Turinys

[Turinys 1](#_Toc483538517)

[1. Užduotis 2](#_Toc483538518)

[2. Bag of features 2](#_Toc483538519)

[3. Histogram of oriented gradients 2](#_Toc483538520)

[4. SIFT (Scale-invariant feature transform) 2](#_Toc483538521)

[5. Tyrimas 3](#_Toc483538522)

[5.1 60-40 3](#_Toc483538523)

[5.2 50-50 4](#_Toc483538524)

[5.3 70-30 5](#_Toc483538525)

[5.4 70-30 6](#_Toc483538526)

[5.5 60-40 7](#_Toc483538527)

[5.6 50-50 7](#_Toc483538528)

[5.7 50-50 SIFT 7](#_Toc483538529)

[5.8 60-40 SIFT 8](#_Toc483538530)

[5.9 70-30 SIFT 9](#_Toc483538531)

[5.10 Grafikai 9](#_Toc483538532)

[6. Kodas 10](#_Toc483538533)

[7. Išvados 19](#_Toc483538534)

# Užduotis

Trimis skirtingais algoritmais : Bag of features, Histogram of Oriented gradients ir SIFT, atpažinti lietuviškas raides rašytas ranka. Duomenų kiekis naudojamas apmokymui ~ 150 raidžių nuotraukų. Ištirti, kuris algoritmas atpažįsta tiksliausiai ranka parašytas lietuviškas raides.

Kiekvienas studentas pasirinko skirtingą algoritmą:

Miglė Beresinevičiūtė: SIFT,

Rokas Vaitkevičius: Bag of Features,

Dovydas Petrutis: Hostogram of Oriented Gradients

Visi algoritmai naudojo tokius pačiu apmokymo ir analizės duomenis.

# Bag of features

Šio algoritmo esmė paveiksliukuose rasti kažkokius pasikartojančius bruožus ir paskui juos naudojant identifikuoti ar tai yra ieškomas objektas naujose nuotraukose. Algoritmas visų pirma suranda bruožus, tada juos struktūrizuoja sukurdamas tarsi žodyną. Kiekvienai kategorijai ieškoma stipriausių t.y. dažniausiai pasikartojančių bruožų, tik jie ir imami tolesnei nuotraukų analizei. Kai algoritmas apmokomas, jam paduodamos naujos nuotraukos ir jis pagal atrinktus bruožus nusprendžia, kokiai kategorijai priklauso tas paveiksliukas.

# Histogram of oriented gradients

Histogram of oriented gradients algoritmas yra naudojamas vaizdų atpažinimui kompiuteryje tam, kad būtų aptikti norimi objektai.

Algoritmas skaičiuoja orientuotų gradientų paplitimą lokalizuotoje vaizdo dalyje. Algoritmas skaičiuoja kaip kreipiasi vaizde esantys objektai

pasirinktu rėmo dydžiu, taip piešdamas vaizdinį gali pamatyti jo kontūrus. Vaizdas būna sudalintas į pikselius.

Pirmame žingsnyje algoritmas susivienodina paveiksliukų dydžius bei spalvas, dažniausiai pasiverčia į juodai baltus. Dažniausiai naudojamas gradientų skaičiavimas 1-D

Gradientai gali būti [-1 0 1] taip pat tokia pat matrica tik transformuota.

Antrame žingsnyje yra kuriama histograma. Kiekvienas pikselis turi savo svorį ir gali kreiptis 180 laipsniu kampu. Kad pamatyti pasikeitimus vaizdas suskaidomas į blokus, kurie gali būti 2x2 4x4 8x8 ar didesnio dydžio kuo mažesnis blokas, tuo didesnis tikslumas pasiekiamas.

# SIFT (Scale-invariant feature transform)

SIFT algoritmas yra naudojamas nuotraukos savybėms išgauti. Iš analizavus kiekvieną nuotrauką su SIFT algoritmu galime gauti savybės deskriptorius ir savybės detektorius. Detektorius iš nuotraukos suranda visus rėmus (apšvietimo ir žiūrėjimo kampų). Deskriptorius – identifikuoja stiprių savybių išvaizdą.

SIFT algoritmas turi parametrus:

peak\_thresh = 0; numatytoji reikšmė

edge\_thresh = 10; numatytoji reikšmė

Abudu šie parametrai padeda nustatyti kaip tiksliai bus ieškomos nuotraukos savybės. Peak reikšmė visada didėja, o edge mažėja.

