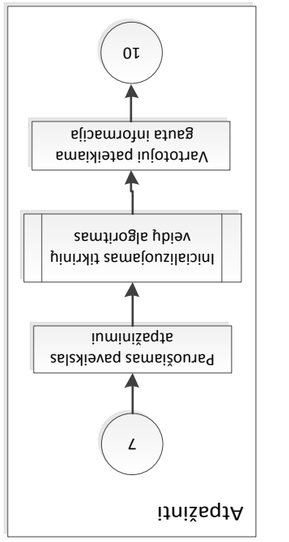
B. Algoritmų ir taikomosios programos schemos

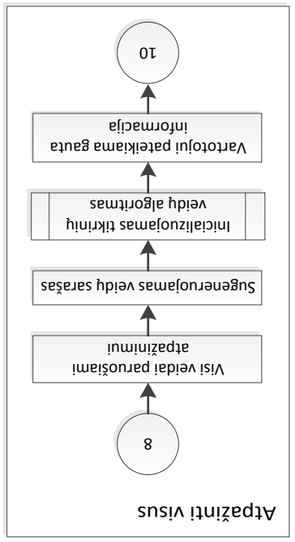


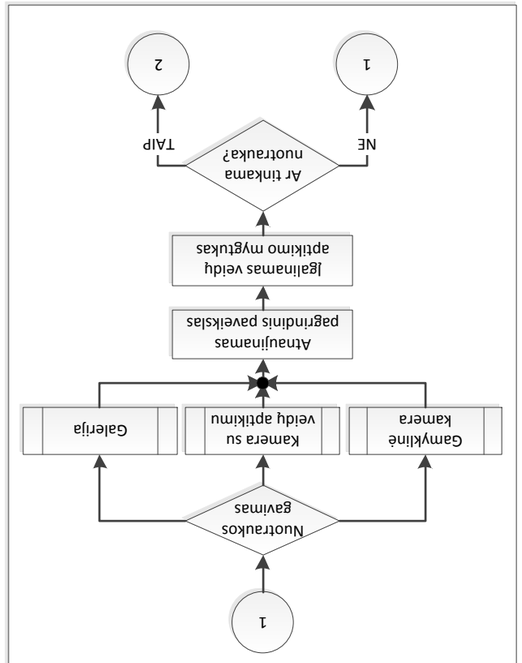


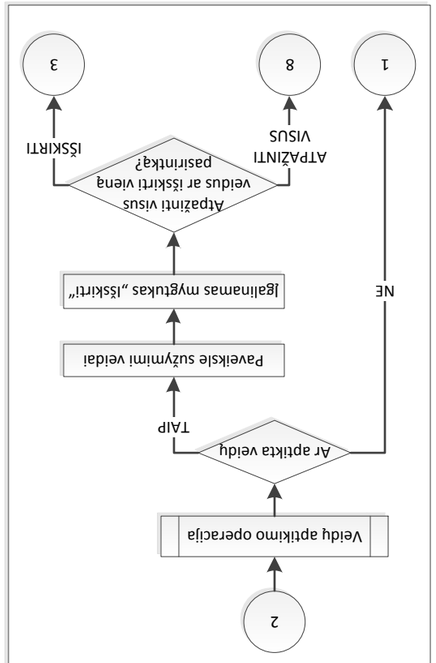


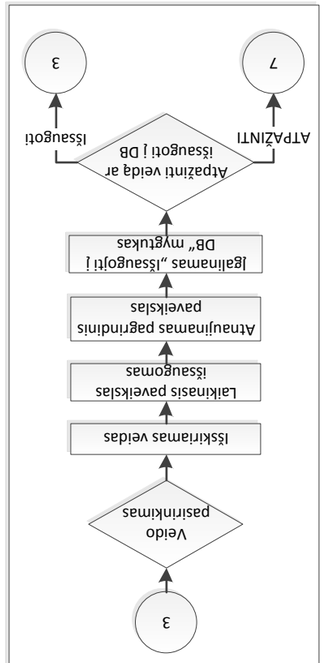


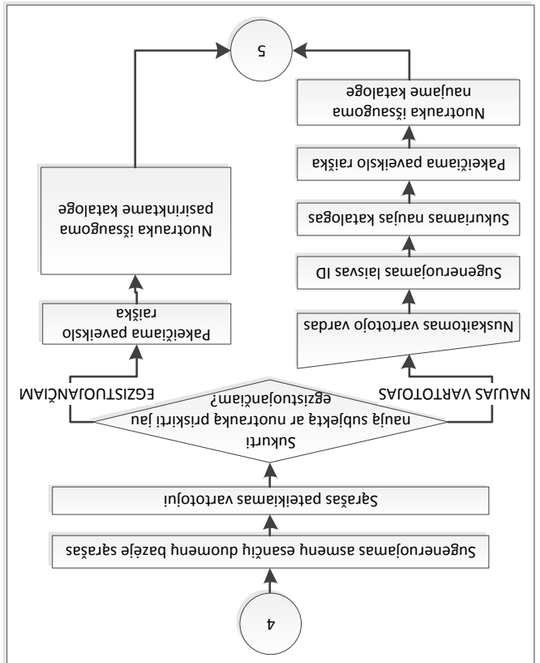












Pak.	Lapas	Dokumento Nr.	Parašas	Data
Sudarė		J. Balinskas		
Tikrino		A. Ūsinskas		
Konsultavo		A. Ūsinskas		

VGTU 2012-05-29.001 E3

Taikomoji veidų atpažinimo programa
Struktūrinė veikimo schema

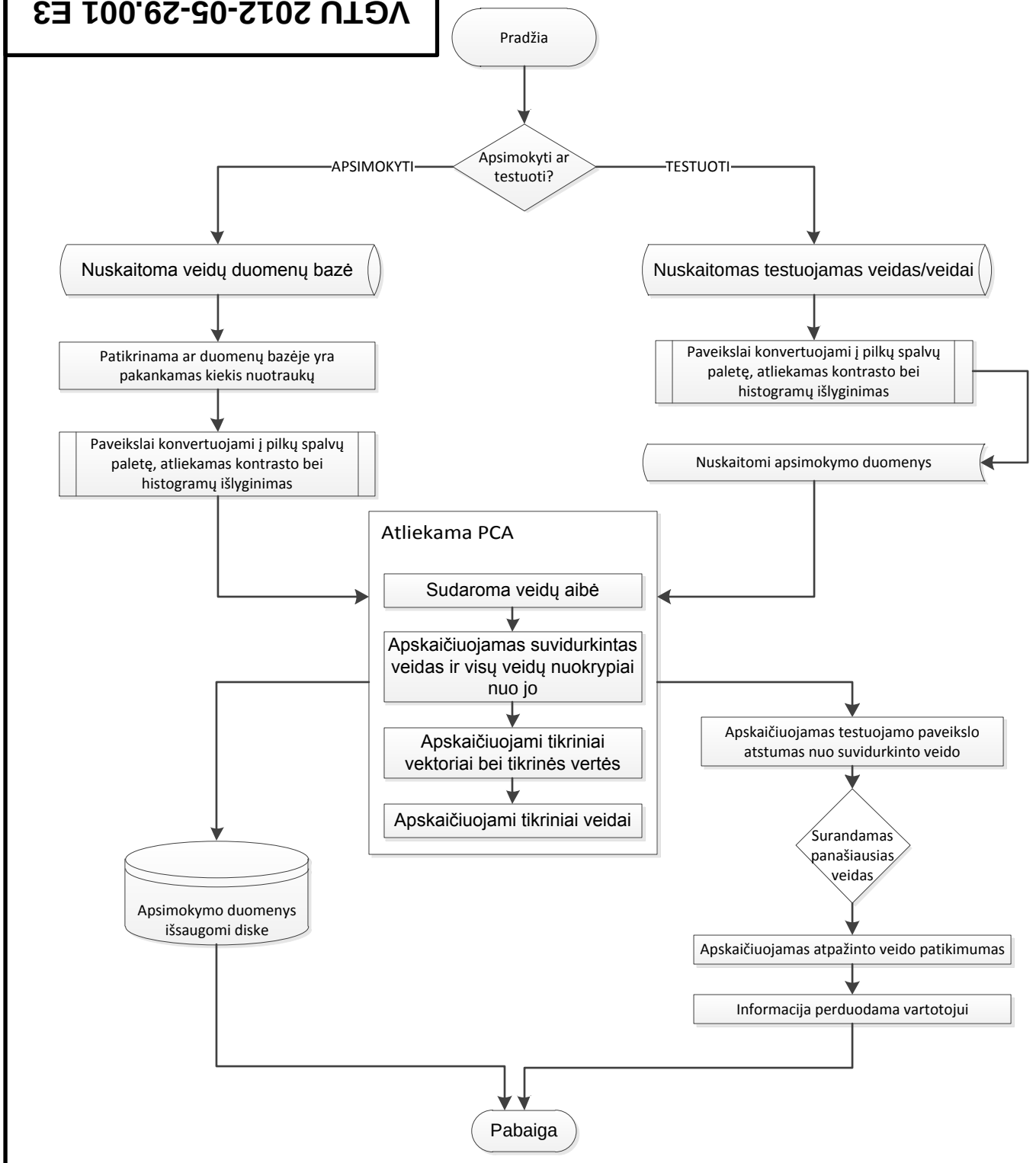
Litera	Masė	Mastelis
D		

Lapas 1	Lapų 2

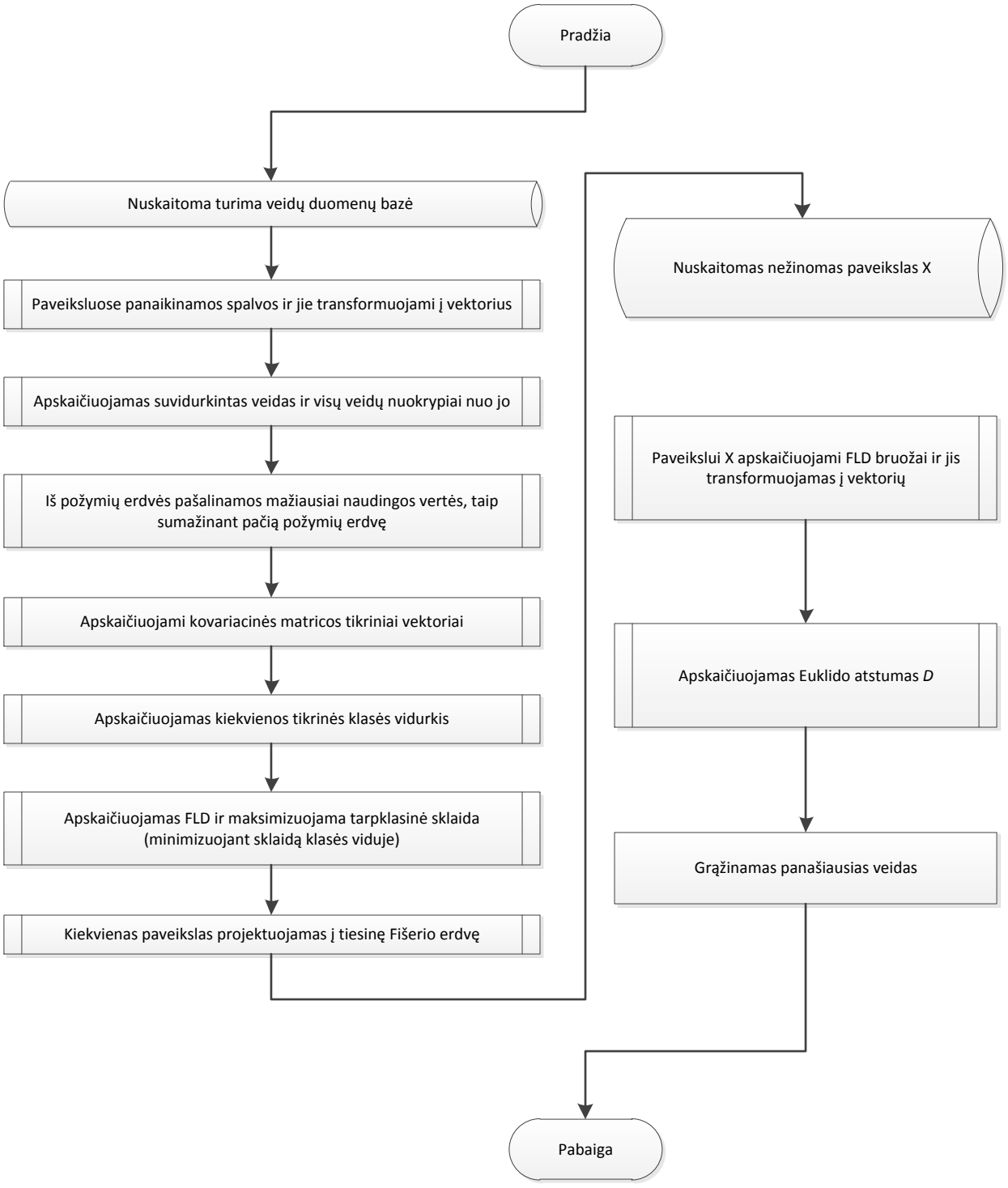
SASfm-10

VGTU 2012-05-29.001 E3

VGTU 2012-05-29.001 E3



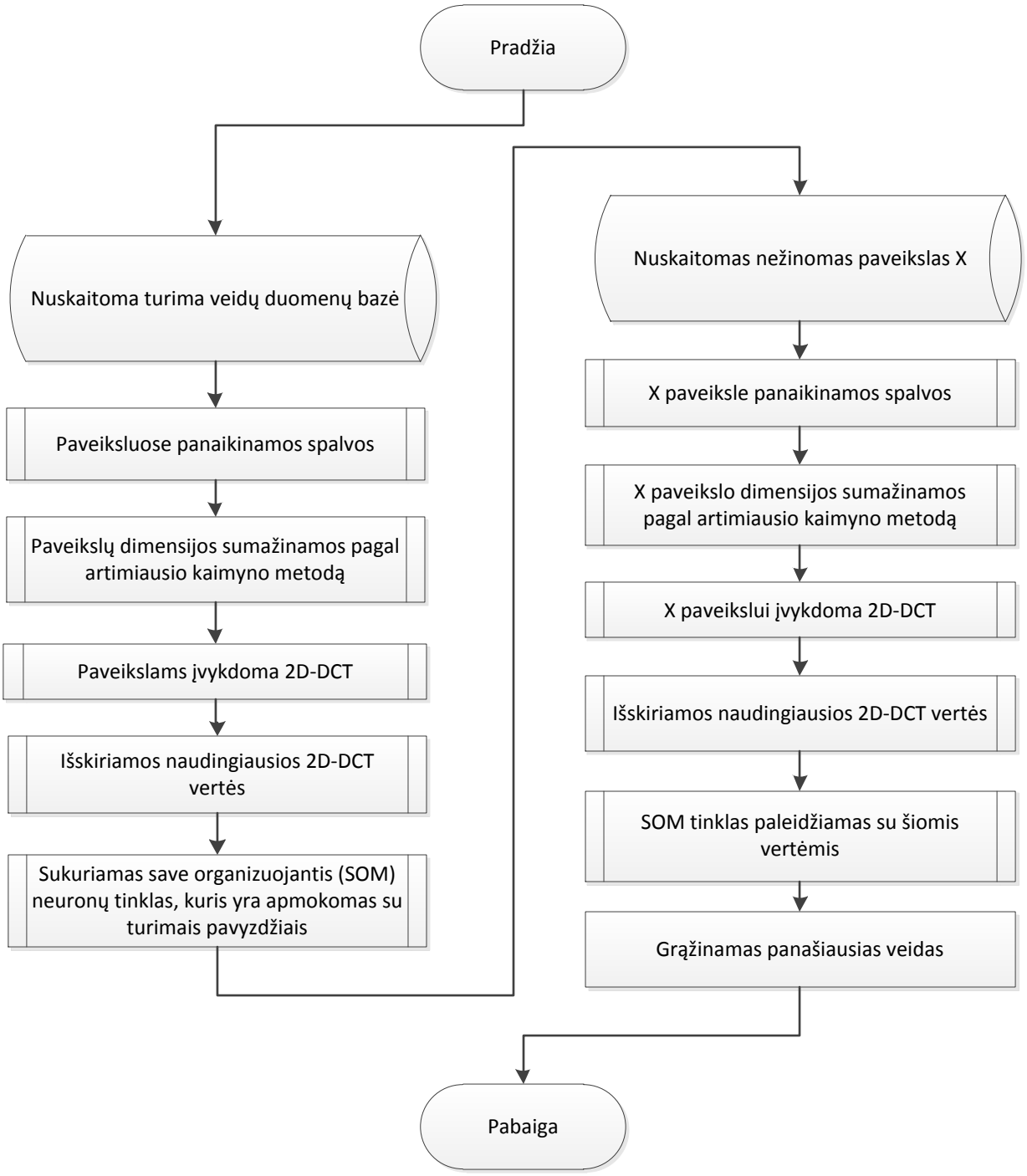
					VGTU 2012-05-29.001 E3	
					Tikrinių veidų algoritmas	Litera: D; Masė: ; Mastelis:
Pak.	Lapas	Dokumento Nr.	Parašas	Data		
Sudarė		J. Balinskas			Struktūrinė algoritmo schema	Lapas 2 / Lapų 2
Tikrino		A. Ušinskas				
Konsultavo		A. Ušinskas				SASfm-10



					VGTU 2012-05-29.002 E3
Pak.	*Lapas*	*Dokumento Nr.*	*Parašas*	*Data*	Fišerio veidų algoritmas
Sudarė		J. Balinskas			Struktūrinė algoritmo schema
Tikrino		A. Ušinskas			
Konsultavo		A. Ušinskas			

Litera	*Masė*	*Mastelis*
D		
Lapas 1	*Lapų 1*	

SASfm-10



					VGTU 2012-05-29.003 E3	
Pak.	Lapas	Dokumento Nr.	Parašas	Data	2D-DCT+SOM algoritmas	Litera / Masė / Mastelis
Sudarė		J. Balinskas				D
Tikrino		A. Ušinskas			Struktūrinė algoritmo schema	Lapas 1 / Lapų 1
Konsultavo		A. Ušinskas				SASfm-10

C. Pranešimo LJMK „Elektronika ir elektrotechnika 2012“ medžiaga

PRANEŠIMO SKAIDRĖS

Vilniaus Gedimino technikos universitetas
Elektronikos fakultetas
Informacinių sistemų katedra
SASfm-10 grupė

Jaunųjų Mokslininkų Konferencija
„Elektronika ir elektrotechnika 2012“

VEIDŲ ATPAŽINIMO ALGORITMŲ TYRIMAS

Autorius Justinas Balinskas
Vadovas doc. dr. Andrius Ušinskas

Darbo tikslas

Uždavinys: Įgyvendinti ir ištirti tris veidų atpažinimo algoritmus

Problema: Išrinkti geriausią algoritmą ir nustatyti optimalius jo parametrus

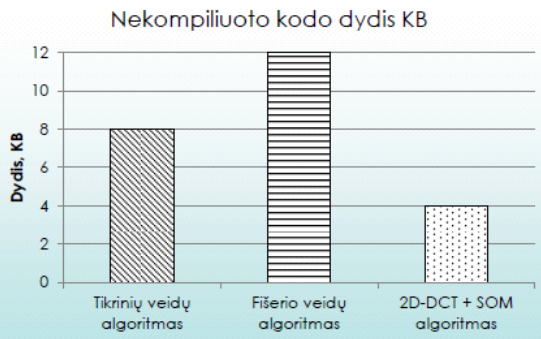
Naudojamos veidų duomenų bazės

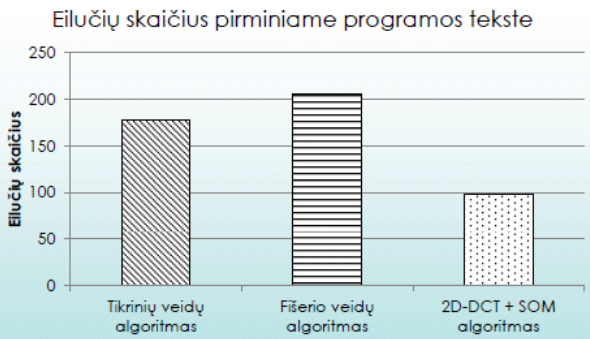
- Mano duomenų bazė
- AT&T duomenų bazė
- Grimasų duomenų bazė

