VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
MATEMATINĖS INFORMATIKOS KATEDRA

**Vytautas Jankauskas**
Bioinformatikos studijų programa

# Lietuviško rankraščio atpažinimas ir taikymas

Bakalauro baigiamasis darbas

Vadovas: lektorius **Irus Grinis**

Vilnius 2017

# Turinys

# Įvadas

Kompiuteriu spausdinta ir ekrane matoma tekstinė medžiaga po truputį keičia žmonių rašymo ir skaitymo kultūrą bei yra naudojama vis plačiau. Tačiau ranka parašytas tekstas vis dar yra (ir tikriausiai visada bus) neatsiejama žmonių gyvenimo dalis. Tai ypač jaučiama švietimo įstaigose – mokyklose, kolegijose, universitetuose, kur svarbesnė informacija užrašoma ranka, kad būtų labiau išryškinta ir geriau įsiminta.

Žmonėms rega atrodo savaime suprantamas ir elementarus jutimo būdas. Jau ankstyvoje vaikystėje pradedama sąmoningai atpažinti įvairias spalvas, formas, objektus. Akimis matomas vaizdas smegenyse išskirstomas į įvairius signalus, kuriais perduodama skirtingų tipų informacija. Pavyzdžiui, norint aptikti aplinkoje konkretų objektą, smegenys analizuoja tik svarbesnes, turinčias reikiamas savybes, matomo vaizdo dalis. Žmogus per savo gyvenimą apdoroja labai daug informacijos, gaunamos visais jutimo organais. Ta informacija leidžia smegenims sukurti be galo daug įvairių sąryšių, padedančių atpažinti objektus.

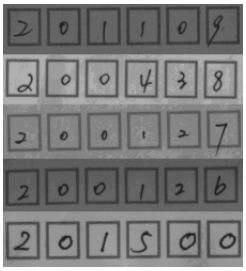
Kompiuterinis vaizdinės medžiagos apdorojimas konkrečiam tikslui įvykdyti yra vadinamas kompiuterine rega [1]. Ji iš esmės atlieka tokią pat paskirtį, kaip ir žmogaus rega. Tačiau kompiuteriai, skirtingai nei žmonės, vaizdinę medžiagą supranta tik kaip ilgą skaičių seką. Dirbtinės regos sistemos neturi jokio objektų atpažinimo modelio, nežino kurioje vietoje reikėtų fokusuoti vaizdą ar kurias jo dalis ignoruoti. Jos pačios neturi jokių sąryšių sistemų, kurios leistų palengvinti užduočių vykdymą. Taip pat bet kokiame vaizde egzistuoja įvairus triukšmas iš aplinkos, kurį sukuria kintantis apšvietimas, oro sąlygos.

Kompiuteriai visus su rega susijusius uždavinius gali išspręsti naudodami tik vaizdinę informaciją ir papildomą informaciją, kurią jiems suteikia žmonės tų užduočių įgyvendinimui. Kompiuterinė rega itin naudinga sprendžiant problemas, kurios iš žmonių reikalauja daug laiko, yra pasikartojančios. Todėl, nepaisant įvairių sunkumų, jos taikymų atsiranda vis daugiau:

- Pastaraisiais metais vis daugėja automatinio automobilių identifikacijos numerių atpažinimo sistemų. Kuriant tokias sistemas atsižvelgiama į įvairius veiksnius – numerių lokaciją, skaičių, kiekį, švarumą (lentelė su numeriais gali būti iš dalies padengta dulkėmis ar purvu), taip pat į apšvietimo sąlygas, vaizdo foną [3].


1 pav. Supaprastinta automobilių numerių atpažinimo schema [8].

- Vienas iš seniausių kompiuterinės regos taikymų yra pašto indeksų atpažinimas ant vokų ir siuntinių, leidžiantis sutaupyti daug laiko paštų darbuotojams [9].



2 pav. Pašto indeksų pavyzdžiai iš Kinijos pašto [5].