Dviejų nuotraukų palyginimui naudojamas deskriptorių palyginimas. Kiek vienodų savybių buvo rasti nuotraukose. Algoritmas ieško panašių savybių. **VL\_UBCMATCH** metodui reikalingas ir trečias parametras *thresh. Thresh* parametras nustato ir randa visas vienodas savybes, jei atstumas tarp dviejų deskriptorių ir padauginus iš thresh skaičiau yra ne aukštesnis už pirmosios nuotraukos deskriptoriaus atstumui iki kitų deskriptoriaus taškų.

Dauguma surastų savybių nėra tikros, todėl pasitelkus *Houht transformacija*. Randa visas labiausiai atitikusias savybes iš visų pusių (pasukimo kampų). Didžiausia balsų turinčios pozicijos dažniausiai būna teisingos abejoms nuotraukoms. Tokiu būdų išrenkamos tik svarbiausios savybės. Tikslesnis būdas nustatyti kokia tai buvo raidė.

Lyginamos nuotraukos turi būti vienodo didžio.

# Tyrimas

## 60-40

Testavimui naudojau bendrą raidžių katalogą. Maždaug 60% kiekvienos kategorijos raidžių buvo naudotos algoritmo apmokymui, o likę 40% analizei. Testavimo duomenų rinkinys: 259 raidžių nuotraukos. Iš kurių 155 naudotos apmokyti algoritmą, kitos 104 jį testuoti,. Su šiais duomenimis gaunami tokie rezultatai:

Testavimas su apmokymo rinkiniu:

PREDICTED

KNOWN | ą č ė ę į š ū ų ž

-------------------------------------------------------------------------

ą | 0.67 0.17 0.00 0.17 0.00 0.00 0.00 0.00 0.00

č | 0.08 0.75 0.00 0.17 0.00 0.00 0.00 0.00 0.00

ė | 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00

ę | 0.08 0.08 0.00 0.83 0.00 0.00 0.00 0.00 0.00

į | 0.00 0.00 0.00 0.00 0.94 0.00 0.00 0.00 0.06

š | 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00

ū | 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00

ų | 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00

ž | 0.00 0.00 0.00 0.00 0.14 0.00 0.00 0.00 0.86

\* Average Accuracy is 0.89.

Vidutinis tikslumas gautas 89%.

Testavimas su naujų raidžių rinkiniu:

PREDICTED

KNOWN | ą č ė ę į š ū ų ž

-------------------------------------------------------------------------

ą | 0.50 0.13 0.00 0.38 0.00 0.00 0.00 0.00 0.00

č | 0.25 0.13 0.00 0.63 0.00 0.00 0.00 0.00 0.00

ė | 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00

ę | 0.38 0.38 0.00 0.25 0.00 0.00 0.00 0.00 0.00

į | 0.00 0.00 0.00 0.00 0.82 0.00 0.00 0.00 0.18

š | 0.00 0.00 0.00 0.00 0.00 0.80 0.00 0.00 0.20

ū | 0.00 0.00 0.00 0.00 0.00 0.00 0.93 0.07 0.00

ų | 0.00 0.00 0.09 0.00 0.00 0.00 0.09 0.82 0.00

ž | 0.00 0.00 0.00 0.00 0.13 0.00 0.00 0.00 0.87

\* Average Accuracy is 0.68.

## 50-50

Rezultatai su duomenų pasiskirstymu: 50% - apmokymui, 50% - analizei. Tai būtų 131 nuotraukos apmokymui, 128 analizei.

Testavimas su apmokymo rinkiniu:

PREDICTED

KNOWN | ą č ė ę į š ū ų ž

-------------------------------------------------------------------------

ą | 0.90 0.00 0.00 0.10 0.00 0.00 0.00 0.00 0.00

č | 0.00 0.90 0.00 0.10 0.00 0.00 0.00 0.00 0.00

ė | 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00

ę | 0.10 0.10 0.00 0.80 0.00 0.00 0.00 0.00 0.00

į | 0.00 0.00 0.00 0.00 0.93 0.00 0.00 0.00 0.07

š | 0.00 0.00 0.00 0.00 0.00 0.94 0.00 0.00 0.06

ū | 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00

ų | 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00

ž | 0.00 0.00 0.00 0.00 0.26 0.05 0.00 0.00 0.68

\* Average Accuracy is 0.91.

Testavimas su naujų raidžių rinkiniu:

PREDICTED

KNOWN | ą č ė ę į š ū ų ž

-------------------------------------------------------------------------

ą | 0.40 0.10 0.00 0.50 0.00 0.00 0.00 0.00 0.00

č | 0.40 0.20 0.00 0.40 0.00 0.00 0.00 0.00 0.00

ė | 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00

ę | 0.40 0.20 0.00 0.40 0.00 0.00 0.00 0.00 0.00

į | 0.00 0.00 0.00 0.00 0.86 0.00 0.00 0.00 0.14

š | 0.00 0.00 0.00 0.00 0.06 0.94 0.00 0.00 0.00

ū | 0.00 0.00 0.06 0.00 0.00 0.00 0.89 0.06 0.00

ų | 0.00 0.00 0.00 0.00 0.00 0.00 0.07 0.93 0.00

ž | 0.00 0.00 0.00 0.00 0.22 0.00 0.00 0.00 0.78

\* Average Accuracy is 0.71.