Pasirinkti algoritmai

- Tikrinių veidų
- Fišerio veidų
- 2D-DCT + SOM

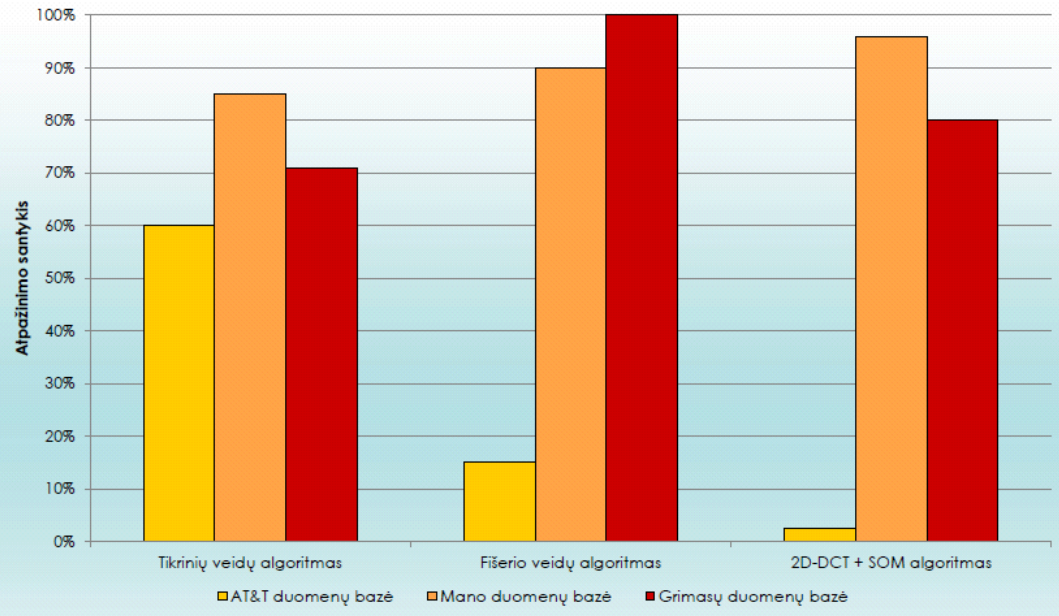
Algoritmų įgyvendinimo sudėtingumas



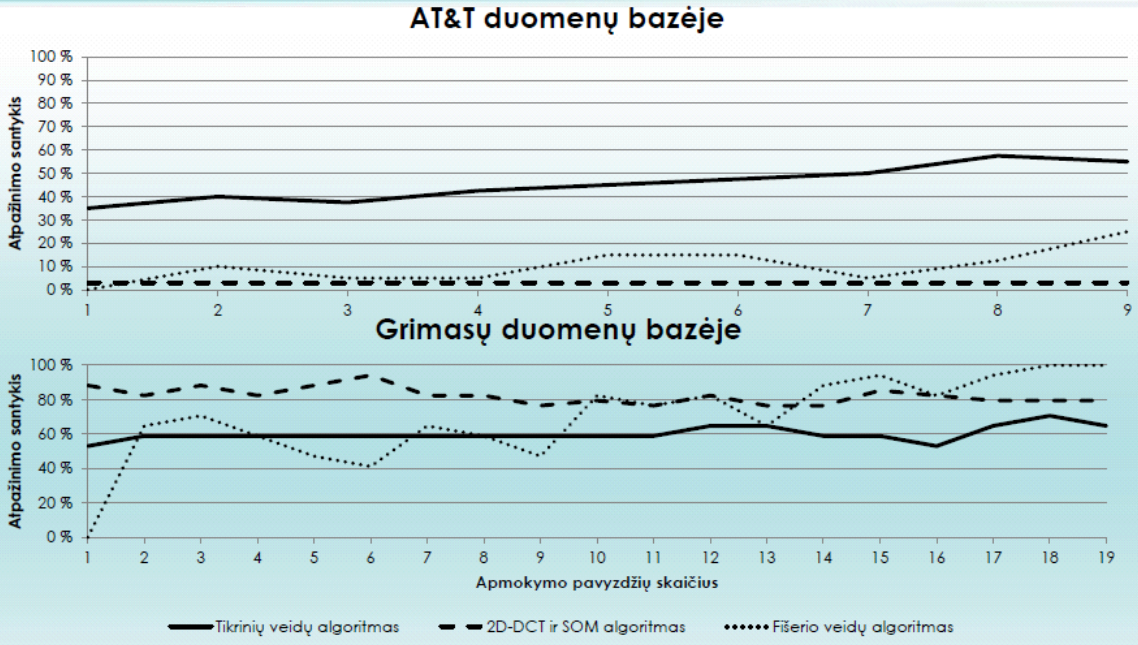


Tikrinių veidų algoritmas	Fišerio veidų algoritmas	2D-DCT+SOM algoritmas
rgb2gray()	**rgb2gray()**	**rgb2gray**
norm()	**norm()**	dct2()
eig()	**eig()**	idct2()
mean()	**mean()**	newsom()
reshape()	**reshape()**	sim()
Operacijos su matricomis	**Operacijos su matricomis**	
std()	fliplr()	
	repmat()	
	circshift()	

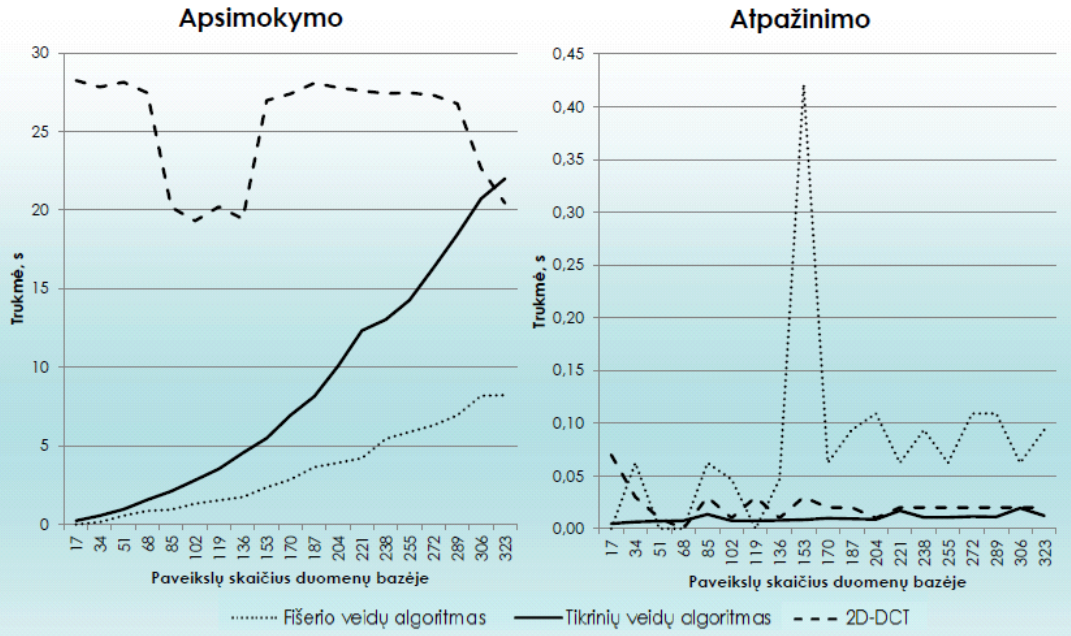
Algoritmų atpažinimo santykis



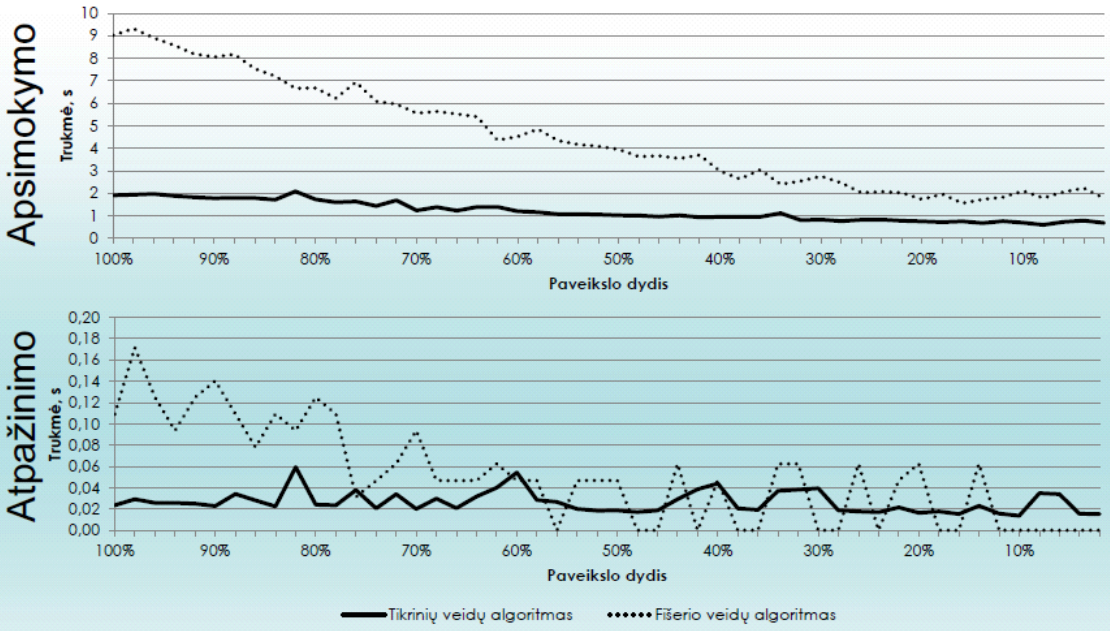
Algoritmų tikslumo priklausomybė nuo pavyzdžių skaičiaus vienam asmeniui



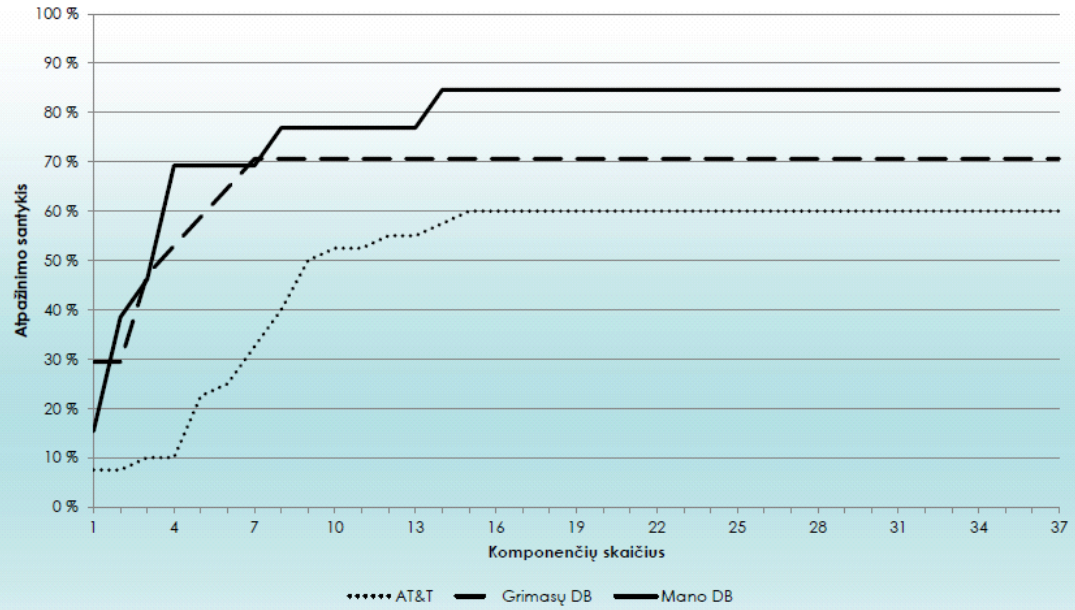
Trukmės priklausomybė nuo paveikslų skaičiaus



Trukmės priklausomybė nuo paveikslų dydžio



Tikrinių veidų algoritmo atpažinimo santykio priklausomybė nuo komponenčių skaičiaus



Rezultatų apibendrinimas

- Šio tyrimo metu buvo ištirti trys veidų atpažinimo algoritmai trijose veidų duomenų bazėse
- „Lengviausias“ algoritmas (KB): 2D-DCT+SOM algoritmas
- Tikrinių veidų algoritmas yra tiksliausias bei geriausiai atpažįsta veidus esant mažam pavyzdžių vienam asmeniui kiekiui
- Geriausias atpažinimo santykis pasiekiamas kai vienas asmuo turi daugiau nei 15 save apibūdinančių nuotraukų
- Paveikslų skaičius duomenų bazėje tiesiogiai įtakoja apsimokymo trukmę, o atpažinimo trukmė praktiškai nekinta
- Paveikslų dydis mažiausiai įtakoja tikrinių veidų algoritmą
- Sparčiausiai atpažinimas veikia tikrinių veidų algoritme (10 ms – 50 ms vienam veidui atpažinti)
- Tikrinių veidų algoritme geriausi rezultatai gaunami naudojant 15 komponenčių

Ačiū už dėmesį!

DALYVIO PAŽYMOS KOPIJA

XV-toji Lietuvos jaunųjų mokslininkų konferencija
„MOKSLAS – LIETUVOS ATEITIS“

ELEKTRONIKA IR ELEKTROTECHNIKA

PAŽYMA

Justinas Balinskas

Vaizdų technologijų ir sistemų inžinerijos, kompiuterių technologijų jungtinėje sekcijoje perskaitė pranešimą

VEIDŲ ATPAŽINIMO ALGORITMŲ TYRIMAS

Mokslo komiteto pirmininkas prof. habil. dr. Romanas Martavičius

Sekcijos pirmininkas dr. Artūras Serackis

2012 m. kovo mėn. 16 d.
Vilnius

D. Pasirinktų algoritmų įgyvendintų *MATLAB* aplinkoje pirminis programos tekstas

TIKRINIŲ VEIDŲ ALGORITMAS

run.m

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %% Plaeidziama programa %%
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  % Sutvarkoma darbo aplinka
6  clear all;
7  close all;
8  clc;
9
10 % Sukuriamas programos meniu
11 choice = menu('Eigenfaces', 'Apmokyti programa','Testuoti program
       ','Iseiti');
12 switch choice
13   % ALGORITMO APMOKYMAS
14     case 1
15     PathTrainDatabase = uigetdir('./TrainDatabase', 'Pasirinkite
           kataloga su apmokymo pavyzdziais' );
16
17     [ MeanFace, NuokrypisNuoMeanFace, Eigenfaces, Vardai,
           EigenVectors, VeiduAibe, img_dim ] = Train(
           PathTrainDatabase);
18
19     save eigenfaces.mat;
20   % ALGORITMO TESTAVIMAS
21   case 2
22     [file path] = uigetfile({'*.jpg;*.jpeg;*.png;*.tiff;*.bmp','
           All Image Files'; '*.*','All Files' }, 'Pasirinkite
           testavimo paveiksla', 'TestDatabase');
23     PathTestImage = [ path file ]; clear file path;
24     load 'eigenfaces.mat';
25         [ OutputName atstumai ] = Recognize(PathTestImage,
               MeanFace, Eigenfaces, Vardai, EigenVectors, VeiduAibe,
                img_dim);
26     OutputName
27   % ISEJIMAS IS PROGRAMOS
28   case 3
29     return;
30 end
31
32 clear choice;
```

train.m

```
1  function [ MeanFace, NuokrypisNuoMeanFace, Eigenfaces, Vardai,
       EigenVectors, VeiduMatrica, img_dim ] = Train(
       ApmokymoKatalogas)
2
3    %% Suzinomas bylu pletinys
4    dlg_text = {'Iveskite naudojama bylu pletini:\nPvz.: ".jpg"'};
5    dlg_title = 'Bylu pletinys';
6    def_val_dlg = {'.jpg'};
7    extension = inputdlg(dlg_text, dlg_title, 1, def_val_dlg);
8    extension = extension{1};
9
10   %% Nuskaitomos reikiamos bylos
11   ApmokymoFailai = dir(strcat(ApmokymoKatalogas, '/*', extension)
         );
12
13   %% Suzinomas N, kuris naudojamas bus veliau
14   dlg_text = {sprintf('Iveskite principiniu komponenciu skaiciu
         paveikslui.\nJis turi buti <= %d', size(ApmokymoFailai,1))};
15   dlg_title = 'Komponenciu skaicius paveikslui';
16   def_val_dlg = {'20'};
17   dlg_answer = inputdlg(dlg_text, dlg_title, 1, def_val_dlg);
18
19   %% ijungiamas taimeris
20   tic_ID = tic; % pradedamas skaiciuoti laikas
21   t1 = cputime;
22
23   ApmokymuSkaicius = size(ApmokymoFailai, 1);
24
25   %% Sukuriamas vardu vektorius
26   Vardai = [];
27   for i = 1 : ApmokymuSkaicius
28     Vardai{i} = ApmokymoFailai(i).name(1:end-4);
29   end
30
31   %% Sukuriama 2D matrica sudaryta  is vektoriu atitinkanciu
         apmokymo pavyzdzius
32   for i = 1 : ApmokymuSkaicius
33     str = strcat(ApmokymoKatalogas,'/',ApmokymoFailai(i).name);
34     img = imread(str);
35     if size(size(img), 2) > 2
36       img = rgb2gray(img);
37     end
38     VeiduMatrica(:,i) = reshape(img,size(img,1)*size(img,2),1);
           % 2D paveikslai konvertuojami i 1D vektorius
39   end
40
41   irow = size(img,1);
42   icol = size(img,2);
43   img_dim = [irow icol];
```

```
44
45    %% Normalizuojami pavyzdziai
46    for i = 1 : size(VeiduMatrica,2)
47      temp = double(VeiduMatrica(:,i));
48      m = mean(temp);
49      st = std(temp);
50      VeiduMatrica(:,i) = (temp-m) * 80 / st+ 100;
51    end
52
53 %   %% Atvaizduojami visi veidai
54 %    figure()
55 %    for i = 1:ApmokymuSkaicius
56 %      subplot(ceil(sqrt(ApmokymuSkaicius)),fix(sqrt(
       ApmokymuSkaicius)),i);
57 %      imshow( reshape( VeiduMatrica(:,i), img_dim ) );
58 %    end
59
60    %% Apskaiciuojamas SuvidurkintasVeidas
61    MeanFace = mean(VeiduMatrica, 2);
62    % MeanFace = sum(VeiduMatrica')' / size(VeiduMatrica, 2);
63
64 %   %% Atvaizduojamas suvidurkintas veidas
65 %    figure('Name', 'MeanFace');
66 %    imshow(cast(reshape(MeanFace, img_dim), 'uint8'));
67
68
69    %% Apskaiciuojamas kiekvieno paveikslo nuokrypis nuo
         suvidurkinto paveikslo
70    NuokrypisNuoMeanFace = [];
71    for i = 1 : ApmokymuSkaicius % Apskaiciuojamas skirtumas tarp
         kiekvieno paveikslo ir suvidurkinto veido
72      temp = double(VeiduMatrica(:,i)) - MeanFace;
73      NuokrypisNuoMeanFace = [NuokrypisNuoMeanFace temp];
74    end
75    %% ANTRAS BUDAS
76    % o = ones(1,size(VeiduMatrica,2));
77    % NuokrypisNuoMeanFace2 = double(VeiduMatrica) - ( MeanFace * o
         );
78
79
80    %% Apskaicuojami EigenValues ir EigenVectors
81    L = NuokrypisNuoMeanFace'*NuokrypisNuoMeanFace; %
         Apskaiciuojama kovariacine matrica C=A*A'.
82    [EigenVectors, EigenValues] = eig(L);
83
84
85    %% Isskiriami kiekvieno veido bruozai (apskaiciuojami
         eigenveidai)
86    N = str2num(dlg_answer{1}); % Naudojamu komponenciu skaicius (
         Negali buti daugiau nei yra apmokymo pavyzdziu)
87
88    EigenVectors = NuokrypisNuoMeanFace * EigenVectors;
```

```
89    EigenVectors = EigenVectors(:,end:-1:end-(N-1)); % Isrenkami
          eigenvektoriai pagal vertingiausias eigenvertes
90
91    Eigenfaces = zeros(size(VeiduMatrica,2),N);
92    for i=1:size(VeiduMatrica,2);
93      Eigenfaces(i,:)=NuokrypisNuoMeanFace(:,i)' * EigenVectors; %
            Kikeviena eilute yra paveikslo parasas
94    end
95
96    %% Sustabdomas taimeris
97    run_time = toc(tic_ID);
98    t2 = cputime - t1;
99
100 %    figure('Name', 'Eigen veidai');
101 %    for i = 1:ApmokymuSkaicius
102 %      subplot(ceil(sqrt(ApmokymuSkaicius)),fix(sqrt(
        ApmokymuSkaicius)),i);
103 %      imshow( cast( reshape( Eigenfaces(:,i), img_dim )', 'uint8
        ') ); title(Vardai(i))
104 %    end
105
106   %% i ekrana isvedama informacija susijusi su apsimokymu
107   pranesimas = sprintf('Apmokymas baigtas.');
108   pranesimas = strcat( pranesimas, sprintf('\nTrukme:\t%f sec',
          run_time));
109   pranesimas = strcat( pranesimas, sprintf('\nCPUTime:\t%g sec',
          t2));
110   pranesimas = strcat( pranesimas, sprintf('\nPavyzdziu sk.:\t%d'
          , ApmokymuSkaicius));
111   pranesimas = strcat( pranesimas, sprintf('\nPaveikslo dim.:\t%
          dx%d', img_dim));
112   msgbox(pranesimas);
```

recognize.m