## 70-30

Rezultatų su duomenų pasiskirstymą 70% - apmokymui, 30% - analizei. Tai būtų 183 nuotraukos apmokymui, 76 analizei.

Testavimas su apmokymo rinkiniu:

KNOWN | ą č ė ę į š ū ų ž

-------------------------------------------------------------------------

ą | 0.71 0.14 0.00 0.14 0.00 0.00 0.00 0.00 0.00

č | 0.07 0.86 0.00 0.07 0.00 0.00 0.00 0.00 0.00

ė | 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00

ę | 0.21 0.14 0.00 0.64 0.00 0.00 0.00 0.00 0.00

į | 0.00 0.00 0.00 0.00 0.95 0.00 0.00 0.00 0.05

š | 0.00 0.00 0.00 0.00 0.00 0.96 0.00 0.00 0.04

ū | 0.00 0.00 0.00 0.00 0.00 0.00 0.96 0.04 0.00

ų | 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00

ž | 0.00 0.00 0.00 0.00 0.15 0.00 0.00 0.00 0.85

\* Average Accuracy is 0.88.

Testavimas su naujų raidžių rinkiniu:

PREDICTED

KNOWN | ą č ė ę į š ū ų ž

-------------------------------------------------------------------------

ą | 0.67 0.17 0.00 0.17 0.00 0.00 0.00 0.00 0.00

č | 0.33 0.00 0.00 0.67 0.00 0.00 0.00 0.00 0.00

ė | 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00

ę | 0.17 0.50 0.00 0.33 0.00 0.00 0.00 0.00 0.00

į | 0.00 0.00 0.00 0.00 0.75 0.00 0.00 0.00 0.25

š | 0.00 0.00 0.00 0.00 0.00 0.90 0.00 0.00 0.10

ū | 0.00 0.00 0.00 0.00 0.00 0.00 0.91 0.09 0.00

ų | 0.00 0.00 0.13 0.00 0.00 0.00 0.00 0.88 0.00

ž | 0.00 0.00 0.00 0.00 0.18 0.00 0.00 0.00 0.82

\* Average Accuracy is 0.69.

Histogram of oriented gradients algoritmas

Testavimui naudojau bendrą raidžių katalogą. Maždaug 60% kiekvienos kategorijos raidžių buvo naudotos algoritmo apmokymui, o likę 40% analizei. Testavimo duomenų rinkinys: 259 raidžių nuotraukos. Iš kurių 155 naudotos apmokyti algoritmą, kitos 104 jį testuoti,. Su šiais duomenimis gaunami tokie rezultatai:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

## 70-30

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Column1** | **ą** | **č** | **ė** | **ę** | **į** | **š** | **ū** | **ų** | **ž** |
| 'ą' | 50 | 33,3333 | 0 | 16,66667 | 0 | 0 | 0 | 0 | 0 |
| 'č' | 33,3333 | 16,66667 | 0 | 33,33333 | 16,66667 | 0 | 0 | 0 | 0 |
| 'ė' | 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 10 |
| 'ę' | 50 | 33,33333 | 0 | 16,66667 | 0 | 0 | 0 | 0 | 0 |
| 'į' | 0 | 0 | 0 | 0 | 87,5 | 0 | 0 | 0 | 12,5 |
| 'š' | 0 | 0 | 10 | 0 | 20 | 70 | 0 | 0 | 0 |
| 'ū' | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 'ų' | 0 | 0 | 12,5 | 0 | 0 | 12,5 | 37,5 | 37,5 | 0 |
| 'ž' | 0 | 0 | 0 | 0 | 18,18182 | 0 | 0 | 0 | 81,81818 |

## 60-40

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Column1** | **ą** | **č** | **ė** | **ę** | **į** | **š** | **ū** | **ų** | **ž** |
| 'ą' | 37,5 | 12,5 | 0 | 37,5 | 12,5 | 0 | 0 | 0 | 0 |
| 'č' | 25 | 12,5 | 0 | 50 | 12,5 | 0 | 0 | 0 | 0 |
| 'ė' | 0 | 0 | 64,28571 | 0 | 0 | 21,42857 | 0 | 0 | 14,28571 |
| 'ę' | 62,5 | 25 | 0 | 12,5 | 0 | 0 | 0 | 0 | 0 |
| 'į' | 0 | 0 | 0 | 0 | 90,90909 | 0 | 0 | 0 | 9,090909 |
| 'š' | 0 | 0 | 6,666667 | 0 | 26,66667 | 60 | 0 | 0 | 6,666667 |
| 'ū' | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 'ų' | 0 | 0 | 9,090909 | 0 | 0 | 9,090909 | 27,27273 | 54,54545 | 0 |
| 'ž' | 0 | 0 | 6,666667 | 0 | 13,33333 | 6,666667 | 0 | 0 | 73,33333 |

## 50-50

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Column1** | **ą** | **č** | **ė** | **ę** | **į** | **š** | **ū** | **ų** | **ž** |
| 'ą' | 0 | 20 | 0 | 80 | 0 | 0 | 0 | 0 | 0 |
| 'č' | 30 | 20 | 0 | 30 | 20 | 0 | 0 | 0 | 0 |
| 'ė' | 0 | 0 | 70,5882 | 0 | 5,88235 | 11,7647 | 0 | 0 | 11,7647 |
| 'ę' | 40 | 30 | 0 | 30 | 0 | 0 | 0 | 0 | 0 |
| 'į' | 0 | 0 | 0 | 0 | 92,85714 | 0 | 0 | 0 | 7,142857 |
| 'š' | 0 | 0 | 5,882353 | 0 | 35,29412 | 41,17647 | 0 | 0 | 17,64706 |
| 'ū' | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 'ų' | 0 | 0 | 7,142857 | 0 | 0 | 7,142857 | 28,57143 | 57,14286 | 0 |
| 'ž' | 0 | 0 | 16,66667 | 0 | 11,11111 | 5,555556 | 0 | 0 | 66,66667 |

## 50-50 SIFT

Apmokymui naudota duomenų : 131

Found correctly: 13 out of 128

Failed to find: 115 out of 128

Accuracy: 10.16 %

Naudojan thresh bendrų savybių radime 2.2.

## 60-40 SIFT

Apmokymui naudota duomenų : 155

Thresh = 1,5

Found correctly: 7 out of 108

Failed to find: 101 out of 108

Accuracy: 6.48 %

Thresh = 2,2

Found correctly: 9 out of 108

Failed to find: 99 out of 108

Accuracy: 8.33 %

Thresh = 2

Found correctly: 9 out of 108

Failed to find: 99 out of 108

Accuracy: 8.33 %

Thresh = 2,5

Found correctly: 11 out of 108

Failed to find: 97 out of 108

Accuracy: 10.19 %

## 70-30 SIFT

Apmokyta su : 183 duomenų

Thresh = 2.5

Found correctly: 22 out of 108

Failed to find: 86 out of 108

Accuracy: 20.37 %

Thresh = 3

Found correctly: 27 out of 108

Failed to find: 81 out of 108

Accuracy: 25.00 %

Thresh = 2

Found correctly: 21 out of 108

Failed to find: 87 out of 108

Accuracy: 19.44 %

## Grafikai

|  |  |  |  |
| --- | --- | --- | --- |
| **Tikslumas / Duomenų pasiskirstymas** | **50-50** | **60-40** | **70-30** |
| BoF | 0.91 | 0.89 | 0.71 |
| HOG | 0.66 | 0.73 | 0.81 |
| SIFT | 0.1 | 0.1 | 0.25 |

# Kodas

|  |
| --- |
| *BoF.m* |
| close all;    rootFolder = fullfile('C:\Users\vaitk\Desktop\IntelektikosProjektas\BoF', 'Raides');  testFolder = fullfile('C:\Users\vaitk\Desktop\IntelektikosProjektas\BoF', 'TestuojamosRaides');  categories = {'ą', 'č', 'ę', 'ė', 'į', 'š', 'ų', 'ū', 'ž'};  imds = imageDatastore(fullfile(rootFolder, categories), 'LabelSource', 'foldernames');  imdts = imageDatastore(fullfile(testFolder, categories), 'LabelSource', 'foldernames');    trainingSet = imds;  validationSet = imdts;  %[trainingSet, validationSet] = splitEachLabel(imds, 0.6);  tsc = countEachLabel(trainingSet)  vsc = countEachLabel(validationSet)    % Tiesiog paziureti ar geroje vietoje iesko raidziu  aNosine = find(validationSet.Labels == 'ą', 1);  cSuVarnele = find(validationSet.Labels == 'č', 1);  eNosine = find(validationSet.Labels == 'ę', 1);  subplot(1,3,1);  imshow(readimage(validationSet,aNosine))  subplot(1,3,2);  imshow(readimage(validationSet,cSuVarnele))  subplot(1,3,3);  imshow(readimage(validationSet,eNosine))      % Apmokome algoritma  bag = bagOfFeatures(trainingSet);    img = readimage(imds, 1);  featureVector = encode(bag, img);  categoryClassifier = trainImageCategoryClassifier(trainingSet, bag);    % Visu pirma isbandome savo apmokyta algoritma su testavimo setu,  % kad paziuretume jog gerai veiki. Tikslumas turi buti pakankamai didelis  confMatrix = evaluate(categoryClassifier, trainingSet);    % Testuojame su naujais duomenimis ir ziurime kokias reiksmes gausim  confMatrix = evaluate(categoryClassifier, validationSet);    % Vidutinis tikslumas  mean(diag(confMatrix));    img = imread(fullfile(rootFolder, 'č', '1.jpg'));  [labelIdx, scores] = predict(categoryClassifier, img);    % Spejama raide  categoryClassifier.Labels(labelIdx) |