```
1  function [OutputName Vardai] = Recognize(TestImage, MeanFace,
       Eigenfaces, Vardai, EigenVectors, VeiduMatrica, img_dim)
2  load eigenfaces.mat;
3
4  % ijungiamas taimeris
5  tic_ID = tic;
6  t1 = cputime;
7
8  %% Isskiriami testuojamo paveikslo PCA bruozai
9  InputImage = imread(TestImage);
10 if size(size(InputImage), 2) > 2
11   InputImage = rgb2gray(InputImage);
12 end
13
14 % i ekrana isvedamas originalus paveikslas
15 figure(1); subplot(1, 3, 1);imshow(InputImage); title('Original')
       ;
16
17 [irow icol] = size(InputImage);
18 InputImage = reshape(InputImage,irow*icol,1);
19
20 %% Apskaiciuojamas atstumas
21 Difference = double(InputImage) - MeanFace;
22
23 % I ekrana isvedamas atstumas tarp atstumas tarp testuojamo ir
       suvidurkinto veido
24 figure(1);subplot(1, 3, 2); imshow( cast( reshape( Difference
       (:,1), img_dim ), 'uint8') ); title('Difference');
25
26 % Sudaromas testuojamo paveikslo bruozu vektorius
27 ProjectedTestImage = Difference' * EigenVectors;
28
29 % Apskaiciuojamas euklido atstumas
30 Euc_dist =[];
31 for i=1:size(VeiduAibe,2)
32     Euc_dist = [ Euc_dist, norm( Eigenfaces(i,:) -
           ProjectedTestImage, 2 ) ];
33 end
34
35 % Surandamas paveikslas, kurio Euklido atstumas maziausias
36 [Euc_dist_min , Recognized_index] = min(Euc_dist);
37 OutputName = Vardai(Recognized_index);
38 %disp('Euc_dists:');disp(Euc_dist');
39 %disp('Recognized index:'); disp(Recognized_index);
40 %disp('min dist:'); disp(Euc_dist_min);
41
42 % sustabdomas taimeris
43 run_time = toc(tic_ID);
44 t2 = cputime - t1;
```

```
45
46 % atvaizduojamas atpazintas paveikslas
47 figure(1);subplot(1, 3, 3); imshow( reshape(VeiduAibe(:,
       Recognized_index), img_dim)); title('Recognized');
48
49 % Atvaizduoja euklido atstumo pasiskirstyma
50 figure(2);
51 stem(Euc_dist);
52
53 %%% Atkuriamas paveikslas
54 % p = [];
55 % aa=size(EigenVectors,2);
56 % for i = 1:aa
57 %     pare = dot(double(InputImage),EigenVectors(:,i));
58 %     p = [p; pare];
59 % end
60 % reshaped = MeanFace + EigenVectors(Recognized_index,:) * p;
61 % reshaped = reshape(reshaped, img_dim);
62 % reshaped = reshaped * -1 / 100000000000;
63 % minmax(minmax(reshaped')')
64 % figure(1);subplot(1, 4, 4);imshow(cast(reshaped, 'uint8'));
       title('Restored');
65
66 %%% Parodo svoriu pasiskirstyma
67 % InImWeight = [];
68 % for i=1:size(EigenVectors,2)
69 %     t = EigenVectors(:,i)';
70 %     WeightOfInputImage = dot(t,Difference');
71 %     InImWeight = [InImWeight; WeightOfInputImage];
72 % end
73 % ll = 1:37;
74 % figure()
75 % subplot(1,2,1)
76 % stem(ll,InImWeight)
77 % title('Weight of Input Face','fontsize',14)
78
79 % i ekrana isvedama informacija susijusi su apsimokymu
80 pranesimas = sprintf('Atpazinimas baigtas.');
81 pranesimas = strcat( pranesimas, sprintf('\nTrukme:\t%f sec',
       run_time));
82 pranesimas = strcat( pranesimas, sprintf('\nCPUTime:\t%g sec', t2
       ));
83 pranesimas = strcat( pranesimas, sprintf('\nMin Euc. dist:\t%g',
       Euc_dist_min));
84 pranesimas = strcat( pranesimas, sprintf('\nAtp. indexas:\t%g',
       Recognized_index));
85 msgbox(pranesimas);
```

FIŠERIO VEIDŲ ALGORITMAS

run.m

```
1  clear all
2  close all
3  clc
4
5  %% Nustatomas apmokymo katalogas ir testavimo byla
6  TrainDatabasePath = uigetdir('TrainDatabase', 'Parinkite apmokymo
       kataloga');
7  [ TestImageName TestImagePath ] = uigetfile({'*.*'; '*.jpg';'*.
       jpeg';'*.pgm';'*.png'; '*.bmp'}, 'Parinkite testavimo byla', '
       TestDatabase');
8  TestImage = strcat(TestImagePath, '\', TestImageName);
9  extension = '*.pgm';
10 Class_population = 9; % Nustatomas kiekvienos klases populiacija
11 Class_number = ( size(dir(TrainDatabasePath),1) - 2 ) /
       Class_population; % Nustatomas klasiu skaicius
12
13 %% Algoritmo apmokymas
14 [MeanFace V_PCA V_Fisher ProjectedImages_Fisher TrainFiles
       imgSIZE] = train(TrainDatabasePath, extension, Class_number,
       Class_population);
15
16 %% Algoritmo testavimas
17 OutputName = Recognition( TestImage, MeanFace, V_PCA, V_Fisher,
       ProjectedImages_Fisher);
18 OutputName = TrainFiles(OutputName).name;
19
20 RecognizedImage = imread(strcat(TrainDatabasePath,'\', OutputName
       ));
21
22 figure('Name', 'Rezultatai')
23 subplot(1,2,1); imshow(TestImage);        title('Test Image');
24 subplot(1,2,2); imshow(RecognizedImage);   title('Recognized Image
       ');
25
26 figure('Name', 'MeanFace')
27 imshow(reshape( cast(MeanFace, 'uint8'), circshift(imgSIZE', 3)')
       ')
28
29 disp(OutputName);
30 disp(TestImageName);
```

train.m

```
1  function [m_database V_PCA V_Fisher ProjectedImages_Fisher
       TrainFiles imgSIZE] = train(TrainDatabasePath, extension,
       Class_number, Class_population)
2
3  %% sukuriamas apsimokymo katalogu sarasas
4  TrainFiles = dir( strcat(TrainDatabasePath, '/', extension) );
5  Train_Number = size(TrainFiles, 1);
6
7
8  %% is 1D paveikslu vektoriu sudaroma  2D matrica
9  T = [];
10 for i = 1 : Train_Number
11     PathToImage = strcat(TrainDatabasePath, '/', TrainFiles(i).
           name);
12     img = imread(PathToImage);
13   if size(size(img), 2) > 2
14       img = rgb2gray(img);
15   end
16     [irow icol] = size(img);
17     temp = reshape(img',irow*icol,1);   % 2D paveikslas
           perdaromas i 1D vektoriu
18     T = [T temp];
19 end
20 imgSIZE = [irow icol];
21 T = double(T);
22
23 P = Class_population * Class_number; % bendras paveikslu skaicius
24
25 %% apskaiciuojamas suvidurkintas paveikslas
26 m_database = mean(T,2);
27
28 %% Apskaiciuojamas kiekvieno paveikslo nuokrypis nuo suvidurkinto
       paveikslo
29 A = T - repmat(m_database,1,P);
30
31 %% Apskaiciuojama kovariacine matrica
32 L = A'*A; % L yra kovariancines matricos C=A*A' pakaitalas
33 [V D] = eig(L); % D stulpeliai yra L (C) eigenvertes
34
35 %% surusiuojamos ir pasalinamos maziausios eigenvertes
36 L_eig_vec = [];
37 for i = P-Class_number : -1 : 1
38     L_eig_vec = [L_eig_vec V(:,i)];
39 end
40
41 %% apskaiciuojami kovariacines matricos eigenvektoriai
42 V_PCA = A * L_eig_vec; % A: centered image vectors
43
44 %% paveikslai projektuojami i eigenerdve
```

```
45 ProjectedImages_PCA = [];
46 for i = 1 : P
47     temp = V_PCA'*A(:,i);
48     ProjectedImages_PCA = [ProjectedImages_PCA temp];
49 end
50
51 %% apskaiciuojamas kiekvienos klases vidurkis eigenerdveje
52 m_PCA = mean(ProjectedImages_PCA,2); % bendras eigenerdves
       vidurkis
53 m = zeros(P-Class_number,Class_number);
54 Sw = zeros(P-Class_number,P-Class_number); % Inicializuojama
       klases sklaidos matrica
55 Sb = zeros(P-Class_number,P-Class_number); % Inicializuojama
       tarpklasines sklaidos matrica
56
57 for i = 1 : Class_number
58     m(:,i) = mean( ( ProjectedImages_PCA(:,((i-1)*
           Class_population+1):i*Class_population) ), 2 )';
59
60     S = zeros(P-Class_number,P-Class_number);
61     for j = ( (i-1)*Class_population+1 ) : ( i*Class_population )
62         S = S + (ProjectedImages_PCA(:,j)-m(:,i))*(
               ProjectedImages_PCA(:,j)-m(:,i))';
63     end
64
65     Sw = Sw + S; % Klases sklaidos matrica
66     Sb = Sb + (m(:,i)-m_PCA) * (m(:,i)-m_PCA)'; % Tarpklasines
           sklaidos matrica
67 end
68
69 %% apskaiciuojamas fisherio diskriminantas
70 %% maksimizuojama tarpklasine sklaida ir minimizuojama sklaida
       klases viduje
71 [J_eig_vec, J_eig_val] = eig(Sb,Sw);
72 J_eig_vec = fliplr(J_eig_vec);
73
74 %% Pasalinamos eigenvertes lygios 0 ir surusiuojama mazejimo
       tvarka
75 for i = Class_number-1 : -1 : 1
76     V_Fisher(:,i) = J_eig_vec(:,i);
77 end
78
79 %% Paveikslai projektojami i tiesine Fisherio erdve
80 for i = 1 : Class_number*Class_population
81     ProjectedImages_Fisher(:,i) = V_Fisher' * ProjectedImages_PCA
           (:,i);
82 end
```

recognition.m

```
1  function OutputName = Recognition(TestImage, m_database, V_PCA,
       V_Fisher, ProjectedImages_Fisher)
2
3  Train_Number = size(ProjectedImages_Fisher,2);
4
5  %% is testuojamo paveikslo isskiriami FLD bruozai
6  InputImage = imread(TestImage);
7  if size( size(InputImage), 2) > 2
8    InputImage = rgb2gray(InputImage);
9  end
10
11 [irow icol] = size(InputImage);
12 InImage = reshape(InputImage',irow*icol,1);
13 Difference = double(InImage)-m_database; % Apskaiciuojamas
       skirtumas nuo suvidurkinto veido
14 ProjectedTestImage = V_Fisher' * V_PCA' * Difference; %
       testuojamo paveikslo bruozu vektorius
15
16 %% Apskaiciuojamas euklido atstumas nuo duomenu bazes
17 Euc_dist = [];
18 for i = 1 : Train_Number
19      q = ProjectedImages_Fisher(:,i);
20      temp = ( norm( ProjectedTestImage - q ) )^2;
21      Euc_dist = [Euc_dist temp];
22 end
23
24 %% surandamas maziausias atstumas
25 [Euc_dist_min , Recognized_index] = min(Euc_dist);
26
27 %% grazinamas atsakymas
28 OutputName = Recognized_index;
```

2D–DCT+SOM ALGORITMAS

FaceRec.m

```
clear all;
clc;

%% Nuskaitomos apmokymo ir testavimo bylos
TrainPath  = 'train/ATT/'; % apmokymo katalogas
TestPath   = 'test/ATT/'; % testavimo katalogas
Extension  = '*.pgm'; % bylu pletinys
TrainFiles = dir( strcat(TrainPath, Extension) );
TestFiles  = dir( strcat(TestPath, Extension)  );
ResCoe     = 0.2; % paveikslu raiskos santykis

%% Algoritmo apmokymas
for i = 1 : size(TrainFiles, 1)
  TRAIN.img_rgb = imread( strcat(TrainPath, TrainFiles(i).name) )
      ; % Nuskaitomas RGB paveikslas
   if size(size(TRAIN.img_rgb),2) > 2
     TRAIN.img_grey(:,:,i) = rgb2gray(TRAIN.img_rgb); % Paveikslas
         paverciamas i nespalvota
   else
     TRAIN.img_grey(:,:,i) = (TRAIN.img_rgb);
   end
   TRAIN.img(:,:,i) = imresize(TRAIN.img_grey(:,:,i), ResCoe, '
      nearest'); % Paveikslas sumazinamas pagal artimiausio
      kaimyno salyga
   TRAIN.DctCoef(:,:,i) = dct2(TRAIN.img(:,:,i)); % Ivykdoma 2D-
      DCT

   % imshow(DctCoef)
   % imshow(cast(DctCoef, 'uint8'))
   %% isskiriami mus dominantys koeficientai
   TRAIN.DctCoefTemp(:,:,i) = TRAIN.DctCoef(:,:,i);
   TRAIN.DctCoefTemp( :, (end/2)+1:end, i) = 0;
   TRAIN.DctCoefTemp( (end/2)+1:end, :, i) = 0;
   TRAIN.DctCoef_reduced(:,:,i)  = TRAIN.DctCoefTemp(:,:,i); %
      Iskiriamas tik virsutinis kairysis koeficientu kampas, nes
      cia pagrindines reiksmes
   TRAIN.ReconstructedIMG(:,:,i) = idct2(TRAIN.DctCoef_reduced
      (:,:,i)); % atliekama 2d-iDCT (atvirkstine 2D-DCT)

   %% 2D paveikslas transformuojamas i 1D vektoriu
   TRAIN.temp = TRAIN.ReconstructedIMG(:,:,i);
   TRAIN.ReconstructedVEC(:, i) = TRAIN.temp(:);
end

%% Algoritmo testavimas
for j = 1 : size(TestFiles, 1)
```

```
39
40    TEST.img_rgb      = imread( strcat(TestPath, TestFiles(j).name)
          ); % Nuskaitomas RGB paveikslas
41    if size(size(TEST.img_rgb),2) > 2
42      TEST.img_grey(:,:,j) = rgb2gray(TEST.img_rgb); % Paveikslas
            paverciamas i nespalvota
43    else
44      TEST.img_grey(:,:,i) = (TEST.img_rgb);
45    end
46    TEST.img(:,:,j) = imresize(TEST.img_grey(:,:,j), ResCoe,   '
          nearest'); % Paveikslas sumazinamas pagal artimiausio
          kaimyno salyga
47    TEST.DctCoef(:,:,j)   = dct2(TEST.img(:,:,j)); % Ivykdoma 2D DCT
48
49    % imshow(DctCoef)
50    % imshow(cast(DctCoef, 'uint8'))
51
52    %% isskiriami mus dominantys koeficientai
53    TEST.DctCoefTemp( :,                :,                j) = TEST.
          DctCoef(:,:,j);
54    TEST.DctCoefTemp( :,                (end/2)+1:end,    j) = 0;
55    TEST.DctCoefTemp( (end/2)+1:end,    :,                j) = 0;
56    TEST.DctCoef_reduced(:,:,j)   = TEST.DctCoefTemp(:,:,j); %
          Iskiriamas tik virsutinis kairysis koeficientu kampas, nes
          cia pagrindines reiksmes
57    TEST.ReconstructedIMG(:,:,j) = idct2(TEST.DctCoef_reduced(:,:,j
          )); % atliekama 2d-iDCT (atvirkstine 2D-DCT)
58
59    %% 2D paveikslas transformuojamas i 1D vektoriu
60    TEST.temp = TEST.ReconstructedIMG(:,:,j);
61    TEST.ReconstructedVEC(:, j) = TEST.temp(:);
62  end
63
64  %% Sutvarkoma darbo aplinka
65  clear i j ResCoe;
66
67  %% sudaromi apmokymo ir testavimo kintamieji SOM tinklui
68  P = TRAIN.ReconstructedVEC;
69  T = TEST.ReconstructedVEC;
70
71  %% sukuriamas naujas save organizuojantis tinklas
72  net = newsom(P, [40, 1], 'hextop', 'linkdist', 100, 3);
73  %net.trainParam.epochs = 500;
74
75  %% SOM apmokomas
76  net = train(net,P);
77
78  %% SOM testuojamas
79  [Y,Pf,Af,E] = sim(net,P(:,:));
80  [YY,Pf,Af,E] = sim(net,T(:,:));
```

E. Pasirikto algoritmo įgyvendinto *Android* OS pirminis programos tekstas

ANDROIDMANIFEST.XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/
       android"
3    package="com.balinskas.fotointent"
4    android:versionCode="1"
5    android:versionName="1.0" >
6
7    <uses-sdk android:minSdkVersion="10" />
8
9    <!-- Reikalaujami permissionai -->
10   <uses-permission android:name="android.permission.CAMERA" />
11   <uses-feature android:name="android.hardware.camera" />
12   <uses-permission android:name="android.permission.
       WRITE_EXTERNAL_STORAGE" />
13   <uses-permission android:name="android.permission.RECORD_AUDIO"
         />
14   <uses-permission android:name="android.permission.
       ACCESS_FINE_LOCATION" />
15
16   <application android:icon="@drawable/ic_launcher" android:label
       ="@string/app_name" >
17
18     <activity
19       android:name=".FotoIntentActivity"
20       android:label="@string/app_name"
21       android:screenOrientation="portrait">
22       <intent-filter>
23         <action android:name="android.intent.action.MAIN" />
24         <category android:name="android.intent.category.LAUNCHER"
               />
25       </intent-filter>
26     </activity>
27     <activity
28       android:name=".myCameraIntent"
29       android:label="@string/app_name"
30       android:screenOrientation="landscape"
31       android:theme="@android:style/Theme.Black.NoTitleBar.
           Fullscreen">
32       <intent-filter>
33         <action android:name="android.intent.action.MAIN"/>
34         <!-- action android:name=".myCameraIntent"/-->
35         <category android:name="android.intent.category.LAUNCHER"
               />
36       </intent-filter>
37     </activity>
38     <activity
39       android:name=".RealTime"
40       android:label="@string/app_name"
```

```
41          android:screenOrientation="landscape"
42          android:theme="@android:style/Theme.Black.NoTitleBar.
                Fullscreen"
43      >
44          <intent-filter>
45              <action android:name="android.intent.action.MAIN" />
46              <!-- action android:name=".RealTime" -->
47              <category android:name="android.intent.category.LAUNCHER"
                    />
48          </intent-filter>
49      </activity>
50
51  </application>
52
53  </manifest>
```

JNI/ANDROID.MK

```
1  # Copyright (C) 2009 The Android Open Source Project
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License
5  # You may obtain a copy of the License at
6  #
7  #      http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing,
10 # software distributed under the License is distributed on an
11 # "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
12 # either express or implied. See the License for the specific
13 # language governing permissions and limitations under the
14 # License.
15
16 LOCAL_PATH := $(call my-dir)
17
18 include $(CLEAR_VARS)
19
20 include /home/jb/OpenCV-2.3.1/share/OpenCV/OpenCV.mk
21 OPENCV_CAMERA_MODULES:=off
22
23 LOCAL_MODULE    := recognize-jni
24 LOCAL_SRC_FILES := recognize-jni.cpp
25
26 # Kad butu galima loginti is C++ aplinkos
27 LOCAL_LDLIBS  := -lGLESv1_CM -ldl -llog
28
29 include $(BUILD_SHARED_LIBRARY)
```

JNI/APPLICATION.MK

```
1  # Kad butu galima naudoti STD bibliotekas
2  APP_STL := stlport_static
3
4  # OpenCV bibliotekos
5  APP_STL := gnustl_static
6  APP_CPPFLAGS := -frtti -fexceptions
7
8  # Nurodomas procesoriaus tipas
9  # armeabi      -- AVD
10 # armeabi-v7a -- HTC Desire
11
12 # APP_ABI := armeabi
13 APP_ABI := armeabi-v7a
```