|  |
| --- |
| *HOG.m* |
| close all;  tic  % Load training and test data using |imageDatastore|.  syntheticDir = fullfile('C:\Users\Dovydas\Desktop\kurs','TestuojamosRaides2');  handwrittenDir = fullfile('C:\Users\Dovydas\Desktop\kurs','Raides2');    % |imageDatastore| recursively scans the directory tree containing the  % images. Folder names are automatically used as labels for each image.  trainingSet = imageDatastore(handwrittenDir, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');  testSet = imageDatastore(syntheticDir, 'IncludeSubfolders', true, 'LabelSource', 'foldernames');    countEachLabel(trainingSet)  countEachLabel(testSet)      figure;    subplot(2,3,1);  imshow(trainingSet.Files{13});    subplot(2,3,2);  imshow(trainingSet.Files{16});    subplot(2,3,3);  imshow(trainingSet.Files{21});    subplot(2,3,4);  imshow(testSet.Files{5});    subplot(2,3,5);  imshow(testSet.Files{7});    subplot(2,3,6);  imshow(testSet.Files{12});      % Show pre-processing results  exTestImage = readimage(testSet,1);  processedImage = imbinarize(rgb2gray(exTestImage));    figure;    subplot(1,2,1)  imshow(exTestImage)    subplot(1,2,2)  imshow(processedImage)    img = readimage(trainingSet, 26);  processedImage = imbinarize(rgb2gray(img));    % Extract HOG features and HOG visualization  [hog\_2x2, vis2x2] = extractHOGFeatures(processedImage,'CellSize',[2 2]);  [hog\_4x4, vis4x4] = extractHOGFeatures(processedImage,'CellSize',[4 4]);  [hog\_8x8, vis8x8] = extractHOGFeatures(processedImage,'CellSize',[8 8]);    % Show the original image  figure;  subplot(2,3,1:3); imshow(processedImage);    % Visualize the HOG features  subplot(2,3,4);  plot(vis2x2);  title({'CellSize = [2 2]'; ['Feature length = ' num2str(length(hog\_2x2))]});    subplot(2,3,5);  plot(vis4x4);  title({'CellSize = [4 4]'; ['Feature length = ' num2str(length(hog\_4x4))]});    subplot(2,3,6);  plot(vis8x8);  title({'CellSize = [8 8]'; ['Feature length = ' num2str(length(hog\_8x8))]});    cellSize = [2 2];  hogFeatureSize = length(hog\_2x2);    numImages = numel(trainingSet.Files);  trainingFeatures = zeros(numImages, hogFeatureSize, 'single');    for i = 1:numImages  img = readimage(trainingSet, i);    img = rgb2gray(img);    % Apply pre-processing steps  img = imbinarize(img);  img = imresize(img, [64 64]);  trainingFeatures(i, :) = extractHOGFeatures(img, 'CellSize', cellSize);  end    % Get labels for each image.  trainingLabels = trainingSet.Labels;    classifier = fitcecoc(trainingFeatures, trainingLabels);    [testFeatures, testLabels] = helperExtractHOGFeaturesFromImageSet(testSet, hogFeatureSize, cellSize);    % Make class predictions using the test features.  predictedLabels = predict(classifier, testFeatures);    % Tabulate the results using a confusion matrix.  [confMat,order] = confusionmat(testLabels, predictedLabels)  confMat = bsxfun(@rdivide,confMat,sum(confMat,2))\*100;    letters = order  colHeadings = arrayfun(@(x)sprintf('%s',x), order,'UniformOutput',false);  format = repmat(' %-7s',1,11);  header = sprintf(format,'letters |',colHeadings{:});  fprintf('\n%s\n%s\n',header,repmat('-',size(header)));  for idx = 1:numel(letters)  fprintf(' %-7s', [letters(idx) ' |']);  fprintf(' %-9.2f', confMat(idx,:));  fprintf('\n')  end    toc |