JNI/RECOGNIZE-JNI.CPP

```
1  /*
2   * Copyright (C) 2009 The Android Open Source Project
3   *
4   * Licensed under the Apache License, Version 2.0
5   * (the "License"); you may not use this file
6   * except in compliance with the License
7   * You may obtain a copy of the License at
8   *
9   *      http://www.apache.org/licenses/LICENSE-2.0
10  *
11  * Unless required by applicable law or agreed to in writing,
12  * software distributed under the License is distributed on an
13  * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
14  * either express or implied. See the License for the specific
15  * language governing permissions and limitations under the
16  * License.
17  */
18
19 // Pridedamos pagrindines bibliotekos
20 #include <android/log.h>
21 #include <jni.h>
22 #include <stdio.h>
23 #include <string.h>
24 #include <math.h>
25 #include <iostream>
26 #include <fstream>
27 using std::ofstream;
28
29 // Pridedamos OpenCV bibliotekos
30 #include <opencv/cv.h>
31 #include <opencv/cvaux.h>
32 #include <opencv/highgui.h>
33
34 // Apsibreziamos LOG funkcijos
35 #define  LOG_TAG "MY_JNI_TAG"
36 #define  LOGI(...) __android_log_print(ANDROID_LOG_INFO,LOG_TAG,
       __VA_ARGS__)
37 #define  LOGE(...) __android_log_print(ANDROID_LOG_ERROR,LOG_TAG,
       __VA_ARGS__)
38
39
40 // Aprasomi globalus kintamieji
41 int          nTrainFaces       = 0; // apmokymo paveikslu kiekis
42 int          nEigens           = 0; // eigenverciu skaicius
43 IplImage ** faceImgArr         = 0; // paveikslu matrica
44 CvMat     * personNumTruthMat = 0; // ID matrica
45 IplImage  * pAvgTrainImg       = 0; // suvidurkintas veidas
46 IplImage ** eigenVectArr       = 0; // eigenvectors
```

```
47 CvMat      * eigenValMat        = 0; // eigenvalues
48 CvMat      * projectedTrainFaceMat = 0; // apmokymo veidu
      projekcijos
49
50 int      * arrNearest;
51 double   * arrLeastDistSq;
52 float    * arrConfidence;
53
54 double    leastDistSq;    // atstumas
55 int       nearest;        // Artimiausio veido ID
56 float     confidence = 0;// patikimumas
57 ofstream  logFile;        // log failas
58
59 // Aprasomi CHAR tipo kintamieji (nuorodos i failus)
60 char buffer[100];
61 char test_file[100]      = "/mnt/sdcard/MyCameraApp/test.txt";
62 char test_file_RT[100]   = "/mnt/sdcard/MyCameraApp/test_RT.txt";
63 char test_file_ALL[100] = "/mnt/sdcard/MyCameraApp/allFaces/
      testAllFaces.txt";
64 char train_file[100]     = "/mnt/sdcard/MyCameraApp/train.txt";
65 char logPath[100]        = "/mnt/sdcard/MyCameraApp/log.txt";
66 char faceData[100]       = "/mnt/sdcard/MyCameraApp/facedata.xml";
67
68 // Aprasomos funkcijos
69 void learn();
70 void recognize();
71 void recognize_RT();
72 int  recognize_ALL();
73 void PCA()();
74 void storeTrainingData();
75 int  loadTrainingData(CvMat ** pTrainPersonNumMat);
76 int  findNearestNeighbor(float * projectedTestFace, float *
      pConfidence);
77 int  loadFaceImgArray(char * filename);
78 void printUsage();
79
80 /*****************************************/
81 /*        APRASOMOS NATIVE FUNKCIJOS       */
82 /*****************************************/
83
84 // Natvie atpazinimas
85 extern "C"{
86     jobject
87     Java_com_balinskas_fotointent_FotoIntentActivity_jniRecognize
         ( JNIEnv* env,
88     jobject thiz)
89     {
90         jclass cls = env->FindClass("com/balinskas/fotointent/
            JavaCls");
91         if(!cls) {
92             LOGE("initClassHelper: failed to get %s class
                reference", "com/balinskas/fotointent/JavaCls");
```

```
93              return 0;
94          }
95
96          jmethodID constr = env->GetMethodID(cls, "<init>", "()V")
                ;
97          if(!constr) {
98              LOGE("initClassHelper: failed to get %s constructor",
                    "com/balinskas/fotointent/JavaCls");
99              return 0;
100         }
101
102         jobject obj = env->NewObject(cls, constr);
103         if(!obj) {
104             LOGE("initClassHelper: failed to create a %s object",
                    "com/balinskas/fotointent/JavaCls");
105             return 0;
106         }
107
108         LOGI("recognize() function called");
109         logFile.open(logPath);
110         recognize();
111         LOGI("recognize() function finished");
112         logFile.close();
113
114
115         jint jintNearest = nearest;
116         jdouble jdoubleLeastDistSq = leastDistSq;
117         jfloat jfloatConfidence = confidence;
118
119         jfieldID fId = env->GetFieldID(cls , "faceID", "I");
120         env->SetIntField( obj, fId, jintNearest);
121
122         jfieldID fDist = env->GetFieldID(cls , "dist", "D");
123         env->SetDoubleField( obj, fDist, jdoubleLeastDistSq);
124
125         jfieldID fConf = env->GetFieldID(cls , "conf", "F");
126         env->SetFloatField( obj, fConf, jfloatConfidence);
127
128         return( env->NewGlobalRef(obj));
129     }
130 }
131
132 // Native apsimokymas
133 extern "C"{
134     jstring
            Java_com_balinskas_fotointent_FotoIntentActivity_jniLearn(
             JNIEnv* env, jobject thiz )
135     {
136         LOGI("learn() function called");
137         learn();
138         LOGI("learn() function finished");
139         return (env->NewStringUTF("your text here"));
```

```
140     }
141 }
142
143 // Native atpazinimas realiu laiku
144 extern "C"{
145     jobject
146     Java_com_balinskas_fotointent_RealTime_jniRecognizeRT( JNIEnv
        * env,
147     jobject thiz)
148     {
149         jclass cls = env->FindClass("com/balinskas/fotointent/
            JavaCls");
150         if(!cls) {
151             LOGE("initClassHelper: failed to get %s class
                reference", "com/balinskas/fotointent/JavaCls");
152             return 0;
153         }
154
155         jmethodID constr = env->GetMethodID(cls, "<init>", "()V")
            ;
156         if(!constr) {
157             LOGE("initClassHelper: failed to get %s constructor",
                "com/balinskas/fotointent/JavaCls");
158             return 0;
159         }
160
161         jobject obj = env->NewObject(cls, constr);
162         if(!obj) {
163             LOGE("initClassHelper: failed to create a %s object",
                "com/balinskas/fotointent/JavaCls");
164             return 0;
165         }
166
167         LOGI("recognize() function called");
168         logFile.open(logPath);
169         recognize_RT();
170         LOGI("recognize() function finished");
171         logFile.close();
172
173
174         jint    jintNearest = nearest;
175         jdouble jdoubleLeastDistSq = leastDistSq;
176         jfloat  jfloatConf = confidence;
177
178         jfieldID fId = env->GetFieldID(cls , "faceID", "I");
179         env->SetIntField( obj, fId, jintNearest);
180
181         jfieldID fDist = env->GetFieldID(cls , "dist", "D");
182         env->SetDoubleField( obj, fDist, jdoubleLeastDistSq);
183
184         jfieldID fConf = env->GetFieldID(cls , "conf", "F");
185         env->SetFloatField( obj, fConf, jfloatConf);
```

```
186
187         return( env->NewGlobalRef(obj));
188     }
189 }
190
191 // Natvie atpazinimas visu veidu vienu metu
192 extern "C"{
193     jobject
194     Java_com_balinskas_fotointent_FotoIntentActivity_jniRecognizeAll
        ( JNIEnv* env,
195     jobject thiz)
196     {
197         int count = 0;
198         FILE * imgListFile = 0;
199         char imgFilename[512];
200         imgListFile = fopen(test_file_ALL, "r");
201
202         // Suskaiciuojami veidai
203         while( fgets(imgFilename, 512, imgListFile) ) ++count;
204
205         // apsibreziami masyvai, kuriuose bus saugomi atsakymai
206         arrNearest      = new int[count];
207         arrLeastDistSq  = new double[count];
208         arrConfidence   = new float[count];
209
210         // suranadama JAVA klase atsakymams saugoti
211         jclass cls = env->FindClass("com/balinskas/fotointent/
            JavaCls");
212         if(!cls) {
213             LOGE("initClassHelper: failed to get %s class
                reference", "com/balinskas/fotointent/JavaCls");
214             return 0;
215         }
216
217         // surandami metodai
218         jmethodID constr = env->GetMethodID(cls, "<init>", "()V")
            ;
219         if(!constr) {
220             LOGE("initClassHelper: failed to get %s constructor",
                "com/balinskas/fotointent/JavaCls");
221             return 0;
222         }
223
224         // sukuriamas naujas objektas atsakymams
225         jobject obj = env->NewObject(cls, constr);
226         if(!obj) {
227             LOGE("initClassHelper: failed to create a %s object",
                "com/balinskas/fotointent/JavaCls");
228             return 0;
229         }
230
231         // pradedamas atpazinimas
```

```
232         LOGI("recognize()␣function␣called");
233         logFile.open(logPath);
234         int size = recognize_ALL();
235         LOGI("recognize()␣function␣finished");
236         logFile.close();
237 
238         /* naujame objekte suzymimi atsakymai */
239         // veido ID
240         jfieldID fArId = env->GetFieldID(cls , "arrFaceID", "[I")
                ;
241         jintArray jArrID;
242         jArrID = (jintArray)env->GetObjectField(obj, fArId);
243         jint *idBody = env->GetIntArrayElements(jArrID, 0);
244         int arrNearestSize = size;
245         for (int i = 0; i<arrNearestSize; i++) {
246             idBody[i] = arrNearest[i];
247         }
248         env->ReleaseIntArrayElements(jArrID, idBody, 0);
249 
250         // atstumas
251         jfieldID fArDist = env->GetFieldID(cls , "arrDist", "[D")
                ;
252         jdoubleArray jArrDist;
253         jArrDist = (jdoubleArray)env->GetObjectField(obj, fArDist
                );
254         jdouble *distBody = env->GetDoubleArrayElements(jArrDist,
                0);
255 //      int arrDistSize = sizeof(arrLeastDistSq) / sizeof(double)
        ;
256         int arrDistSize = size;
257         for (int i = 0; i<arrDistSize; i++) {
258             distBody[i] = arrLeastDistSq[i];
259         }
260         env->ReleaseDoubleArrayElements(jArrDist, distBody, 0);
261 
262         // patikimumas
263         jfieldID fArConf = env->GetFieldID(cls , "arrConf", "[F")
                ;
264         jfloatArray jArrConf;
265         jArrConf = (jfloatArray)env->GetObjectField(obj, fArConf)
                ;
266         jfloat *confBody = env->GetFloatArrayElements(jArrConf,
                0);
267 //      int arrConfSize = sizeof(arrConfidence) / sizeof(float);
268         int arrConfSize = size;
269         for (int i = 0; i<arrConfSize; i++) {
270             confBody[i] = arrConfidence[i];
271         }
272         env->ReleaseFloatArrayElements(jArrConf, confBody, 0);
273 
274         return( env->NewGlobalRef(obj));
275     }
```

```
276 }
277
278
279 /***************************************/
280 /*          APRASOMOS C++ FUNKCIJOS        */
281 /***************************************/
282
283 // apsimokymo funkcija
284 void learn()
285 {
286     printf("\tlearn() was called\n");
287
288     int i;
289
290     // uzkraunami apsimokymo duomenys
291     nTrainFaces = loadFaceImgArray(train_file);
292
293     if( nTrainFaces < 2 )
294     {
295         sprintf(stderr,
296         "Apsimokymui reikalingi 2 ar daugiau veidu\n"
297         "Siuo metu ju yra tik %d\n", nTrainFaces);
298         LOGE(stderr);
299         return;
300     }
301
302     // Apmokymo paveikslams atliekama
303     // principiniu komponenciu analize (PCA)
304     PCA();
305
306     // Apmokymo paveikslai projektuojami i PCA erdve
307     projectedTrainFaceMat = cvCreateMat(nTrainFaces, nEigens,
            CV_32FC1);
308     for(i=0; i<nTrainFaces; i++)
309     {
310         cvEigenDecomposite(
311         faceImgArr[i],
312         nEigens,
313         eigenVectArr,
314         0, 0,
315         pAvgTrainImg,
316         projectedTrainFaceMat->data.fl + i*nEigens);
317     }
318     // veidu duomenys issaugomi xml byloje
319     storeTrainingData();
320 }
321
322 // pagal sarasa nuskaitomi paveikslai
323 int loadFaceImgArray(char * filename)
324 {
325     FILE * imgListFile = 0;
326     char imgFilename[512];
```

```
327     int iFace, nFaces=0;
328
329     // atidaromas sarasas
330     imgListFile = fopen(filename, "r");
331
332     // suskaiciuojamas irasu kiekis
333     while( fgets(imgFilename, 512, imgListFile) ) ++nFaces;
334     rewind(imgListFile);
335
336     // Rezervuojama atmintis veidu ir ID masyvams
337     faceImgArr = (IplImage **)cvAlloc( nFaces*sizeof(IplImage *)
            );
338     personNumTruthMat = cvCreateMat( 1, nFaces, CV_32SC1 );
339
340     // Veidu paveikslai issaugomi masyve
341     for(iFace=0; iFace<nFaces; iFace++)
342     {
343         // Nuskaitomas asmens ID ir kelias iki paveikslo
344         fscanf(imgListFile,
345         "%d %s", personNumTruthMat->data.i+iFace, imgFilename);
346
347         // Nuskaitomas paveikslas
348         faceImgArr[iFace] = cvLoadImage(imgFilename,
                CV_LOAD_IMAGE_GRAYSCALE);
349
350         // Ivykdomas histogramos islyginimas
351         // http://dasl.mem.drexel.edu/~noahKuntz/openCVTut5.html
352         cvEqualizeHist( faceImgArr[iFace], faceImgArr[iFace] );
353
354         // KONTRASTAS
355         // http://www.aishack.in/2010/02/filtering-images/
356          cvScale(faceImgArr[iFace], faceImgArr[iFace], 1.3);
357     }
358
359     fclose(imgListFile);
360     return nFaces;
361 }
362
363 // Apskaiciuojami PCA
364 void PCA()
365 {
366     int i;
367     CvTermCriteria calcLimit;
368     CvSize faceImgSize;
369
370     // nustatomas naudotinas tikriniu verciu kiekis
371     nEigens = nTrainFaces-1;
372
373     // rezervuojama atminis eigenvektoriu paveikslams
374     faceImgSize.width = faceImgArr[0]->width;
375     faceImgSize.height = faceImgArr[0]->height;
```

```
376     eigenVectArr = (IplImage**)cvAlloc(sizeof(IplImage*) *
            nEigens);
377     for(i=0; i<nEigens; i++)
378     eigenVectArr[i] = cvCreateImage(faceImgSize, IPL_DEPTH_32F,
            1);
379
380     // sukuriamas eigenverciu masyvas
381     eigenValMat = cvCreateMat( 1, nEigens, CV_32FC1 );
382
383     // sukuriamas paveikslas suvidurkintam veidui saugoti
384     pAvgTrainImg = cvCreateImage(faceImgSize, IPL_DEPTH_32F, 1);
385
386     // Nustatomas PCA nutraukimo kriterijus
387     calcLimit = cvTermCriteria( CV_TERMCRIT_ITER, nEigens, 1);
388
389     // Apskaiciuojamas suvidurkintas paveikslas, eigenvertes ir
            eigenvektoriai
390     cvCalcEigenObjects( nTrainFaces,
391                         (void*)faceImgArr,
392                         (void*)eigenVectArr,
393                         CV_EIGOBJ_NO_CALLBACK,
394                         0,
395                         0,
396                         &calcLimit,
397                         pAvgTrainImg,
398                         eigenValMat->data.fl);
399 }
400
401 // issaugomi apsimokymo duomenys
402 void storeTrainingData()
403 {
404     CvFileStorage * fileStorage;
405     int i;
406
407     // atidaroma byla, kurioje saugomi duomenys
408     fileStorage = cvOpenFileStorage( faceData, 0,
            CV_STORAGE_WRITE );
409
410     // issaugomi visi duomenys
411     cvWriteInt( fileStorage, "nEigens", nEigens );
412     cvWriteInt( fileStorage, "nTrainFaces", nTrainFaces );
413     cvWrite(fileStorage, "trainPersonNumMat", personNumTruthMat,
            cvAttrList(0,0));
414     cvWrite(fileStorage, "eigenValMat", eigenValMat, cvAttrList
            (0,0));
415     cvWrite(fileStorage, "projectedTrainFaceMat",
            projectedTrainFaceMat, cvAttrList(0,0));
416     cvWrite(fileStorage, "avgTrainImg", pAvgTrainImg, cvAttrList
            (0,0));
417     for(i=0; i<nEigens; i++)
418     {
419         char varname[200];
```

```
420             sprintf( varname, "eigenVect_%d", i );
421             cvWrite(fileStorage, varname, eigenVectArr[i], cvAttrList
                    (0,0));
422         }
423
424         // uzdaroma byla
425         cvReleaseFileStorage( &fileStorage );
426     }
427
428     // uzkraunami apmokymo duomenys
429     int loadTrainingData(CvMat ** pTrainPersonNumMat)
430     {
431         CvFileStorage * fileStorage;
432         int i;
433
434         // atidaroma byla
435         fileStorage = cvOpenFileStorage( faceData, 0, CV_STORAGE_READ
                );
436         if( !fileStorage )
437         {
438             LOGE("Can't open facedata.xml\n");
439             return 0;
440         }
441
442         // nuskaitomi duomenys
443         nEigens = cvReadIntByName(fileStorage, 0, "nEigens", 0);
444         nTrainFaces = cvReadIntByName(fileStorage, 0, "nTrainFaces",
                0);
445         *pTrainPersonNumMat = (CvMat *)cvReadByName(fileStorage, 0, "
                trainPersonNumMat", 0);
446         eigenValMat = (CvMat *)cvReadByName(fileStorage, 0, "
                eigenValMat", 0);
447         projectedTrainFaceMat =
448         (CvMat *)cvReadByName(fileStorage, 0, "projectedTrainFaceMat"
                , 0);
449         pAvgTrainImg = (IplImage *)cvReadByName(fileStorage, 0, "
                avgTrainImg", 0);
450         eigenVectArr = (IplImage **)cvAlloc(nTrainFaces*sizeof(
                IplImage *));
451         for(i=0; i<nEigens; i++)
452         {
453             char varname[200];
454             sprintf( varname, "eigenVect_%d", i );
455             eigenVectArr[i] = (IplImage *)cvReadByName(fileStorage,
                    0, varname, 0);
456         }
457
458         // uzdaroma byla
459         cvReleaseFileStorage( &fileStorage );
460
461         return 1;
462     }
```

```
463
464 // surandamas panasiausias veidas
465 int findNearestNeighbor(float * projectedTestFace, float *
        pConfidence)
466 {
467     leastDistSq = DBL_MAX;
468     int i, iTrain, iNearest = 0;
469
470     for(iTrain=0; iTrain<nTrainFaces; iTrain++)
471     {
472         double distSq=0;
473
474         for(i=0; i<(nEigens); i++)
475         {
476             float d_i =
477                     projectedTestFace[i] -
478                     projectedTrainFaceMat->data.fl[iTrain *
                        nEigens + i];
479
480             distSq += d_i*d_i / eigenValMat->data.fl[i];  //
                Mahalanobis
481             // distSq += d_i*d_i; // Euclidean
482         }
483         if (distSq < leastDistSq) {
484             leastDistSq = distSq;
485             iNearest = iTrain;
486         }
487     }
488     // Pagal atstuma apskaiciuojamas patikimumas
489 //  *pConfidence = 1.0f - sqrt(leastDistSq / (float)255*255*255 )
        ; // Euclidean
490     *pConfidence = 1.0f - leastDistSq; // Mahalanobis
491
492     return iNearest;
493 }
494
495 // atpazinimo funkcija
496 void recognize()
497 {
498     int i, nTestFaces = 0; // testuojamu paveikslu kiekis
499     CvMat * trainPersonNumMat = 0; // asmens numeris apmokymo
        metu
500     float * projectedTestFace = 0;
501
502     // nuskaitomas testavimo paveikslas
503     nTestFaces = loadFaceImgArray(test_file);
504
505     // Uzkraunama issaugota apmokymo informacija
506     if( !loadTrainingData( &trainPersonNumMat ) ) return;
507
508     // Testuojamas paveikslas projektuojamas i PCA erdve
```

```
509     projectedTestFace = (float *)cvAlloc( nEigens*sizeof(float) )
            ;
510     for(i=0; i<nTestFaces; i++)
511     {
512         int iNearest, truth;
513         cvEigenDecomposite(
514                 faceImgArr[i],
515                 nEigens,
516                 eigenVectArr,
517                 0, 0,
518                 pAvgTrainImg,
519                 projectedTestFace);
520
521         // Surandamas panasiausias paveikslas
522         iNearest = findNearestNeighbor(projectedTestFace, &
                confidence);
523         truth = personNumTruthMat->data.i[i];
524         nearest = trainPersonNumMat->data.i[iNearest];
525
526         // Isvedama informacija
527         sprintf(buffer, "\tnearest = %d, Truth = %d\n", nearest,
                truth);
528         LOGI(buffer);
529         logFile << buffer;
530     }
531 }
532
533 // atpazinimo realiu laiku funkcija
534 void recognize_RT()
535 {
536     int i, nTestFaces = 0; // testuojamu paveikslu kiekis
537     CvMat * trainPersonNumMat = 0; // asmens numeris apmokymo
            metu
538     float * projectedTestFace = 0;
539
540     // nuskaitomas testavimo paveikslas
541     nTestFaces = loadFaceImgArray(test_file_RT);
542
543     // Uzkraunama issaugota apmokymo informacija
544     if( !loadTrainingData( &trainPersonNumMat ) ) return;
545
546     // Testuojamas paveikslas projektuojamas i PCA erdve
547     projectedTestFace = (float *)cvAlloc( nEigens*sizeof(float) )
            ;
548
549     for(i=0; i<nTestFaces; i++)
550     {
551         int iNearest, truth;
552         cvEigenDecomposite(
553                 faceImgArr[i],
554                 nEigens,
555                 eigenVectArr,
```

```
556                     0, 0,
557                     pAvgTrainImg,
558                     projectedTestFace);
559
560             // Surandamas panasiausias paveikslas
561             iNearest = findNearestNeighbor(projectedTestFace, &
                    confidence);
562             truth = personNumTruthMat->data.i[i];
563             nearest = trainPersonNumMat->data.i[iNearest];
564
565             // Isvedama informacija
566             logFile << buffer;
567         }
568     }
569
570     // visu veidu atpazinimo funkcija
571     int recognize_ALL()
572     {
573         int i, nTestFaces = 0; // testuojamu paveikslu kiekis
574         CvMat * trainPersonNumMat = 0; // asmens numeris apmokymo
                metu
575         float * projectedTestFace = 0;
576
577         // nuskaitomas testavimo paveikslas
578         nTestFaces = loadFaceImgArray(test_file_ALL);
579
580         // isvedama informacija apie nuskaitytu veidu kieki
581         sprintf(buffer, "\n\n%d test faces loaded\n", nTestFaces);
582         LOGI(buffer);
583
584         // Uzkraunama issaugota apmokymo informacija
585         if( !loadTrainingData( &trainPersonNumMat ) ) return 0;
586
587         // Testuojamas paveikslas projektuojamas i PCA erdve
588         projectedTestFace = (float *)cvAlloc( nEigens*sizeof(float) )
                ;
589         int size = 0;
590         for(i=0; i<nTestFaces; i++)
591         {
592             int iNearest, truth;
593             cvEigenDecomposite(
594                     faceImgArr[i],
595                     nEigens,
596                     eigenVectArr,
597                     0, 0,
598                     pAvgTrainImg,
599                     projectedTestFace);
600
601             // Surandamas panasiausias paveikslas
602             iNearest = findNearestNeighbor(projectedTestFace, &
                    confidence);
603             truth = personNumTruthMat->data.i[i];
```

```
604             nearest = trainPersonNumMat->data.i[iNearest];
605
606             // Isvedama informacija
607             sprintf(buffer, "\tnearest = %d, Truth = %d, Confidence =
                    %1.3f, Dist = %1.3f\n", nearest, truth, confidence,
                    leastDistSq);
608             LOGI(buffer);
609             logFile << buffer;
610
611             // informacija issaugoma masyvuose
612             arrNearest[i] = nearest;
613             arrLeastDistSq[i] = leastDistSq;
614             arrConfidence[i] = confidence;
615
616             size++;
617         }
618
619         return size;
620     }
```

RES/LAYOUT/FOTOINTENTLAYOUT.XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <AbsoluteLayout
3      android:layout_width="fill_parent"
4      android:layout_height="fill_parent"
5      xmlns:android="http://schemas.android.com/apk/res/android"
6      >
7
8      <FrameLayout
9          android:id="@+id/camera_preview"
10         android:layout_width="fill_parent"
11         android:layout_height="fill_parent"
12         android:background="@drawable/face" >
13
14     </FrameLayout>
15
16     <Button
17         android:layout_width="70dp"
18         android:layout_height="wrap_content"
19         android:layout_x="450dp"
20         android:layout_y="150dp"
21         android:id="@+id/bFoto"
22         android:text="Foto" />
23
24 </AbsoluteLayout>
```

RES/LAYOUTS/MAIN.XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/
       android"
3    android:layout_width="fill_parent"
4    android:layout_height="fill_parent"
5    android:orientation="vertical" >
6
7    <LinearLayout
8      android:id="@+id/linearLayout1"
9      android:layout_width="match_parent"
10     android:layout_height="wrap_content"
11     android:gravity="center_horizontal"
12     android:orientation="vertical" >
13
14     <LinearLayout
15       android:id="@+id/linearLayout2"
16       android:layout_width="match_parent"
17       android:layout_height="wrap_content"
18       android:gravity="center_horizontal"
19       android:orientation="horizontal" >
20
21       <Button
22         android:id="@+id/buttonFoto"
23         android:layout_width="wrap_content"
24         android:layout_height="wrap_content"
25         android:text="@string/foto" />
26
27       <Button
28         android:id="@+id/buttonDetect"
29         android:layout_width="wrap_content"
30         android:layout_height="wrap_content"
31         android:text="@string/detect"
32         android:textColor="#A9A9A9" />
33
34       <Button
35         android:id="@+id/buttonExtract"
36         android:layout_width="wrap_content"
37         android:layout_height="wrap_content"
38         android:text="@string/extract"
39         android:textColor="#A9A9A9" />
40     </LinearLayout>
41
42     <ImageView
43       android:id="@+id/test_image"
44       android:layout_width="wrap_content"
45       android:layout_height="wrap_content"
46       android:maxHeight="25mm"
47       android:contentDescription="@string/foto"
```

```
48          android:src="@drawable/face" />
49
50      <Button
51          android:id="@+id/buttonSave"
52          android:layout_width="wrap_content"
53          android:layout_height="wrap_content"
54          android:text="@string/save"
55          android:visibility="invisible" />
56    </LinearLayout>
57
58    <TextView
59        android:id="@+id/myTxtView"
60        android:layout_width="fill_parent"
61        android:layout_height="wrap_content"
62        android:layout_gravity="clip_horizontal"
63        android:gravity="center_horizontal"
64        android:textAppearance="?android:attr/textAppearanceSmall"
65        android:textColor="@android:color/
              secondary_text_dark_nodisable" />
66
67  </LinearLayout>
```

RES/LAYOUTS/MENU.XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android">
3
4    <item android:id="@+id/menu_recognize"
5        android:title="Atpazinti" />
6
7    <item android:id="@+id/menu_recognize_all"
8        android:title="Atpazinti visus" />
9
10   <item android:id="@+id/menu_genlist"
11       android:title="Generuoti sarasa" />
12
13   <item android:id="@+id/menu_learn"
14       android:title="Apsimokinti" />
15
16   <item android:id="@+id/menu_RealTime"
17       android:title="RealTime" />
18
19  </menu>
```

RES/VALUES/STRINGS.XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <string name="hello">Veidu aptikimas ir isskyrimas</string>
5      <string name="app_name">Veidu atpazinimas tikriniu veidu
           metodu</string>
6      <string name="foto">Nuotrauka</string>
7      <string name="detect">Aptikti</string>
8      <string name="extract">Isskirti</string>
9      <string name="save">Issaugoti i DB</string>
10
11 </resources>
```