|  |
| --- |
| trainData.m (SIFT) |
| function main  %rootFolder = fullfile('C:\Users\Migle\Desktop\Repositories\6 Semester\Intelektika\IntelektikosProjektas\SIFT\60-40\Raides');  rootFolder = fullfile('C:\Users\Migle\Desktop\Repositories\6 Semester\Intelektika\IntelektikosProjektas\SIFT\70-30\Raides3');  %rootFolder = fullfile('C:\Users\Migle\Desktop\Repositories\6 Semester\Intelektika\IntelektikosProjektas\SIFT\50-50\Raides');    featuresFilesNames = {'0.txt','1.txt','2.txt', '3.txt', '4.txt', '5.txt','6.txt','7.txt', '8.txt', '9.txt'};  descriptorsFilesNames = {'00.txt','11.txt','22.txt', '33.txt', '44.txt', '55.txt','66.txt','77.txt', '88.txt', '99.txt'};      % Get list of all subfolders.  allSubFolders = genpath(rootFolder);    remain = allSubFolders;  listOfFolderNames = {};    while true  [singleSubFolder, remain] = strtok(remain, ';');  if isempty(singleSubFolder)  break;  end  listOfFolderNames = [listOfFolderNames singleSubFolder];  end  numberOfFolders = length(listOfFolderNames);  featuresSet = zeros(numberOfFolders,2);  trueLetters = {'A nosine','S','S','A nosine'};      % Process all image files in those folders.  for k = 1 : numberOfFolders  sum = 0;  % Get this folder and print it out.  thisFolder = listOfFolderNames{k};  fprintf('Processing folder %s\n', thisFolder);  % Get PNG files.  filePattern = sprintf('%s/\*.png', thisFolder);  baseFileNames = dir(filePattern);  % Add on TIF files.  filePattern = sprintf('%s/\*.tif', thisFolder);  baseFileNames = [baseFileNames; dir(filePattern)];  % Add on JPG files.  filePattern = sprintf('%s/\*.jpg', thisFolder);  baseFileNames = [baseFileNames; dir(filePattern)];  numberOfImageFiles = length(baseFileNames);  % Now we have a list of all files in this folder.  if numberOfImageFiles >= 1  % Go through all those image files.  for f = 1 : numberOfImageFiles  fullFileName = fullfile(thisFolder, baseFileNames(f).name);    fullFileName = fullfile(thisFolder, baseFileNames(f).name);    %algoritmh    I1 = imread(fullFileName);  I1 = single(vl\_imdown(rgb2gray(I1)));  I1= imresize(I1, [550 NaN]);  %these parameters limit the number of features detected    peak\_thresh = 0; %increase to limit; default is 0  edge\_thresh = 10; % decrease to limit default is 10    [f1,d1] = vl\_sift(I1,...  'PeakThresh', peak\_thresh, ...  'edgeThresh', edge\_thresh );    name = char(featuresFilesNames(k));  descriptName = char(descriptorsFilesNames(k));  fid = fopen(name, 'wt');  dlmwrite(name,d1,'-append','delimiter','\t','precision',18);  fid2 = fopen(descriptName, 'wt');  dlmwrite(descriptName,f1,'-append','delimiter','\t','precision',18);  fclose(fid);  fclose(fid2);      end  else  %fprintf(' Folder %s has no image files in it.\n', thisFolder);  end  fprintf('Baige \n');  end |