SRC/COM/BALINSKAS/FOTOINTENT/FOTOINTENTACTIVITY.JAVA

```
1  package com.balinskas.fotointent;
2
3  // importuojamos reikalings bibliotekos
4  import java.awt.font.NumericShaper;
5  import java.io.BufferedWriter;
6  import java.io.File;
7  import java.io.FileOutputStream;
8  import java.io.FileWriter;
9  import java.io.IOException;
10 import java.io.OutputStream;
11 import java.util.ArrayList;
12 import java.util.List;
13 import android.app.Activity;
14 import android.app.AlertDialog;
15 import android.content.DialogInterface;
16 import android.content.Intent;
17 import android.graphics.Bitmap;
18 import android.graphics.BitmapFactory;
19 import android.graphics.Canvas;
20 import android.graphics.Color;
21 import android.graphics.Matrix;
22 import android.graphics.Paint;
23 import android.graphics.PointF;
24 import android.media.FaceDetector;
25 import android.net.Uri;
26 import android.os.Bundle;
27 import android.os.Environment;
28 import android.provider.MediaStore;
29 import android.text.method.ScrollingMovementMethod;
30 import android.util.Log;
31 import android.view.Menu;
32 import android.view.MenuInflater;
33 import android.view.MenuItem;
34 import android.view.View;
35 import android.view.View.OnClickListener;
36 import android.widget.Button;
37 import android.widget.EditText;
38 import android.widget.ImageView;
39 import android.widget.TextView;
40 import android.widget.Toast;
41
42
43 // Aprasoma programa
44 public class FotoIntentActivity extends Activity {
45
46   // deklaruojami globalus kintamieji
47   final  int FINAL_FACE_HEIGHT_PX = 90;
48   final  int FINAL_FACE_WIDTH_PX  = 60;
```

```
49    public int ratio = 2;
50
51    public static String listName = "train.txt";
52    public static String appDirName = "MyCameraApp";
53    public static String imageName = "LastImage";
54
55    private static final String TAG = "FotoIntent";
56    private static final int CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE =
          100;
57
58    public Bitmap faceExtracted;
59    Bitmap bMap;
60    private Uri fileUri;
61
62    ImageView image;
63
64    Button myButtonFoto;
65    Button myButtonDetect;
66    Button myButtonExtract;
67    Button myButtonSave;
68
69    // NATIVE funkcijos
70    public native String  stringFromJNI();
71    public native String  jniLearn();
72    public native JavaCls jniRecognize();
73    public native JavaCls jniRecognizeAll();
74
75    // Mano klases
76    myFiles FS = new myFiles();
77
78    // uzkraunamos native funkcijos
79    static {
80      System.loadLibrary("recognize-jni");
81    }
82
83    /** Called when the activity is first created. */
84    @Override
85    public void onCreate(Bundle savedInstanceState) {
86      super.onCreate(savedInstanceState);
87      setContentView(R.layout.main);
88      Log.v(TAG, "Layout was set");
89
90
91      //-------------------------------------------------------//
92      //-- Pagal nuorodoas surandami vis grafiniai elementai --//
93      //-------------------------------------------------------//
94      myButtonFoto   = (Button)findViewById(R.id.buttonFoto);
95      myButtonDetect  = (Button)findViewById(R.id.buttonDetect);
96      myButtonExtract = (Button)findViewById(R.id.buttonExtract);
97      myButtonSave   = (Button)findViewById(R.id.buttonSave);
98      image = (ImageView) findViewById(R.id.test_image);
99
```

```
100       // mygtuku veiksmai
101       photoChooserButton();
102     }
103
104     // sukuriamas meniu
105     @Override
106     public boolean onCreateOptionsMenu(Menu menu) {
107       MenuInflater menuInflater = getMenuInflater();
108       menuInflater.inflate(R.layout.menu, menu);
109       return true;
110     }
111
112     // aprasomi veiksmai atitinkantis kiekviena meniu irasa
113     public boolean onOptionsItemSelected(MenuItem item)
114     {
115       switch (item.getItemId())
116       {
117       case R.id.menu_recognize:
118         Toast.makeText(this, "Meginama atpazinti...", Toast.
              LENGTH_SHORT).show();
119
120         Log.v(TAG, "Begining");
121         long recStart = System.currentTimeMillis(); // Matuojama
              atpazinimo trukme: pradzios laikas
122         JavaCls faceResult = jniRecognize();
123         long recTime = System.currentTimeMillis() - recStart; //
              Matuojama atpazinimo trukme: trukme
124         Log.v("time", "Atpzinimas uztruko " + recTime + " ms");
125
126         int    faceID   = faceResult.faceID;
127         double  faceDist = faceResult.dist;
128         float faceConf = faceResult.conf;
129
130         Log.v(TAG, "ID: "+faceID+" Dist: "+faceDist+" Conf: "+
              faceConf);
131         faceResult=null;
132
133         // Gaunamas vardas pagal ID
134         String vardas = null;
135         File f = new File(Environment.getExternalStorageDirectory()
              +File.separator+appDirName+File.separator+faceID);
136         String[] filenames = f.list();
137         for (int i=0; i<filenames.length; i++) {
138           if (filenames[i].endsWith(".id")) {
139             vardas = filenames[i].substring(0, filenames[i].length
                  () - 3);
140             break;
141           }
142         }
143
144         TextView t=new TextView(this);
145         t=(TextView)findViewById(R.id.myTxtView);
```

```
146
147         if ( faceConf >= 0.4 ) {
148           t.setText("ID: " + faceID + "  Vardas: " + vardas + "\
                nAtstumas = "+ String.format("%.6f", faceDist) + "\
                nPatikimumas = " + String.format("%2.2f %%", faceConf
                *100) );
149         }
150         else {
151           t.setText("ID: " + faceID + " Vardas: NEZINOMAS " + "
                Patikimumas = " + String.format("%2.2f %%", faceConf
                *100) );
152         }
153
154         return true;
155       case R.id.menu_recognize_all:
156         recAllFaces();
157         return true;
158       case R.id.menu_genlist:
159         // Single menu item is selected do something
160         // Ex: launching new activity/screen or show alert message
161         Toast.makeText(this, "Generuojamas bylu sarasas...", Toast.
              LENGTH_SHORT).show();
162
163         FS.genList();
164         return true;
165
166       case R.id.menu_learn:
167         Toast.makeText(this, "Apsimokoma...", Toast.LENGTH_SHORT).
              show();
168         jniLearn();
169         return true;
170       case R.id.menu_RealTime:
171         Log.v(TAG, "Case: " + item);
172
173         Intent myCameraRealTime = new Intent(FotoIntentActivity.
              this, RealTime.class);
174         myCameraRealTime.putExtra(MediaStore.EXTRA_OUTPUT, fileUri)
              ;
175         Log.v(TAG, "Intent created: " + myCameraRealTime);
176
177         startActivityForResult(myCameraRealTime,
              CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
178         //startActivityForResult(new Intent(FotoIntentActivity.this
              , ViewfinderEE368.class),
              CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
179
180         return true;
181       default:
182         return super.onOptionsItemSelected(item);
183       }
184     }
185
```

```
186   // nustatomi mygtuko "Nuotrauka" veiksmai ir pasirinkimo langas
187   public void photoChooserButton() {
188     myButtonSave.setVisibility(View.INVISIBLE);
189
190     myButtonFoto.setOnClickListener(new OnClickListener() {
191       public void onClick(View v) {
192         // Iskvieciama veido atpazinimo funkcija
193
194         List<String> listButtonFoto = new ArrayList<String>();
195         listButtonFoto.add("Gamykline kamera");
196         listButtonFoto.add("Kamera su veidu aptikimu");
197         listButtonFoto.add("Galerija");
198
199         final CharSequence[] items = listButtonFoto.toArray(new
              String[listButtonFoto.size()]);
200
201         AlertDialog.Builder builder = new AlertDialog.Builder(
              FotoIntentActivity.this);
202         builder.setTitle("Pasirinkite metoda");
203
204         builder.setItems(items, new DialogInterface.
              OnClickListener() {
205           public void onClick(DialogInterface dialog, int item) {
206             Toast.makeText(getApplicationContext(), "Pasirinktas
                  metodas: " + items[item], Toast.LENGTH_SHORT).show
                  ();
207             Log.v(TAG, "item: " + item);
208
209             fileUri = getOutputMediaFileUri(); // create a file
                  to save the image
210
211             switch (item) {
212               case 0: // Foto intent
213                 Log.v(TAG, "Case: " + item);
214
215 //------------------------------------------------------------//
216 //-- Sukuriamas INTENT kamerai bei URI nuotraukai --//
217 //------------------------------------------------------------//
218
219                 // create Intent to take a picture and return
                      control to the calling application
220                 final Intent intentCamera = new Intent(MediaStore
                      .ACTION_IMAGE_CAPTURE);
221                 intentCamera.putExtra(MediaStore.EXTRA_OUTPUT,
                      fileUri); // set the image file name
222                 Log.v(TAG, "Intent created: " + intentCamera);
223
224                 // start the image capture Intent
225                 startActivityForResult(intentCamera,
                      CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
226                 break;
227
```

```
228                 case 1: // myCamera
229                   Log.v(TAG, "Case: " + item);
230
231                   Intent myCameraIntent = new Intent(
                          FotoIntentActivity.this, myCameraIntent.class)
                          ;
232                   myCameraIntent.putExtra(MediaStore.EXTRA_OUTPUT,
                          fileUri);
233                   Log.v(TAG, "Intent created: " + myCameraIntent);
234
235                   startActivityForResult(myCameraIntent,
                          CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
236
237                   //startActivityForResult(new Intent(
                          FotoIntentActivity.this, ViewfinderEE368.class
                          ), CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
238
239                   break;
240                 case 2: // galery
241                   Log.v(TAG, "Case: " + item);
242
243                   final Intent intentGallery = new Intent(Intent.
                          ACTION_PICK, android.provider.MediaStore.
                          Images.Media.EXTERNAL_CONTENT_URI);
244                   intentGallery.putExtra(MediaStore.EXTRA_OUTPUT,
                          fileUri); // set the image file name
245                   Log.v(TAG, "Intent created: " + intentGallery);
246
247                   startActivityForResult(intentGallery,
                          CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
248
249                   break;
250               }
251             }
252           });
253           builder.show();
254         }
255       });
256     }
257
258     // Aprasomi veiksmai, kurie atliekami gavus
259     // rezultata is ikviestos veiklos
260     @Override
261     protected void onActivityResult(int requestCode, int resultCode
          , Intent data) {
262
263       /** ************************************************ */
264       /** Gaunama nuotrauka kuri issaugoma i atminties kortele, */
265       /** o veliau atnaujinama pagrindiniame lange          */
266       /** ************************************************ */
267       // Receiving camera intent result //
268       if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE) {
```

```
269         if (resultCode == RESULT_OK) {
270           // Image captured and saved to fileUri specified in the
                Intent
271           Toast.makeText(this, "Image saved", Toast.LENGTH_SHORT).
                show();
272           updateMainFace();
273           Log.v(TAG, "On activity result: Image saved");
274         } else if (resultCode == RESULT_CANCELED) {
275           // User cancelled the image capture
276           Toast.makeText(this, "Canceled bu user", Toast.
                LENGTH_SHORT).show();
277           Log.v(TAG, "On activity result: canceled by user");
278         } else {
279           // Image capture failed, advise user
280           Toast.makeText(this, "Capturing failed", Toast.
                LENGTH_SHORT).show();
281           Log.v(TAG, "On activity result: capturing failed");
282         }
283       }
284
285     }
286
287     // Aprasoma URI funkcija failo saugojimui
288     private static Uri getOutputMediaFileUri(){
289       return Uri.fromFile(getOutputMediaFile());
290     }
291
292     // sukuriama byla nuotraukos saugojimui
293     public static File getOutputMediaFile(){
294
295       /** SUKURIAMA DIREKTORIJA */
296       // To be safe, you should check that the SDCard is mounted
297       // using Environment.getExternalStorageState() before doing
            this.
298       File mediaStorageDir = new File(Environment.
            getExternalStorageDirectory(), appDirName);
299       Log.v(TAG, "mediaStorageDir: " + mediaStorageDir);
300       // Create the storage directory if it does not exist
301       if (! mediaStorageDir.exists()){
302         if (! mediaStorageDir.mkdirs()){
303           Log.d("MyCameraApp", "failed to create directory");
304           return null;
305         }
306       }
307       else {
308         Log.v(TAG, "Storage DIR already exists");
309       }
310
311       /** SUKURIAMA NUOTRAUKA */
312       String foto_file = mediaStorageDir.getPath() + File.separator
              + imageName + ".jpg";
313       File mediaFile = new File(foto_file);
```

```
314       Log.v(TAG, "mediaFile: " + foto_file);
315       if (mediaFile.exists()){
316         Log.v(TAG, "Media file already exists");
317       }
318
319       return mediaFile;
320     }
321
322     // Atnaujina pagrindini paveiksleli
323     public void updateMainFace() {
324       // Nuskaitom nuotrauka
325       BitmapFactory.Options BitmapFactoryOptions = new
              BitmapFactory.Options();
326       BitmapFactoryOptions.inPreferredConfig = Bitmap.Config.
              RGB_565;
327       Log.v(TAG, Environment.getExternalStorageDirectory().toString
              ()+File.separator+appDirName+File.separator+imageName);
328       Bitmap original_bMap = BitmapFactory.decodeFile(Environment.
              getExternalStorageDirectory().toString()+File.separator+
              appDirName+File.separator+imageName+".jpg",
              BitmapFactoryOptions);
329
330       // Pakeiciam nuotraukos dydi RATIO kartus
331       bMap = getResizedBitmap(original_bMap, original_bMap.
              getHeight()/ratio , original_bMap.getWidth()/ratio);
332
333       Log.v(TAG, "Old dimensions:   Height=" + original_bMap.
              getHeight() + "   Width=" + original_bMap.getWidth());
334       Log.v(TAG, "New dimensions:   Height=" + bMap.getHeight() +
                    " Width=" + bMap.getWidth());
335
336       original_bMap.recycle();
337
338       // Atnaujinam nuotrauka pagrindiniame lange
339       image.setImageBitmap(bMap);
340       Log.v(TAG, "Face has been updated");
341
342       // Igalinamas migtukas veidu aptikimui
343       enableButtonDetect();
344
345     }
346
347     // Sumazina gauta paveiksla
348     public Bitmap getResizedBitmap(Bitmap bm, int newHeight, int
              newWidth) {
349       // gaunas originalios nuotraukos ismatavimus
350       int width = bm.getWidth();
351       int height = bm.getHeight();
352
353       // apskaiciuojam santiki naujas dydis / senas dydis
354       float scaleWidth = ((float) newWidth) / width;
355       float scaleHeight = ((float) newHeight) / height;
```

```
356
357     // create a matrix for the manipulation
358     Matrix matrix = new Matrix();
359
360     // resize the bit map
361     matrix.postScale(scaleWidth, scaleHeight);
362
363     // recreate the new Bitmap
364     Bitmap resizedBitmap = Bitmap.createBitmap(bm, 0, 0, width,
           height, matrix, false);
365     return resizedBitmap;
366   }
367
368   /** ********************************** */
369   /**      VEIDU APTIKIMAS / ATPAZINIMAS      */
370   /** ********************************** */
371
372   // Igalinamas mygtukas skirtas veidu aptikimui ir suzymejimui
373   public void enableButtonDetect() {
374
375     // Nustatom mygtuko teksto spalva i JUODA
376     myButtonDetect.setTextColor(Color.parseColor("#000000"));
377
378     // Laukiame kol bus paspaustas mygtukas
379     myButtonDetect.setOnClickListener(new OnClickListener() {
380       public void onClick(View v) {
381         // Iskvieciama veido atpazinimo funkcija
382         MyFaceDetection();
383       }
384     });
385   }
386
387   public int imageWidth, imageHeight; // paveikslo raiska
388   public static final int maxNumberOfFace = 20; // maksimalus
       ieskoma veidu kiekis
389   public int numberOfFaceDetected; // aptiktu veidu skaicius
390
391   // veidu detektorius
392   public FaceDetector myFaceDetectArray;
393   public FaceDetector.Face myFaces[] = new FaceDetector.Face[
       maxNumberOfFace];
394   public FaceDetector.Face currentFace = null;
395
396   // veido bruozai
397   public PointF myEyeMidPoint[] = new PointF[maxNumberOfFace];
398   public float  myEyesDistance[]  = new float[maxNumberOfFace];
399   public float  myConfidence[]  = new float[maxNumberOfFace];
400
401   // veidu aptikima
402   public void MyFaceDetection() {
403
404     imageWidth = bMap.getWidth();
```

```
405       imageHeight = bMap.getHeight();
406
407       myFaceDetectArray = new FaceDetector(imageWidth, imageHeight,
              maxNumberOfFace);
408       numberOfFaceDetected = myFaceDetectArray.findFaces(bMap,
              myFaces);
409       Log.v(TAG, "Number Of faces detected: " +
              numberOfFaceDetected);
410
411       // Jei aptiktas nors vienas veida ...
412       if (numberOfFaceDetected > 0) {
413
414         // isvedam zinute i ekrana su aptiktu veidu skaiciumi
415         Toast.makeText(this, "Aptikta veidu: " +
              numberOfFaceDetected, Toast.LENGTH_LONG).show();
416
417         // Suzymimi aptikti veidai
418         Bitmap bMapFaces;
419         bMapFaces = overlayBitmap(bMap);
420         image.setImageBitmap(bMapFaces);
421
422         // Igalinamas mygtukas "Extract"
423         enableButtonExtract();
424       }
425       else {
426         // Perspejam kad nebuvo aptiktas nei vienas veidas
427         Toast.makeText(this, "Neaptiktas nei vienas veidas\
              nMeginkite dar karta fotografuoti", Toast.LENGTH_LONG).
              show();
428       }
429     }
430
431     // Suzymi aptiktus veidus
432     private Bitmap overlayBitmap(Bitmap bmp1) {
433
434     // Sukuriamas naujas paveikslas
435       Bitmap bmOverlay = Bitmap.createBitmap(bmp1.getWidth(), bmp1.
              getHeight(), bmp1.getConfig());
436
437       // Sukuriama nauja piesimo drobe
438       Canvas canvas = new Canvas(bmOverlay);
439
440       // Drobeje nupiesiame originalu paveiksla, kuri modifikuosime
441       canvas.drawBitmap(bmp1, new Matrix(), null);
442
443       // Sukuriamas piesinys
444       Paint myPaint = new Paint();
445       myPaint.setColor(Color.GREEN);
446       myPaint.setStyle(Paint.Style.STROKE);
447       myPaint.setStrokeWidth(3);
448
449       // i-tasis veidas
```

```
450     for(int i=0; i < numberOfFaceDetected; i++)
451     {
452       currentFace = myFaces[i];
453
454       PointF myEyeMP = new PointF();
455       currentFace.getMidPoint(myEyeMP);
456       myEyeMidPoint[i] = myEyeMP;
457       myEyesDistance[i] = currentFace.eyesDistance();
458       myConfidence[i] = currentFace.confidence();
459
460       /** Aplink i-taji veida nubreziamas staciakampis **/
461       // Kaire linija
462       canvas.drawLine((myEyeMidPoint[i].x-myEyesDistance[i]), (
              myEyeMidPoint[i].y-myEyesDistance[i]*2/2),
463             (myEyeMidPoint[i].x-myEyesDistance[i]), (
                  myEyeMidPoint[i].y+myEyesDistance[i]*2),
464             myPaint);
465       // Desine linija
466       canvas.drawLine((myEyeMidPoint[i].x+myEyesDistance[i]), (
              myEyeMidPoint[i].y-myEyesDistance[i]*2/2),
467             (myEyeMidPoint[i].x+myEyesDistance[i]), (
                  myEyeMidPoint[i].y+myEyesDistance[i]*2),
468             myPaint);
469       // Virsutine linija
470       canvas.drawLine((myEyeMidPoint[i].x-myEyesDistance[i]), (
              myEyeMidPoint[i].y-myEyesDistance[i]*2/2),
471             (myEyeMidPoint[i].x+myEyesDistance[i]), (
                  myEyeMidPoint[i].y-myEyesDistance[i]*2/2),
472             myPaint);
473       // Apatine linija
474       canvas.drawLine((myEyeMidPoint[i].x-myEyesDistance[i]), (
              myEyeMidPoint[i].y+myEyesDistance[i]*2),
475             (myEyeMidPoint[i].x+myEyesDistance[i]), (
                  myEyeMidPoint[i].y+myEyesDistance[i]*2),
476             myPaint);
477       // ir jo viduryje parasomas jo numeris
478       myPaint.setTextSize(20);
479       canvas.drawText(""+i, (myEyeMidPoint[i].x - myEyesDistance[
              i]/4), (myEyeMidPoint[i].y + myEyesDistance[i]/4),
              myPaint);
480     }
481     // grazinama nuotrauka su piesineliais
482     return bmOverlay;
483   }
484
485   // Igalina mygtuka skirta veidu isskyrimui
486   public void enableButtonExtract() {
487
488     // Nustatom mygtuko teksto spalva i JUODA
489     myButtonExtract.setTextColor(Color.parseColor("#000000"));
490
491     // Laukiame kol bus paspaustas mygtukas
```

```
492     myButtonExtract.setOnClickListener(new OnClickListener() {
493       public void onClick(View v) {
494         // Iskvieciama veido atpazinimo funkcija
495         List<String> strings = new ArrayList<String>();
496         for (int i = 0; i < numberOfFaceDetected; i++) {
497           strings.add("Nr. "+i);
498         }
499 
500         final CharSequence[] items = strings.toArray(new String[
              strings.size()]);
501 
502         AlertDialog.Builder builder = new AlertDialog.Builder(
              FotoIntentActivity.this);
503         builder.setTitle("Pasirinkite veida");
504 
505         builder.setItems(items, new DialogInterface.
              OnClickListener() {
506           public void onClick(DialogInterface dialog, int item) {
507             Toast.makeText(getApplicationContext(), "Pasirinktas
                  veidas " + items[item], Toast.LENGTH_SHORT).show()
                  ;
508             Log.v(TAG, "Pasirinktas veidas: " + item);
509             extractFace(item);
510           }
511         });
512         builder.show();
513       }
514     });
515   }
516 
517   // Isskiria is nuotraukos pasirinkta veida
518   private void extractFace(int i) {
519     int x   = (int)(myEyeMidPoint[i].x - myEyesDistance[i]);
520     int y   = (int)(myEyeMidPoint[i].y - myEyesDistance[i]);
521     int width = (int)(myEyesDistance[i]*2);
522     int heigth  = (int)(myEyesDistance[i]*3);
523 
524     // patikrinam kad neiseitumem uz nuotraukos ribu
525     if (x < 0) {
526       x = 0;
527     }
528     if (x > bMap.getWidth()) {
529       x = bMap.getWidth();
530     }
531     if (x+width > bMap.getWidth()) {
532       width = bMap.getWidth() - x;
533     }
534     if (y < 0) {
535       y = 0;
536     }
537     if (y > bMap.getHeight()) {
538       y = bMap.getHeight();
```

```
539       }
540       if (y+heigth > bMap.getHeight()) {
541         heigth = bMap.getHeight() - y;
542       }
543
544       faceExtracted = Bitmap.createBitmap(
545         bMap,
546         x,  // Left
547         y, // Top
548         width, // Right
549         heigth // Botom
550       );
551
552       image.setImageBitmap(faceExtracted); // iskerpam veida
553       saveExtractedFace(faceExtracted); // ji issaugom
554       enableButtonSave(); // igalinam SAVE mygtuka
555     }
556
557     // igalina save mygtuka
558     public void enableButtonSave() {
559
560       // Nustatom mygtuko teksto spalva i JUODA
561       myButtonSave.setVisibility(View.VISIBLE);
562
563       // Laukiame kol bus paspaustas mygtukas
564       myButtonSave.setOnClickListener(new OnClickListener() {
565         public void onClick(View v) {
566
567           final CharSequence[] items    = FS.getFolderNames(); //
                KATALOGU PAVADINIMAI
568           final CharSequence[] userIDs = FS.getUserIDs();    //
                KATALOGAMS PRISKIRTU ASMENU VARDAI
569
570           AlertDialog.Builder builder = new AlertDialog.Builder(
                FotoIntentActivity.this);
571           builder.setTitle("Pasirinkite katalogа");
572
573           builder.setItems(userIDs, new DialogInterface.
                OnClickListener() {
574             public void onClick(DialogInterface dialog, int item) {
575               if (item==0){
576                 Log.v(TAG, "Pasirinkta sukurti nauja kataloga");
577
578                 AlertDialog.Builder alert = new AlertDialog.Builder
                      (FotoIntentActivity.this);
579                 alert.setTitle("Naujas katalogas");
580                 alert.setMessage("iveskite pavadinima:");
581                 // Set an EditText view to get user input
582                 final EditText input = new EditText(
                      FotoIntentActivity.this);
583                 alert.setView(input);
```

```
584                 alert.setPositiveButton("Ok", new DialogInterface.
                        OnClickListener() {
585                   public void onClick(DialogInterface dialog, int
                          whichButton) {
586                     String value = ""+input.getText();
587                     String newFolderName = FS.newFolderName();
588                     if (FS.createDirIfNotExists(newFolderName)) {
589                       saveBitmap(faceExtracted, newFolderName);
590                       FS.createID(newFolderName, value);
591                       Toast.makeText(FotoIntentActivity.this, "
                              Naujas katalogas \""+value+"\" sukurtas\
                              nPaveikslas issaugotas", Toast.LENGTH_LONG
                              ).show();
592                     }
593                   }
594                 });
595                 alert.setNegativeButton("Cancel", new
                        DialogInterface.OnClickListener() {
596                   public void onClick(DialogInterface dialog, int
                          whichButton) {
597                   // Canceled.
598                   Toast.makeText(getApplicationContext(), "
                          Paveikslas neissaugotas", Toast.LENGTH_SHORT).
                          show();
599                   }
600                 });
601                 alert.show();
602               }
603               else {
604                 Toast.makeText(getApplicationContext(), "
                        Pasirinktas katalogas: " + items[item], Toast.
                        LENGTH_SHORT).show();
605                 saveBitmap(faceExtracted, items[item].toString());
606               }
607             }
608           });
609           builder.show();
610         }
611       });
612     }
613
614     // issaugomas paveikslas
615     public void saveBitmap(Bitmap mBitmap, String dir) {
616       String strFilename;
617       String path;
618
619       strFilename = FS.newFileName(dir);
620       path = Environment.getExternalStorageDirectory().toString()+
              File.separator+appDirName+File.separator+dir;
621
622
623       try{
```

```
624         Log.v(TAG, path);
625
626         OutputStream fOut = null;
627         File file = new File(path, strFilename+".jpg");
628         fOut = new FileOutputStream(file);
629
630         Bitmap mBitmapResizedFace;
631         mBitmapResizedFace = getResizedBitmap(mBitmap,
              FINAL_FACE_HEIGHT_PX, FINAL_FACE_WIDTH_PX);
632         mBitmapResizedFace.compress(Bitmap.CompressFormat.JPEG,
              100, fOut);
633         fOut.flush();
634         fOut.close();
635
636         MediaStore.Images.Media.insertImage(getContentResolver(),
              file.getAbsolutePath(),file.getName(),file.getName());
637       }
638       catch (Exception e) {
639         e.printStackTrace();
640       }
641     }
642
643     // issaugomas iskirptas veidas
644     public void saveExtractedFace(Bitmap mBitmap) {
645       String strFilename = "LastFace.jpg";
646       String path;
647
648       path = Environment.getExternalStorageDirectory().toString()+
            File.separator+appDirName;
649
650       try{
651         Log.v(TAG, path);
652
653         OutputStream fOut = null;
654         File file = new File(path, strFilename);
655         fOut = new FileOutputStream(file);
656
657         Bitmap mBitmapResizedFace;
658         mBitmapResizedFace = getResizedBitmap(mBitmap,
              FINAL_FACE_HEIGHT_PX, FINAL_FACE_WIDTH_PX);
659         mBitmapResizedFace.compress(Bitmap.CompressFormat.JPEG,
              100, fOut);
660         fOut.flush();
661         fOut.close();
662
663         MediaStore.Images.Media.insertImage(getContentResolver(),
              file.getAbsolutePath(),file.getName(),file.getName());
664       }
665       catch (Exception e) {
666         e.printStackTrace();
667       }
668     }
```

```
669
670     // visu veidu atpazinimas
671     public void recAllFaces() {
672
673       String path = Environment.getExternalStorageDirectory().
             toString()+File.separator+appDirName+File.separator+"
             allFaces";
674       String testAll = "testAllFaces.txt";
675
676       // istrina visus failus is katalogo
677       if (numberOfFaceDetected != 0) {
678         File directory = new File(path);
679         // Get all files in directory
680         File[] files = directory.listFiles();
681         for (File file : files)
682         {
683           // Delete each file
684           if (!file.delete())
685           {
686             // Failed to delete file
687             Log.e("TAG", "Failed to delete "+file);
688           }
689          }
690         Log.v(TAG, "Visi failai istrinti");
691
692         // apskaiciuojamos veido esminiu tasku koordinates
693         for (int i=0; i<numberOfFaceDetected; i++) {
694           int x    = (int)(myEyeMidPoint[i].x - myEyesDistance[i]);
695           int y    = (int)(myEyeMidPoint[i].y - myEyesDistance[i]);
696           int width = (int)(myEyesDistance[i]*2);
697           int heigth  = (int)(myEyesDistance[i]*3);
698
699           // patikrinama ar neisanama uz ribu
700           if (x < 0) {
701             x = 0;
702           }
703           if (x > bMap.getWidth()) {
704             x = bMap.getWidth();
705           }
706           if (x+width > bMap.getWidth()) {
707             width = bMap.getWidth() - x;
708           }
709           if (y < 0) {
710             y = 0;
711           }
712           if (y > bMap.getHeight()) {
713             y = bMap.getHeight();
714           }
715           if (y+heigth > bMap.getHeight()) {
716             heigth = bMap.getHeight() - y;
717           }
718
```

```
719           // iskerpamas veidas
720           faceExtracted = Bitmap.createBitmap(
721             bMap,
722             x,     // Left
723             y,     // Top
724             width,   // Right
725             heigth   // Botom
726           );
727           Log.v(TAG, "Veidas iskirtpas");
728           Log.v(TAG, "path="+path);
729
730           // Issaugo visus veidus kaip paveikslus
731           String strFilename = "" + i + ".jpg";
732           try{
733               Log.v(TAG, path);
734
735               OutputStream fOut = null;
736             File file = new File(path, strFilename);
737             fOut = new FileOutputStream(file);
738
739             Bitmap mBitmapResizedFace;
740             mBitmapResizedFace = getResizedBitmap(faceExtracted,
                    FINAL_FACE_HEIGHT_PX, FINAL_FACE_WIDTH_PX);
741             mBitmapResizedFace.compress(Bitmap.CompressFormat.JPEG,
                    100, fOut);
742             fOut.flush();
743             fOut.close();
744
745             MediaStore.Images.Media.insertImage(getContentResolver
                    (),file.getAbsolutePath(),file.getName(),file.
                    getName());
746           }
747           catch (Exception e) {
748             e.printStackTrace();
749           }
750           Log.v(TAG, "Veidas issaugotas");
751           faceExtracted.recycle();
752         }
753
754         /** ************************************* **/
755         /** Sugeneruoja kataloge esanciu bylu sarasa **/
756         /** ************************************* **/
757         File file = new File(path);
758         File[] rootFiles = file.listFiles();
759         File textFile = new File(path + File.separator + testAll);
760
761         try {
762           textFile.createNewFile();
763           BufferedWriter buf = new BufferedWriter(new FileWriter(
                  textFile, true));
764
765           int[][] ID_NR = new int[numberOfFaceDetected][2];
```

```
766
767           for ( int i=0; i<rootFiles.length; i++ ) {
768             Log.v(TAG, "FILE: " + rootFiles[i].toString() );
769             buf.write("0 " + rootFiles[i].toString() + "\n" );
770           }
771
772           buf.close();
773         } catch (IOException e) {
774           // TODO Auto-generated catch block
775           e.printStackTrace();
776         }
777       }
778
779       // iskvieciama NATIVE veidu atpazinimo funkcija
780       JavaCls result = jniRecognizeAll();
781
782       Log.v(TAG, "DONE JNI");
783
784       int numberOfPersons;
785       if (numberOfFaceDetected == 0) {
786         File file = new File(path);
787         Log.v(TAG, "path ="+path+" length =" + file.listFiles().
              length);
788         numberOfPersons = file.listFiles().length - 1;
789       }
790       else {
791         numberOfPersons = numberOfFaceDetected;
792       }
793
794       for (int j=0; j<numberOfPersons; j++) {
795         Log.v(TAG, "fID ="+result.arrFaceID[j] + " Dist =" +
              result.arrDist[j] + " Conf =" + result.arrConf[j]);
796       }
797
798
799       // Gaunamas vardas pagal ID
800       String[] vardas = new String[numberOfPersons];
801       for (int j=0; j<numberOfPersons; j++) {
802         File f = new File(Environment.getExternalStorageDirectory()
              +File.separator+appDirName+File.separator+result.
              arrFaceID[j]);
803 Log.v(TAG, f.getAbsolutePath());
804         String[] filenames = f.list();
805         for (int i=0; i<filenames.length; i++) {
806 Log.v(TAG, filenames[i].toString());
807           if (filenames[i].endsWith(".id")) {
808             vardas[j] = filenames[i].substring(0, filenames[i].
                  length() - 3);
809             break;
810           }
811         }
812       }
```

```
813 Log.v(TAG, "Kiek vardu: "+vardas.length);
814
815     String buffer = "";
816     for (int i=0; i<numberOfPersons; i++) {
817       if ( result.arrConf[i] <= 0.4 ) {
818         buffer = buffer +
819               (numberOfPersons-i-1) + " : NEZINOMAS (panasiausias
                  - " + vardas[i] + " - " + String.format("%.2f %%"
                   , result.arrConf[i] * 100) + ")\n";
820       }
821       else {
822         buffer = buffer + (numberOfPersons-i-1) + " : " + vardas[
                  i] + " - " + String.format("%.2f %%", result.arrConf[i
                  ] * 100) + "\n";
823       }
824     }
825     Log.v(TAG, buffer);
826
827     // atnaujinama informacija ekrane apie atpazintus veidus
828     TextView t=new TextView(this);
829     t=(TextView)findViewById(R.id.myTxtView);
830     t.setMovementMethod(ScrollingMovementMethod.getInstance());
831     t.setText(buffer);
832   }
833
834 }
```

SRC/COM/BALINSKAS/FOTOINTENT/JAVACLS.JAVA

```
1  /*********************************
2  * Grazina atpazinto veido / veidu ID,
3  * Euklido kvadratini atstuma ir
4  * patikimumas
5  *********************************/
6
7  package com.balinskas.fotointent;
8
9  public class JavaCls {
10   public int    faceID;
11   public double dist;
12   public float  conf;
13
14   public int[]    arrFaceID =
15     new int[FotoIntentActivity.maxNumberOfFace];
16   public double[] arrDist   =
17     new double[FotoIntentActivity.maxNumberOfFace];
18   public float[]  arrConf   =
19     new float[FotoIntentActivity.maxNumberOfFace];
20 }
```

SRC/COM/BALINSKAS/FOTOINTENT/MYFILES.JAVA

```
1  /** IVAIRIOS FUNKCIJOS SKIRTOS DARBUI SU BYLOMIS IR KATALOGAIS */
2
3  package com.balinskas.fotointent;
4
5  import java.io.BufferedWriter;
6  import java.io.File;
7  import java.io.FileWriter;
8  import java.io.IOException;
9  import java.util.ArrayList;
10 import java.util.List;
11 import android.os.Environment;
12 import android.util.Log;
13
14 public class myFiles {
15
16   private static final String TAG = "myFiles";
17
18   // Sugeneruoja failu sarasa kuris bus naudojamas algoritmo
       apmokymui
19   public void genList() {
20     String path = Environment.getExternalStorageDirectory().
         toString()+File.separator+FotoIntentActivity.appDirName;
21     File file = new File(path);
22     File[] rootFiles = file.listFiles();
23     File textFile = new File(path + File.separator +
         FotoIntentActivity.listName);
24     Boolean isException = false;
25
26     try {
27       if (textFile.exists()) {
28         textFile.delete();
29         Log.v(TAG, FotoIntentActivity.listName + "␣exists.␣
             Replacing␣it");
30       }
31       textFile.createNewFile();
32       BufferedWriter buf = new BufferedWriter(new FileWriter(
           textFile, true));
33
34       for ( int i=0; i<rootFiles.length; i++ ) {
35         if (rootFiles[i].isDirectory()) {
36
37           String dirFiles[] =  rootFiles[i].list();
38
39           try {
40             @SuppressWarnings("unused")
41             int temp  = Integer.parseInt(rootFiles[i].getName());
42           } catch (Exception e) {
43             isException = true;
```

```
44              Log.e(TAG, "Exception in folder name... Unable to
                    cast folder name to INT");
45            }
46
47            if (!isException) {
48              for (int j = 0; j < dirFiles.length; j++) {
49                if (dirFiles[j].endsWith(".jpg")) {
50 //                 Log.v(TAG, "FILE:  " + rootFiles[i].
    getAbsolutePath() + File.separator + dirFiles[j].toString());
51                  buf.write(rootFiles[i].getName() + " " +
                        rootFiles[i].getAbsolutePath() + File.
                        separator + dirFiles[j].toString() + "\n" );
52                }
53              }
54            }
55            isException = false;
56          }
57        }
58        buf.close();
59      } catch (IOException e) {
60        e.printStackTrace();
61      }
62    }
63
64    // Sugeneruoja nauja (INT) failo pavadinima, kuriuo galima
        issaugoti byla
65    public String newFileName(String name) {
66
67      String path;
68      int intFilename = 0;
69      String strFilename;
70
71      path = Environment.getExternalStorageDirectory().toString()+
          File.separator+FotoIntentActivity.appDirName+File.
          separator+name;
72
73      // Gaunamas kataloge esanciu failu sarasas ir parenkamas
          naujas failo pavadinimas (INT) kuris dar neegzistuoja
74      File f = new File(path);
75      File[] files = f.listFiles();
76
77      int currentFile;
78      int largestFile = 0;
79      for (File inFile : files) {
80        if (inFile.isFile()) {
81          // is directory
82          Log.v(TAG, "Failas :" + inFile.getName() + " Parsed: " +
              inFile.getName().substring(0, inFile.getName().length
              ()-4));
83
84          try {
```

```
85            currentFile = Integer.parseInt(inFile.getName().
                  substring(0, inFile.getName().length()-4));
86          } catch (Exception e) {
87            currentFile = 0;
88            Log.v(TAG, "Exception");
89          }
90
91          if (currentFile >= largestFile) {
92            largestFile = currentFile;
93            Log.v(TAG, "Filename incremented =" + largestFile);
94          }
95        }
96      }
97
98      intFilename = largestFile + 1;
99      strFilename =  "" + intFilename;
100
101     return strFilename;
102   }
103
104   // Sugeneruoja nauja (INT) katalogo pavadinima, kuriame galima
        issaugoti bylas
105   public String newFolderName() {
106
107     String path;
108     int intDir = 0;
109     String strDirName;
110
111     path = Environment.getExternalStorageDirectory().toString()+
          File.separator+FotoIntentActivity.appDirName;
112
113     // Gaunamas kataloge esanciu failu sarasas ir parenkamas
          naujas failo pavadinimas (INT) kuris dar neegzistuoja
114     File f = new File(path);
115     File[] files = f.listFiles();
116
117     int currentDir;
118     int largestDir = 0;
119     for (File inFile : files) {
120       if (inFile.isDirectory()) {
121         try {
122           currentDir = Integer.parseInt(inFile.getName());
123         } catch (Exception e) {
124           currentDir = 0;
125         }
126         if (currentDir >= largestDir) {
127           largestDir = currentDir;
128         }
129       }
130     }
131
132     intDir = largestDir + 1;
```

```
133       strDirName =  "" + intDir;
134
135       return strDirName;
136     }
137
138     // Sukuria programos direktorija, jei tokia neegzistuoja
139     public boolean createDirIfNotExists(String newDir) {
140
141       boolean ret = true;
142
143       String path = Environment.getExternalStorageDirectory().
              toString()+File.separator+FotoIntentActivity.appDirName+
              File.separator+newDir;
144       File file = new File(path);
145
146       if (!file.exists()) {
147         if (!file.mkdirs()) {
148           Log.e("TravellerLog :: ", "Problem creating Image folder"
                  );
149           ret = false;
150         }
151       }
152       return ret;
153     }
154
155     // Sukuria ID faila
156     public boolean createID(String DIR, String name) {
157
158       boolean ret = true;
159
160       String path = Environment.getExternalStorageDirectory().
              toString()+File.separator+FotoIntentActivity.appDirName+
              File.separator+DIR+File.separator+name+".id";
161       File file = new File(path);
162
163       if (!file.exists()) {
164         try {
165           if (!file.createNewFile()) {
166             Log.e("TravellerLog :: ", "Problem creating Image
                    folder");
167             ret = false;
168           }
169         } catch (IOException e) {
170           // TODO Auto-generated catch block
171           e.printStackTrace();
172         }
173       }
174       return ret;
175     }
176
177     // sugeneruoja katalogu sarasa
178     public CharSequence[] getFolderNames() {
```

```
179     List<String> dirNames = new ArrayList<String>();
180
181     dirNames.add("Sukurti nauja kataloga");
182
183     File f = new File(Environment.getExternalStorageDirectory(),
            FotoIntentActivity.appDirName);
184     File[] files = f.listFiles();
185
186     for (int index=0; index < files.length; index++) {
187       if (files[index].isDirectory()) {
188         try {
189           // Patikrinam ar direktorijos pavadinimas yra int
190           @SuppressWarnings("unused")
191           int temp = Integer.parseInt(files[index].getName());
192           // jei taip - pridedam i sarasa
193           dirNames.add(files[index].getName());
194         } catch (Exception e) {
195           e = null;
196         }
197       }
198     }
199     final CharSequence[] items = dirNames.toArray(new String[
            dirNames.size()]);
200     return items;
201   }
202
203   // sugeneruoja vartotoju esanciu DB sarasa
204   public CharSequence[] getUserIDs() {
205     List<String> IDs = new ArrayList<String>();
206
207     IDs.add("Sukurti nauja kataloga");
208
209     File f = new File(Environment.getExternalStorageDirectory(),
            FotoIntentActivity.appDirName);
210     File[] files = f.listFiles();
211
212
213     for (int index=0; index < files.length; index++) {
214       if (files[index].isDirectory()) {
215         try {
216           // Patikrinam ar direktorijos pavadinimas yra int
217           @SuppressWarnings("unused")
218           int temp = Integer.parseInt(files[index].getName());
219
220           File nameInDB = new File(files[index].getAbsolutePath()
                );
221           String[] filenames = nameInDB.list();
222           for(int i = 0; i<filenames.length; i++) {
223             if (filenames[i].endsWith(".id")) {
224               IDs.add(filenames[i].substring(0, filenames[i].
                      length() - 3));
225             }
```

```
226             }
227           } catch (Exception e) {
228 //            Log.e(TAG, "Unable to convert file: " + files[index].
      getName());
229             e = null;
230           }
231         }
232       }
233       final CharSequence[] items = IDs.toArray(new String[IDs.size
          ()]);
234       return items;
235     }
236 }
```

SRC/COM/BALINSKAS/FOTOINTENT/REALTIME.JAVA

```
1  /** VEIDU ATPAZINIMAS REALIU LAIKU */
2
3  package com.balinskas.fotointent;
4
5  import java.io.File;
6  import java.io.FileNotFoundException;
7  import java.io.FileOutputStream;
8  import java.io.IOException;
9
10 import android.app.Activity;
11 import android.content.Context;
12 import android.graphics.Bitmap;
13 import android.graphics.Bitmap.Config;
14 import android.graphics.Canvas;
15 import android.graphics.Color;
16 import android.graphics.Matrix;
17 import android.graphics.Paint;
18 import android.graphics.PointF;
19 import android.hardware.Camera;
20 import android.hardware.Camera.PictureCallback;
21 import android.hardware.Camera.PreviewCallback;
22 import android.media.FaceDetector;
23 import android.os.Bundle;
24 import android.os.Environment;
25 import android.util.Log;
26 import android.view.SurfaceHolder;
27 import android.view.SurfaceView;
28 import android.view.View;
29 import android.widget.Button;
30 import android.widget.FrameLayout;
31
32 // Aprasoma veikla
33 public class RealTime extends Activity {
34
35   // deklaruojamos JNI klases
36   public native String  stringFromJNI();
37   public native JavaCls jniRecognizeRT();
38
39   // Uzkraunamos JNI klases
40   static {
41     System.loadLibrary("recognize-jni");
42   }
43
44   // Mano klases
45   myFiles FS = new myFiles();
46
47   // deklaruojami globalus kintamieji
48   private Preview mPreview;
```

```
49    private DrawOnTop mDrawOnTop;
50    public Camera mCamera;
51    final int maxNumberOfFace = 1;
52    String TAG = "myCamera";
53    String timeTAG = "time";
54    int mImageWidth, mImageHeight;
55
56    // aprasomi veiksmai kurie atliekami paleidziant veikla
57    @Override
58    protected void onCreate(Bundle savedInstanceState) {
59      super.onCreate(savedInstanceState);
60
61      // sukuriama vartotojo sasaja
62      setContentView(R.layout.fotointentlayout);
63
64      // Sukuriami Preview ir DrawOnTop vaizdai
65      mDrawOnTop = new DrawOnTop(this);
66      mPreview = new Preview(this, mDrawOnTop);
67      FrameLayout preview = (FrameLayout) findViewById(R.id.
          camera_preview);
68      preview.addView(mPreview);
69      addContentView(mDrawOnTop, preview.getLayoutParams());
70
71      // susiejamas mygtukas
72      Button captureButton = (Button) findViewById(R.id.bFoto);
73      captureButton.setOnClickListener(
74        new View.OnClickListener() {
75          public void onClick(View v) {
76            // gaunama nuotrauka
77            mCamera.takePicture(null, null, mPicture);
78          }
79        }
80      );
81    }
82
83    // DrawOnTop klase -- piesiama ant kadro
84    public class DrawOnTop extends View {
85      Bitmap mBitmap;
86      Paint newPaintForFaces = new Paint();
87      byte[] mYUVData;
88      int[] mRGBData;
89
90      public DrawOnTop(Context context) {
91        super(context);
92        mBitmap = null;
93        mYUVData = null;
94        mRGBData = null;
95      }
96
97      @Override
98      protected void onDraw(Canvas canvas) {
99        if (mBitmap != null)
```

```
100         {
101           int canvasWidth = canvas.getWidth();
102           int canvasHeight = canvas.getHeight();
103
104           // Konvertuojama is YUV i RGB spalvu palete
105           decodeYUV420SP(mRGBData, mYUVData, mImageWidth,
                  mImageHeight);
106           // decodeYUV420SPGrayscale(mRGBData, mYUVData,
                  mImageWidth, mImageHeight);
107
108           /** SUKONFIGURUOJAMAS PIESIMO SLUOKSNIS VEIDU APTIKIMUI
                  */
109           newPaintForFaces.setStyle(Paint.Style.FILL);
110           newPaintForFaces.setColor(Color.WHITE);
111           newPaintForFaces.setTextSize(30);
112           newPaintForFaces.setStrokeWidth(3);
113
114           /** Veidu aptikimas */
115           int numberOfFaceDetected;
116           FaceDetector myFaceDetectArray;
117           FaceDetector.Face myFaces[] = new FaceDetector.Face[
                  maxNumberOfFace];
118           FaceDetector.Face currentFace = null;
119           PointF myEyeMidPoint[] = new PointF[maxNumberOfFace];
120           float myEyesDistance[] = new float[maxNumberOfFace];
121           float myConfidence[] = new float[maxNumberOfFace];
122           float ratioX = (float)canvasWidth / (float)mImageWidth;
123           float ratioY = (float) canvasHeight / (float)mImageHeight
                  ;
124           Bitmap bmpFaces;
125
126           // sukuriamas piesinys
127           bmpFaces = Bitmap.createBitmap(mRGBData, mImageWidth,
                  mImageHeight, Bitmap.Config.RGB_565);
128
129           // aptinkami veidai
130           myFaceDetectArray = new FaceDetector(mImageWidth,
                  mImageHeight, maxNumberOfFace);
131           numberOfFaceDetected = myFaceDetectArray.findFaces(
                  bmpFaces, myFaces);
132           bmpFaces.recycle();
133
134           if (numberOfFaceDetected > 0) {
135             Log.v(TAG, "Nuber of detected faces: " +
                    numberOfFaceDetected);
136           }
137
138
139           /** Suzymimi veidai **/
140           // i-tasis veidas
141           for(int i=0; i < numberOfFaceDetected; i++)
142           {
```

```
143             currentFace = myFaces[i];
144
145             PointF myEyeMP = new PointF();
146             currentFace.getMidPoint(myEyeMP);
147
148             myEyeMidPoint[i]  = myEyeMP;
149             myEyesDistance[i] = currentFace.eyesDistance();
150             myConfidence[i]   = currentFace.confidence();
151
152             // pazymimas tarpuakis
153             canvas.drawCircle( (float)(myEyeMidPoint[i].x * ratioX)
                    , (float)(myEyeMidPoint[i].y * ratioY), 10,
                    newPaintForFaces);
154
155             // nupiesiamas kvadratas aplink veida
156             float distX = myEyesDistance[i];
157             float distY = myEyesDistance[i]*2;
158             // Kaire linija
159             canvas.drawLine((myEyeMidPoint[i].x-distX)*ratioX, (
                    myEyeMidPoint[i].y-distY/2)*ratioY,
160                     (myEyeMidPoint[i].x-distX)*ratioX, (
                            myEyeMidPoint[i].y+distY)*ratioY,
161                     newPaintForFaces);
162             // Desine linija
163             canvas.drawLine((myEyeMidPoint[i].x+distX)*ratioX, (
                    myEyeMidPoint[i].y-distY/2)*ratioY,
164                     (myEyeMidPoint[i].x+distX)*ratioX, (
                            myEyeMidPoint[i].y+distY)*ratioY,
165                     newPaintForFaces);
166             // Virsutine linija
167             canvas.drawLine((myEyeMidPoint[i].x-distX)*ratioX, (
                    myEyeMidPoint[i].y-distY/2)*ratioY,
168                     (myEyeMidPoint[i].x+distX)*ratioX, (
                            myEyeMidPoint[i].y-distY/2)*ratioY,
169                     newPaintForFaces);
170             // Apatine linija
171             canvas.drawLine((myEyeMidPoint[i].x-distX)*ratioX, (
                    myEyeMidPoint[i].y+distY)*ratioY,
172                     (myEyeMidPoint[i].x+distX)*ratioX, (
                            myEyeMidPoint[i].y+distY)*ratioY,
173                     newPaintForFaces);
174         }
175
176     /* ********************************************************
177      * Is kadro iskerpa aptikta veida kaip int[] ir jei reikia *
178      * ji issaugo jpg formatu. Nuotraukos virsus ir apacia yra *
179      * nukerpami System.arraycopy() funkcija. Sonai nukerpami  *
180      * ciklo pagalba                                           *
181      ********************************************************/
182         int faceID = 0;
183         double faceDist = 0;
184         float faceConf = 0;
```

```
185         String vardas = null;
186
187         // Matuojama paruosimo trukme: pradzios laikas
188         long cropStart = System.currentTimeMillis();
189
190         // iskerpamas veidas
191         if(numberOfFaceDetected==1) {
192           int y = 320;
193           int x = 240;
194           int myW = mImageWidth;
195           int myH = mImageHeight;
196
197           // Apskaiciuojamos veido koordinates
198           int yTop  = (int)(myEyeMidPoint[0].y - myEyesDistance
                  [0]);
199           int yBottom = (int)(myEyeMidPoint[0].y + 2 *
                  myEyesDistance[0]);
200           int xLeft = (int)(myEyeMidPoint[0].x - myEyesDistance
                  [0]);
201           int xRight  = (int)(myEyeMidPoint[0].x + myEyesDistance
                  [0]);
202
203           // patikrinama ar neiseinama uz ribu
204           if ( yTop < 0) yTop = 0;
205           if ( yBottom > myH) yBottom = myH-1;
206           if ( xLeft < 0) xLeft = 0;
207           if ( xRight > myW) xRight = myW-1;
208
209           // Is nuotraukos iskerpamas veidas
210           // nukerpamas nuotraukos virus ir apacia
211           int[] myRGBimage_temp = new int[(yBottom-yTop+1)*y];
212           System.arraycopy(mRGBData, yTop*y, myRGBimage_temp, 0,
                  myRGBimage_temp.length);
213
214           int[] myRGB = new int[(yBottom-yTop+1)*(xRight-xLeft+1)
                  ];
215           int srcStart, dstStart, dstLength;
216           srcStart = xLeft;
217           dstLength = xRight - xLeft + 1;
218           dstStart = 0;
219           for (int i = 0 ; i < yBottom-yTop+1; i++) {
220             System.arraycopy(myRGBimage_temp, srcStart, myRGB,
                    dstStart, dstLength);
221             dstStart = dstStart + dstLength;
222             srcStart = srcStart + y;
223           }
224           myRGBimage_temp = null; // panaikinamas kintamasis
225
226           // sukuriamas naujas paveikslas
227           Bitmap RGB_bitmap = Bitmap.createBitmap( myRGB, (xRight
                  -xLeft+1), (yBottom-yTop+1), Config.RGB_565);
228
```

```
229             // pakeiciama jo raiska
230             Bitmap mBitmapResizedFace;
231             mBitmapResizedFace = getResizedBitmap(RGB_bitmap, 90,
                    60);
232             RGB_bitmap.recycle();
233
234             // paveikslas issaugojamas
235             try {
236               FileOutputStream out = new FileOutputStream("/sdcard/
                      MyCameraApp/LastFaceRT.jpg");
237               mBitmapResizedFace.compress(Bitmap.CompressFormat.
                      JPEG, 90, out);
238             } catch (Exception e) {
239               e.printStackTrace();
240             }
241             mBitmapResizedFace.recycle();
242
243             long cropTime = System.currentTimeMillis() - cropStart;
                    // Matuojama paruosimo trukme: trukme
244             Log.v(timeTAG, "Paruosimas uztruko " + cropTime + " ms"
                    );
245
246             // ATPAZISTAMAS VEIDAS
247             long recStart = System.currentTimeMillis(); //
                    Matuojama atpazinimo trukme: pradzios laikas
248             JavaCls faceResult = jniRecognizeRT(); // iskvieciama
                    native atpazinimo f-ja
249             faceID = faceResult.faceID; // atsakymas = ID
250             faceDist = faceResult.dist; // atsakymas = atstumas
251             faceConf = faceResult.conf; // atsakymas = patikimumas
252             Log.v(TAG, "ID: "+faceID+" Dist: "+faceDist+" Conf: "+
                    faceConf);
253             faceResult=null;
254
255             // Gaunamas vardas pagal ID
256             File f = new File(Environment.
                    getExternalStorageDirectory()+File.separator+"
                    MyCameraApp"+File.separator+faceID);
257             String[] filenames = f.list();
258             for (int i=0; i<filenames.length; i++) {
259               if (filenames[i].endsWith(".id")) {
260                 vardas = filenames[i].substring(0, filenames[i].
                        length() - 3);
261                 break;
262               }
263             }
264             long recTime = System.currentTimeMillis() - recStart;
                    // Matuojama atpazinimo trukme: trukme
265             Log.v(timeTAG, "Atpzinimas uztruko " + recTime + " ms")
                    ;
266
267           }
```

```
268
269
270
271             /** ISVEDAMA INFORMACIJA I EKRANA */
272             canvas.drawText("Aptiktu veidu skaicius: "+
                    numberOfFaceDetected, 10, 50, newPaintForFaces);
273             canvas.drawText("Veido patikimumas: "+myConfidence[0],
                    10, 80, newPaintForFaces);
274
275             if (faceID==0){
276               canvas.drawText("Atpazintas ID: null", 10, 370,
                      newPaintForFaces);
277             }else{
278               canvas.drawText("Atpazintas ID: "+faceID, 10, 370,
                      newPaintForFaces);
279             }
280             if (faceConf >= 0.4) {
281               canvas.drawText("Vardas: " + vardas, 10, 400,
                      newPaintForFaces);
282             }
283             else if (faceConf > 0) {
284               canvas.drawText("Vardas: NEZINOMAS (galbut " + vardas +
                      ")", 10, 400, newPaintForFaces);
285             }
286             else {
287               canvas.drawText("Vardas: null", 10, 400,
                      newPaintForFaces);
288             }
289             if (faceDist == 0 && faceConf == 0) {
290               canvas.drawText("Atstumas:    null", 10, 430,
                      newPaintForFaces);
291               canvas.drawText("Patikimumas:  null", 10, 460,
                      newPaintForFaces);
292             }
293             else {
294               canvas.drawText("Atstumas:    " + String.format("%.6f",
                      faceDist), 10, 430, newPaintForFaces);
295               canvas.drawText("Patikimumas:  " + String.format("%.2f
                      %%", faceConf*100), 10, 460, newPaintForFaces);
296             }
297           } // end if statement
298           super.onDraw(canvas);
299         } // end onDraw method
300       }
301
302       // dekoduoja YUV420SP spalvu palete i RGB palete
303       public void decodeYUV420SP(int[] rgb, byte[] yuv420sp, int
              width, int height) {
304         final int frameSize = width * height;
305
306         for (int j = 0, yp = 0; j < height; j++) {
307           int uvp = frameSize + (j >> 1) * width, u = 0, v = 0;
```

```
308         for (int i = 0; i < width; i++, yp++) {
309           int y = (0xff & ((int) yuv420sp[yp])) - 16;
310           if (y < 0) y = 0;
311           if ((i & 1) == 0) {
312             v = (0xff & yuv420sp[uvp++]) - 128;
313             u = (0xff & yuv420sp[uvp++]) - 128;
314           }
315
316           int y1192 = 1192 * y;
317           int r = (y1192 + 1634 * v);
318           int g = (y1192 - 833 * v - 400 * u);
319           int b = (y1192 + 2066 * u);
320
321           if (r < 0) r = 0; else if (r > 262143) r = 262143;
322           if (g < 0) g = 0; else if (g > 262143) g = 262143;
323           if (b < 0) b = 0; else if (b > 262143) b = 262143;
324
325           rgb[yp] = 0xff000000 | ((r << 6) & 0xff0000) | ((g >> 2)
                & 0xff00) | ((b >> 10) & 0xff);
326         }
327       }
328     }
329
330     // dekoduoja YUV420SP spalvu palete i pilka palete
331     public void decodeYUV420SPGrayscale(int[] rgb, byte[] yuv420sp,
          int width, int height)
332     {
333       final int frameSize = width * height;
334
335       for (int pix = 0; pix < frameSize; pix++)
336       {
337         int pixVal = (0xff & ((int) yuv420sp[pix])) - 16;
338         if (pixVal < 0) pixVal = 0;
339         if (pixVal > 255) pixVal = 255;
340         rgb[pix] = 0xff000000 | (pixVal << 16) | (pixVal << 8) |
              pixVal;
341       } // pix
342     }
343
344     // pakeicia paveikslo raiska
345     public Bitmap getResizedBitmap(Bitmap bm, int newHeight, int
          newWidth) {
346
347       // gaunas originalios nuotraukos ismatavimus
348       int width = bm.getWidth();
349       int height = bm.getHeight();
350
351       // apskaiciuojam santiki naujas dydis / senas dydis
352       float scaleWidth = ((float) newWidth) / width;
353       float scaleHeight = ((float) newHeight) / height;
354
355       // Sukuriama matrica skirta manipuliacijoms
```

```
356       Matrix matrix = new Matrix();
357
358       // Pakeiciama raiska
359       matrix.postScale(scaleWidth, scaleHeight);
360
361       // Atkuriamas paveikslas
362       Bitmap resizedBitmap = Bitmap.createBitmap(bm, 0, 0, width,
              height, matrix, false);
363       return resizedBitmap;
364     }
365
366     // kadru atvaizdavimas
367     public class Preview extends SurfaceView implements
          SurfaceHolder.Callback {
368       SurfaceHolder mHolder;
369       DrawOnTop mDrawOnTop;
370       boolean mFinished;
371
372       Preview(Context context, DrawOnTop drawOnTop) {
373         super(context);
374
375         mDrawOnTop = drawOnTop;
376         mFinished = false;
377
378         // Idiegiamas SurfaceHolder.Callback jog mes galetumem
379         // zinoti kada pavirsius yra sukuriamas ar sunaikinamas
380         mHolder = getHolder();
381         mHolder.addCallback(this);
382         mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
383       }
384
385       public void surfaceCreated(SurfaceHolder holder) {
386         mCamera = Camera.open();
387         try {
388           mCamera.setPreviewDisplay(holder);
389
390           // Preview callback naudojamas kai tik yra prieinamas
                naujas kadras
391           mCamera.setPreviewCallback(new PreviewCallback() {
392             @SuppressWarnings("unused")
393             int H, W;
394             public void onPreviewFrame(byte[] data, Camera camera)
395             {
396               if ( (mDrawOnTop == null) || mFinished )
397                 return;
398               if (mDrawOnTop.mBitmap == null)
399               {
400                 // pagal kameros parametrus
401                 // inicializuojamas drawOnTop
402                 Camera.Parameters params = camera.getParameters();
403                 mImageWidth = params.getPreviewSize().width;
404                 mImageHeight = params.getPreviewSize().height;
```

```
405                 mDrawOnTop.mBitmap = Bitmap.createBitmap(
                        mImageWidth, mImageHeight, Bitmap.Config.RGB_565
                        );
406                 mDrawOnTop.mRGBData = new int[mImageWidth *
                        mImageHeight];
407                 mDrawOnTop.mYUVData = new byte[data.length];
408                 H = params.getPictureSize().height;
409                 W = params.getPictureSize().width;
410               }
411               // YUV duomenys perduodami i drawOnTop
412               System.arraycopy(data, 0, mDrawOnTop.mYUVData, 0,
                      data.length);
413               mDrawOnTop.invalidate();
414             }
415           });
416           Camera.Parameters parameters = mCamera.getParameters();
417           android.hardware.Camera.Size previewSize = parameters.
                  getPreviewSize();
418           android.hardware.Camera.Size pictureSize = parameters.
                  getPictureSize();
419           android.hardware.Camera.Size JPEGThumbSize = parameters.
                  getJpegThumbnailSize();
420         }
421         catch (IOException exception) {
422           mCamera.release();
423           mCamera = null;
424         }
425       }
426
427       // Pabaigus darba atlaisvinama kamera
428       public void surfaceDestroyed(SurfaceHolder holder) {
429         mFinished = true;
430         mCamera.setPreviewCallback(null);
431         mCamera.stopPreview();
432         mCamera.release();
433         mCamera = null;
434       }
435
436       // nustatomi kameros parametrai
437       public void surfaceChanged(SurfaceHolder holder, int format,
              int w, int h) {
438         Camera.Parameters parameters = mCamera.getParameters();
439         parameters.setPreviewSize(320, 240);
440         parameters.setPreviewFrameRate(5);
441         parameters.setSceneMode(Camera.Parameters.SCENE_MODE_AUTO);
442         parameters.setFocusMode(Camera.Parameters.FOCUS_MODE_AUTO);
443         parameters.setFlashMode(Camera.Parameters.FLASH_MODE_AUTO);
444         parameters.setWhiteBalance(Camera.Parameters.
                WHITE_BALANCE_AUTO);
445
446         mCamera.setParameters(parameters);
447         mCamera.startPreview();
```

```
448       }
449     }
450
451     // nuotraukos gavimas
452     private PictureCallback mPicture = new PictureCallback() {
453       public void onPictureTaken(byte[] data, Camera camera) {
454
455         Log.v(TAG, "Picture taken...");
456         File pictureFile = FotoIntentActivity.getOutputMediaFile();
457         if (pictureFile == null){
458           Log.d(TAG, "Error creating media file, check storage
                permissions: "); // + e.getMessage());
459           return;
460         }
461         else {
462           Log.d(TAG, "pictureFile created successfully");
463         }
464
465         // issaugomas paveikslas
466         try {
467           FileOutputStream fos = new FileOutputStream(pictureFile);
468           fos.write(data);
469           fos.close();
470         } catch (FileNotFoundException e) {
471           Log.d(TAG, "File not found: " + e.getMessage());
472         } catch (IOException e) {
473           Log.d(TAG, "Error accessing file: " + e.getMessage());
474         }
475
476         mCamera.setPreviewCallback(null);
477
478         // Nutraukia activity ir gryzta atgal po 2 sekundziu
479         try {
480           setResult(Activity.RESULT_OK);
481           Log.v(TAG, "RESULT_OK");
482           Thread.sleep(2000);
483           finish();
484
485         } catch (Exception e) {
486           e.getLocalizedMessage();
487         }
488       }
489     };
490
491 }
```