|  |
| --- |
| findLetter.m (SIFT) |
| function test    rootFolder = fullfile('C:\Users\Migle\Desktop\Repositories\6 Semester\Intelektika\IntelektikosProjektas\SIFT\60-40\TestuojamosRaides');  found = 0;  failed = 0;  allSubFolders = genpath(rootFolder);    featuresFilesNames = {'1.txt','2.txt', '3.txt', '4.txt', '5.txt','6.txt','7.txt', '8.txt', '9.txt'};  descriptorsFilesNames = {'11.txt','22.txt', '33.txt', '44.txt', '55.txt','66.txt','77.txt', '88.txt', '99.txt'};  %trueLetters = {'C','C','C','C','C','C','C','C','C','C'};  %trueLetters = {'A nosine','A nosine','A nosine','A nosine','A nosine','A nosine','A nosine','A nosine','A nosine','A nosine'};  %50/50  % trueLetters = {'A nosine','A nosine','A nosine','A nosine','A nosine','A nosine','A nosine','A nosine','A nosine','A nosine',...  % 'C','C','C','C','C','C','C','C','C','C',...  % 'E nosine','E nosine','E nosine','E nosine','E nosine','E nosine','E nosine','E nosine','E nosine','E nosine',...  % 'E','E','E','E','E','E','E','E','E','E','E','E','E','E','E','E','E',...  % 'I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine',...  % 'S','S','S','S','S','S','S','S','S','S','S','S','S','S','S','S','S',...  % 'U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji',...  % 'U ilgoji','U ilgoji','U ilgoji',...  % 'U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine',...  % 'Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ'};  %60/40  % trueLetters = {'A nosine','A nosine','A nosine','A nosine','A nosine','A nosine','A nosine','A nosine',...  % 'C','C','C','C','C','C','C','C',...  % 'E nosine','E nosine','E nosine','E nosine','E nosine','E nosine','E nosine','E nosine',...  % 'E','E','E','E','E','E','E','E','E','E',...  % 'I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine',...  % 'S','S','S','S','S','S','S','S','S','S',...  % 'U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji',...  % 'U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine',...  % 'Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ'};  %70/30  trueLetters = {'A nosine','A nosine','A nosine','A nosine','A nosine','A nosine',...  'C','C','C','C','C','C',...  'E nosine','E nosine','E nosine','E nosine','E nosine','E nosine',...  'E','E','E','E','E','E','E','E','E','E','E','E','E','E',...  'I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine','I nosine',...  'S','S','S','S','S','S','S','S','S','S','S','S','S','S','S',...  'U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji','U ilgoji',...  'U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine','U nosine',...  'Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ','Þ'};    remain = allSubFolders;  listOfFolderNames = {};    while true  [singleSubFolder, remain] = strtok(remain, ';');  if isempty(singleSubFolder)  break;  end  listOfFolderNames = [listOfFolderNames singleSubFolder];  end  numberOfFolders = length(listOfFolderNames);  featuresSet = zeros(numberOfFolders,2);      % Process all image files in those folders.    for k = 1 : numberOfFolders  sum = 0;  % Get this folder and print it out.  thisFolder = listOfFolderNames{k};  % Get PNG files.  filePattern = sprintf('%s/\*.png', thisFolder);  baseFileNames = dir(filePattern);  % Add on TIF files.  filePattern = sprintf('%s/\*.tif', thisFolder);  baseFileNames = [baseFileNames; dir(filePattern)];  % Add on JPG files.  filePattern = sprintf('%s/\*.jpg', thisFolder);  baseFileNames = [baseFileNames; dir(filePattern)];  numberOfImageFiles = length(baseFileNames);  % Now we have a list of all files in this folder.  if numberOfImageFiles >= 1  % Go through all those image files.  for f = 1 : numberOfImageFiles  fullFileName = fullfile(thisFolder, baseFileNames(f).name);      %algoritmh  I2 = imread(fullFileName);  I2 = single(vl\_imdown(rgb2gray(I2)));  I2= imresize(I2, [550 NaN]);    peak\_thresh = 0;  edge\_thresh = 10;    [f2,d2] = vl\_sift(I2,...  'PeakThresh', peak\_thresh,...  'EdgeThresh', edge\_thresh );    % matching    letter = matching(d2,f2,featuresFilesNames,descriptorsFilesNames,I2);  letter  f  t = strcmp(letter,trueLetters(f));  if t == 1  found = found + 1;  else  failed = failed +1;  end      end  end  end    percentage = (found \* 100)/(found+failed);  fprintf('Found correctly: %d out of %d\n',found, found + failed);  fprintf('Failed to find: %d out of %d\n',failed, found + failed);  fprintf('Accuracy: %.2f %%\n',percentage);    end    function [letter] = matching(d2,f2,featuresFilesNames,descriptorsFilesNames,I2)  for k = 1:9    sum = 0;  name1 = char(featuresFilesNames(k));  name2 = char(descriptorsFilesNames(k));  d11 = importdata(name1);  d1 = uint8(d11);    f11 = importdata(name2);  f1 = double(f11);      thresh = 2;  [matches, scores] = vl\_ubcmatch(d1,d2,thresh);    indices1 = matches(1,:);  f1match = f1(:,indices1);  d1match = d1(:,indices1);    indices2 = matches(2,:);  f2match = f2(:,indices2);  d2match = d2(:,indices2);      allScales = zeros(1,size(matches,2));  allAngs = zeros(1,size(matches,2));  allX = zeros(1, size(matches,2));  allY = zeros(1, size(matches,2));    for i=1:size(matches, 2)  scaleRatio = f1match(3,i)/f2match(3,i);  dTheta = f1match(4,i) - f2match(4,i);    %force dTheta to be between -pi and +pi  while dTheta > pi  dTheta = dTheta - 2\*pi;  end  while dTheta < pi  dTheta = dTheta + 2\*pi;  end  allScales(i) = scaleRatio;  allAngs(i) = dTheta;    x1 = f1match(1,i);  y1 = f1match(2,i);  x2 = f2match(1,i);  y2 = f2match(2,i);    offset = [-x1; -y1];  offset = offset / scaleRatio;  offset = [ cos(dTheta) +sin(dTheta); -sin(dTheta) cos(dTheta)]\*offset;    allX(i) = x2 + offset(1);  allY(i) = y2 + offset(2);  end    %Use a coars Hough space  aBin = -pi:(pi/4):pi;  sBin = 0.5:(2):10;  xBin = 1:(size(I2,2)/5):size(I2,2);  yBin = 1:(size(I2,1)/5):size(I2,1);    H = zeros(length(aBin), length(sBin), length(xBin), length(yBin));  for i=1:size(matches,2)  a = allAngs(i);  s = allScales(i);  x = allX(i);  y = allY(i);    %find bin that is closest to a,s,x,y  [~,ia] = min(abs(a-aBin));  [~,is] = min(abs(s-sBin));  [~,ix] = min(abs(x-xBin));  [~,iy] = min(abs(y-yBin));    H(ia,is,ix,iy) = H(ia,is,ix,iy)+ 1;  end    %Find all bins with 3 or more features  [ap,sp,xp,yp] = ind2sub(size(H),find(H>=3));    %get features to coresponding to the largest bin  nFeatures = max(H(:));  [ap,sp,xp,yp] = ind2sub(size(H), find(H == nFeatures));  indices = [];  for i=1:size(matches,2)  a = allAngs(i);  s = allScales(i);  x = allX(i);  y = allY(i);    [~,ia] = min(abs(a-aBin));  [~,is] = min(abs(s-sBin));  [~,ix] = min(abs(x-xBin));  [~,iy] = min(abs(y-yBin));    if ia==ap(1) && is==sp(1) && ix==xp(1) && iy==yp(1)  indices = [indices i];  end  end      sum = length(indices)+ sum;    indices = [];  featuresSet(k,1) = sum(:,1);  featuresSet(k,2) = k;    end    %count where was the max matched features  maxFeatures = 0;  maxIndex = 0;  for n=1:length(featuresSet)  if maxFeatures < featuresSet(n,1)  maxFeatures = featuresSet(n,1);  maxIndex = n;  end  end    letter = '';  if maxIndex == 1  letter = 'A nosine';  end  if maxIndex == 2  letter = 'C';  end  if maxIndex == 3  letter = 'E nosine';  end  if maxIndex == 4  letter = 'E';  end  if maxIndex == 5  letter = 'I nosine';  end  if maxIndex == 6  letter = 'S';  end  if maxIndex == 7  letter = 'U nosine';  end  if maxIndex == 8  letter = 'U ilgoji';  end  if maxIndex == 9  letter = 'Þ';  end    end |

# Išvados

Galime teigti, kad algoritmai veikia, teisingai, bet ne ypač tiksliai. Priklausomai nuo duomenų kiekio apmokymui ir analizei rezultatai per daug nesikeitė, buvo 68-71% tikslumo tarpe. Tokius rezultatus labiausiai įtakoja duomenys ir jų paruošimas, kadangi raštas ranka turi daug skirtingų savybių ir sunku parašyti panašias raides, tai ir tikslumas gavosi pusėtinas. Vis dėlto Bag of Features galima naudoti teksto rašyto ranka atpažinimui. Histogram of oriented gradients algoritmas veikia geriau su didesniu apmokymu duomenų kiekiu, buvo kad algoritmas raidę prognozavo 100%, buvo, kad prognozės nepasitvirtino ir raidės konkrečiai prognozuoti nepavyko, tačiau ją palaikė panašiomis. SIFT algoritmas testuojant jo veikimą buvo tikslus, beveik 100%, tačiau pasirinkus duomenis, kad tiktų ir kitiems algoritmams (buvo parinktas nuotraukos dydis). SIFT algoritmas veikia pakankamai greitai jei analizuojamų duomenų nėra daug. Algoritmas būtų geriau naudojamas atpažinti objektus iš skirtingų kampų ir rasti ar tai yra tas pats objektas. SIFT algoritmas yra galingas, tačiau labai priklauso nuo mažų parametrų(thresh, peak\_thresh, edge\_thresh) rasti geresnius sprendimus. Taip pat, tuo pačiu metu atlikus dviejų nuotraukų savybių analizę ir palyginimą galima gauti tikslesnius duomenis, negu visas vienos raidės formų savybes sudėjus į vieną didžiulį savybių rinkinį.