SRC/COM/BALINSKAS/FOTOINTENT/MYCAMERAINTENT.JAV

```
1  package com.balinskas.fotointent;
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.FileOutputStream;
6  import java.io.IOException;
7
8  import android.app.Activity;
9  import android.content.Context;
10 import android.graphics.Bitmap;
11 import android.graphics.Canvas;
12 import android.graphics.Color;
13 import android.graphics.Paint;
14 import android.graphics.PointF;
15 import android.hardware.Camera;
16 import android.hardware.Camera.PictureCallback;
17 import android.hardware.Camera.PreviewCallback;
18 import android.media.FaceDetector;
19 import android.os.Bundle;
20 import android.util.Log;
21 import android.view.SurfaceHolder;
22 import android.view.SurfaceView;
23 import android.view.View;
24 import android.widget.Button;
25 import android.widget.FrameLayout;
26
27   // Aprasoma veikla
28 public class myCameraIntent extends Activity {
29
30   private Preview mPreview;
31   private DrawOnTop mDrawOnTop;
32   public Camera mCamera;
33   String TAG = "myCamera";
34
35   // aprasomi veiksmai kurie atliekami paleidziant veikla
36   @Override
37   protected void onCreate(Bundle savedInstanceState) {
38     super.onCreate(savedInstanceState);
39
40     // sukuriama vartotojo sasaja
41     setContentView(R.layout.fotointentlayout);
42
43     // Sukuriami Preview ir DrawOnTop vaizdai
44     mDrawOnTop = new DrawOnTop(this);
45     mPreview = new Preview(this, mDrawOnTop);
46     FrameLayout preview = (FrameLayout) findViewById(R.id.
         camera_preview);
47     preview.addView(mPreview);
```

```
48      addContentView(mDrawOnTop, preview.getLayoutParams());
49
50      // susiejamas mygtukas
51      Button captureButton = (Button) findViewById(R.id.bFoto);
52      captureButton.setOnClickListener(
53        new View.OnClickListener() {
54          public void onClick(View v) {
55            // gaunama nuotrauka
56            mCamera.takePicture(null, null, mPicture);
57          }
58        }
59      );
60
61    }
62
63    // DrawOnTop klase -- piesiama ant kadro
64    public class DrawOnTop extends View {
65      Bitmap mBitmap;
66      Paint newPaintForFaces = new Paint();
67      byte[] mYUVData;
68      int[] mRGBData;
69      int mImageWidth, mImageHeight;
70
71      public DrawOnTop(Context context) {
72        super(context);
73        mBitmap = null;
74        mYUVData = null;
75        mRGBData = null;
76      }
77
78      @Override
79      protected void onDraw(Canvas canvas) {
80        if (mBitmap != null)
81        {
82          int canvasWidth = canvas.getWidth();
83          int canvasHeight = canvas.getHeight();
84
85          // Konvertuojama is YUV i RGB spalvu palete
86          decodeYUV420SP(mRGBData, mYUVData, mImageWidth,
                mImageHeight);
87          // decodeYUV420SPGrayscale(mRGBData, mYUVData,
                mImageWidth, mImageHeight);
88
89          /** SUKONFIGURUOJAMAS PIESIMO SLUOKSNIS VEIDU APTIKIMUI
                */
90          newPaintForFaces.setStyle(Paint.Style.FILL);
91          newPaintForFaces.setColor(Color.WHITE);
92          newPaintForFaces.setTextSize(30);
93          newPaintForFaces.setStrokeWidth(3);
94
95          /** DETECT FACES */
96          final int maxNumberOfFace = 6;
```

```
97          int numberOfFaceDetected;
98          FaceDetector myFaceDetectArray;
99          FaceDetector.Face myFaces[] = new FaceDetector.Face[
                maxNumberOfFace];
100         FaceDetector.Face currentFace = null;
101         PointF myEyeMidPoint[]  = new PointF[maxNumberOfFace];
102         float myEyesDistance[]  = new float[maxNumberOfFace];
103         float myConfidence[]  = new float[maxNumberOfFace];
104         float ratioX = (float)canvasWidth  / (float)mImageWidth;
105         float ratioY =(float) canvasHeight / (float)mImageHeight;
106
107         // sukuriamas piesinys
108         Bitmap bmpFaces;
109         bmpFaces = Bitmap.createBitmap(mRGBData, mImageWidth,
                mImageHeight, Bitmap.Config.RGB_565);
110
111         // Aptinkami veidai
112         myFaceDetectArray = new FaceDetector(mImageWidth,
                mImageHeight, maxNumberOfFace);
113         numberOfFaceDetected = myFaceDetectArray.findFaces(
                bmpFaces, myFaces);
114
115         if (numberOfFaceDetected > 0) {
116           Log.v(TAG, "Nuber of detected faces: " +
                  numberOfFaceDetected);
117         }
118
119
120         /** Suzymimi veidai **/
121         // i-tasis veidas
122         for(int i=0; i < numberOfFaceDetected; i++)
123         {
124           currentFace = myFaces[i];
125
126           PointF myEyeMP = new PointF();
127           currentFace.getMidPoint(myEyeMP);
128
129           myEyeMidPoint[i] = myEyeMP;
130           myEyesDistance[i] = currentFace.eyesDistance();
131           myConfidence[i] = currentFace.confidence();
132
133           Log.v(TAG, "Face[" + i + "] ---"+ "EyeMidPoint(x=" +
                  myEyeMidPoint[i].x + ",y=" + myEyeMidPoint[i].y + ")
                  myEyesDist(" + myEyesDistance[i] + ") conf(" +
                  myConfidence[i] + ")");
134
135
136           // ir jo viduryje parasomas jo numeris
137           canvas.drawCircle( (float)(myEyeMidPoint[i].x * ratioX)
                  , (float)(myEyeMidPoint[i].y * ratioY), 10,
                  newPaintForFaces);
138
```

```
139             float distX = myEyesDistance[i];
140             float distY = myEyesDistance[i]*2;
141             // Kaire linija
142             canvas.drawLine((myEyeMidPoint[i].x-distX)*ratioX, (
                    myEyeMidPoint[i].y-distY/2)*ratioY,
143                     (myEyeMidPoint[i].x-distX)*ratioX, (
                            myEyeMidPoint[i].y+distY)*ratioY,
144                     newPaintForFaces);
145             // Desine linija
146             canvas.drawLine((myEyeMidPoint[i].x+distX)*ratioX, (
                    myEyeMidPoint[i].y-distY/2)*ratioY,
147                     (myEyeMidPoint[i].x+distX)*ratioX, (
                            myEyeMidPoint[i].y+distY)*ratioY,
148                     newPaintForFaces);
149             // Virsutine linija
150             canvas.drawLine((myEyeMidPoint[i].x-distX)*ratioX, (
                    myEyeMidPoint[i].y-distY/2)*ratioY,
151                     (myEyeMidPoint[i].x+distX)*ratioX, (
                            myEyeMidPoint[i].y-distY/2)*ratioY,
152                     newPaintForFaces);
153             // Apatine linija
154             canvas.drawLine((myEyeMidPoint[i].x-distX)*ratioX, (
                    myEyeMidPoint[i].y+distY)*ratioY,
155                     (myEyeMidPoint[i].x+distX)*ratioX, (
                            myEyeMidPoint[i].y+distY)*ratioY,
156                     newPaintForFaces);
157         }
158
159         /** I EKRANA ISVEDAMA INFORMACIJA APIE APTIKTUS VEIDUS */
160         canvas.drawText("Aptiku veidu skaicius: "+
                numberOfFaceDetected, 10, 50, newPaintForFaces);
161
162       } // end if statement
163       super.onDraw(canvas);
164     } // end onDraw method
165
166     // dekoduoja YUV420SP spalvu palete i RGB palete
167     public void decodeYUV420SP(int[] rgb, byte[] yuv420sp, int
            width, int height) {
168       final int frameSize = width * height;
169
170       for (int j = 0, yp = 0; j < height; j++) {
171         int uvp = frameSize + (j >> 1) * width, u = 0, v = 0;
172         for (int i = 0; i < width; i++, yp++) {
173           int y = (0xff & ((int) yuv420sp[yp])) - 16;
174           if (y < 0) y = 0;
175           if ((i & 1) == 0) {
176             v = (0xff & yuv420sp[uvp++]) - 128;
177             u = (0xff & yuv420sp[uvp++]) - 128;
178           }
179
180           int y1192 = 1192 * y;
```

```
181             int r = (y1192 + 1634 * v);
182             int g = (y1192 - 833 * v - 400 * u);
183             int b = (y1192 + 2066 * u);
184 
185             if (r < 0) r = 0; else if (r > 262143) r = 262143;
186             if (g < 0) g = 0; else if (g > 262143) g = 262143;
187             if (b < 0) b = 0; else if (b > 262143) b = 262143;
188 
189             rgb[yp] = 0xff000000 | ((r << 6) & 0xff0000) | ((g >>
                    2) & 0xff00) | ((b >> 10) & 0xff);
190           }
191         }
192       }
193 
194       // dekoduoja YUV420SP spalvu palete i pilka palete
195       public void decodeYUV420SPGrayscale(int[] rgb, byte[]
             yuv420sp, int width, int height)
196       {
197         final int frameSize = width * height;
198 
199         for (int pix = 0; pix < frameSize; pix++)
200         {
201           int pixVal = (0xff & ((int) yuv420sp[pix])) - 16;
202           if (pixVal < 0) pixVal = 0;
203           if (pixVal > 255) pixVal = 255;
204           rgb[pix] = 0xff000000 | (pixVal << 16) | (pixVal << 8) |
                  pixVal;
205         } // pix
206       }
207 
208     }
209 
210     public class Preview extends SurfaceView implements
           SurfaceHolder.Callback {
211       SurfaceHolder mHolder;
212       DrawOnTop mDrawOnTop;
213       boolean mFinished;
214 
215       // kadru atvaizdavimas
216       Preview(Context context, DrawOnTop drawOnTop) {
217         super(context);
218 
219         mDrawOnTop = drawOnTop;
220         mFinished = false;
221 
222         // Idiegiamas SurfaceHolder.Callback jog mes galetumem
223         // zinoti kada pavirsius yra sukuriamas ar sunaikinamas
224         mHolder = getHolder();
225         mHolder.addCallback(this);
226         mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
227       }
228 
```

```
229     public void surfaceCreated(SurfaceHolder holder) {
230       mCamera = Camera.open();
231       try {
232         mCamera.setPreviewDisplay(holder);
233
234         // Preview callback naudojamas kai tik yra prieinamas
              naujas kadras
235         mCamera.setPreviewCallback(new PreviewCallback() {
236           public void onPreviewFrame(byte[] data, Camera camera)
237           {
238             if ( (mDrawOnTop == null) || mFinished )
239               return;
240
241             if (mDrawOnTop.mBitmap == null)
242             {
243               // pagal kameros parametrus
244               // inicializuojamas drawOnTop
245               Camera.Parameters params = camera.getParameters();
246               mDrawOnTop.mImageWidth = params.getPreviewSize().
                    width;
247               mDrawOnTop.mImageHeight = params.getPreviewSize().
                    height;
248               mDrawOnTop.mBitmap = Bitmap.createBitmap(mDrawOnTop
                    .mImageWidth,
249               mDrawOnTop.mImageHeight, Bitmap.Config.RGB_565);
250               mDrawOnTop.mRGBData = new int[mDrawOnTop.
                    mImageWidth * mDrawOnTop.mImageHeight];
251               mDrawOnTop.mYUVData = new byte[data.length];
252             }
253
254             // YUV duomenys perduodami i drawOnTop
255             System.arraycopy(data, 0, mDrawOnTop.mYUVData, 0,
                  data.length);
256             mDrawOnTop.invalidate();
257           }
258         });
259
260         Camera.Parameters parameters = mCamera.getParameters();
261         android.hardware.Camera.Size previewSize = parameters.
              getPreviewSize();
262         android.hardware.Camera.Size pictureSize = parameters.
              getPictureSize();
263         android.hardware.Camera.Size JPEGThumbSize = parameters.
              getJpegThumbnailSize();
264       }
265       catch (IOException exception) {
266         mCamera.release();
267         mCamera = null;
268       }
269     }
270
271     // Pabaigus darba atlaisvinama kamera
```

```
272     public void surfaceDestroyed(SurfaceHolder holder) {
273       mFinished = true;
274       mCamera.setPreviewCallback(null);
275       mCamera.stopPreview();
276       mCamera.release();
277       mCamera = null;
278     }
279
280     // nustatomi kameros parametrai
281     public void surfaceChanged(SurfaceHolder holder, int format,
          int w, int h) {
282       Camera.Parameters parameters = mCamera.getParameters();
283       parameters.setPreviewSize(320, 240);
284       parameters.setPreviewFrameRate(5);
285       parameters.setSceneMode(Camera.Parameters.SCENE_MODE_AUTO);
286       parameters.setFocusMode(Camera.Parameters.FOCUS_MODE_AUTO);
287       parameters.setFlashMode(Camera.Parameters.FLASH_MODE_AUTO);
288       parameters.setWhiteBalance(Camera.Parameters.
            WHITE_BALANCE_AUTO);
289
290       mCamera.setParameters(parameters);
291       mCamera.startPreview();
292     }
293
294   }
295
296   // nuotraukos gavimas
297   private PictureCallback mPicture = new PictureCallback() {
298     public void onPictureTaken(byte[] data, Camera camera) {
299
300       Log.v(TAG, "Picture taken...");
301
302
303       File pictureFile = FotoIntentActivity.getOutputMediaFile();
304       if (pictureFile == null){
305         Log.d(TAG, "Error creating media file, check storage
              permissions: "); // + e.getMessage());
306         return;
307       }
308       else {
309         Log.d(TAG, "pictureFile created successfully");
310       }
311
312       // issaugomas paveikslas
313       try {
314         FileOutputStream fos = new FileOutputStream(pictureFile);
315         fos.write(data);
316         fos.close();
317       } catch (FileNotFoundException e) {
318         Log.d(TAG, "File not found: " + e.getMessage());
319       } catch (IOException e) {
320         Log.d(TAG, "Error accessing file: " + e.getMessage());
```

```
321         }
322
323         mCamera.setPreviewCallback(null);
324
325         // Nutraukia activity ir gryzta atgal po 2 sekundziu
326         try {
327           setResult(Activity.RESULT_OK);
328           Log.v(TAG, "RESULT_OK");
329           Thread.sleep(2000);
330           finish();
331
332         } catch (Exception e) {
333           e.getLocalizedMessage();
334         }
335       }
336     };
337
338 }
```

F. KOMPAKTINĖ PLOKŠTELĖ

- **Išeities kodai:**
  - *MATLAB*
  - *Android SDK*
- **Elektroninė ataskaitos versija**