- Beveik kiekvieno išmaniojo telefono kameroje yra galimybė naudoti veido aptikimo funkciją.
- Vietose, kur renkasi dideli pėsčiųjų srautai, skaičiuojamas praeinančių žmonių kiekis. Tam naudojamos įprastos vaizdo kameros vaizdo srautą segmentuojant ir jame išskiriant kiekvieną žmogų atskirai [6].
- Sukurtas maliarijos diagnozavimo metodas, naudojant kompiuterinę regą [2]. Gydytojui leidžia daug efektyviau dirbti, kadangi įrankis analizuoja kraujo mėginių atvaizdus ir atrenka tik tuos, kuriuose didžiausia maliarijos infekcijos tikimybė.

Šie pavyzdžiai rodo, kad kompiuterinės regos galimybės vis didėja, ją naudojančios sistemos tobulėja ir yra vis plačiau taikomos kasdieniame gyvenime. Ir vis dar egzistuoja daug problemų, prie kurių sprendimo galėtų prisidėti kompiuterinė rega. Įvairi programinė įranga gana lengvai atpažįsta spausdintas raides, skaičius ar kitus simbolius. Tą padaryti sąlyginai lengviau lyginant su ranka parašytais simboliais.

Lietuviškas rankraštis pasirinktas kaip bakalauro darbo temos objektas dėl to, jog jo atpažinimas ir taikymas gali būti panaudotas praktiškai Lietuvoje. Viena iš taikymo galimybių – palengvinti mokymosi procesą mokymo įstaigose sunkiau ar lėčiau besimokantiems mokiniams. Taip pat ir studentams, kuriems paskaitų metu kartais reikia spėti greitai užsirašyti didelius informacijos kiekius. Toks pritaikymo būdas leistų labiau susikoncentruoti į ugdytojo pateikiamą informaciją, siekiant ją suprasti. Tuo pačiu metu būtų galima išsaugoti ranka užrašytą tekstą skaitmeniniu formatu ir jį vėliau panaudoti mokymosi tikslais.

Šio darbo tikslas:

- Sukurti kompiuterinės programos, kuri atpažįsta lietuvišką rankraštį, prototipą. Programa turi gebėti iš nuotraukų su ant lentos užrašytu tekstu atpažinti ir vartotojui pateikti skaitmeninį teksto variantą.

Uždaviniai:

- Susipažinti su kompiuterinės regos biblioteka OpenCV, naudojama įvairioms vaizdo apdorojimo operacijoms atlikti.
- Susipažinti su elementariais mašininio mokymosi įrankiais ir algoritmais.
- Sukurti mokymosi duomenų aibę ir su ja apmokyti teksto atpažinimo algoritmą.

- Sukurti Python programavimo kalba parašytą algoritmą, kuris nuskaito ir kuo tiksliau atpažįsta lietuvišką tekstą iš nuotraukos.

Teorinė šio darbo reikšmė yra sužinoti, kokios vaizdo apdorojimo procedūros ir įrankiai yra būtini norint normalizuoti atvaizdą, iš jo pašalinti aplinkos triukšmus. Įgyti bazines žinias apie duomenų rinkimą, paruošimą mašininio mokymosi algoritmams ir apie pačius algoritmus. Taip pat praplėsti supratimą apie panašių problemų sprendimus naudojantis kompiuterine rega.

Sukurti lietuvišką rankraštį atpažįstančios programos prototipą yra praktinė šio darbo reikšmė. Ši programa suteiktų galimybę greitai ir sklandžiai išsaugoti ranka parašytą tekstą skaitmeniniu formatu. Tai leistų ją panaudoti švietimo srityje moksleiviams, studentams. Taip pat darbas yra orientuotas į lietuvių kalbos rašmenis, todėl jis gali būti naudingas ir sprendžiant teksto vertimo į kitas kalbas problemas.

# 1 Naudotos technologijos

## 1.1 Kompiuterinės regos biblioteka OpenCV

Pagrindinis šio bakalauro darbo įrankis, naudotas programos prototipo kūrimui, yra OpenCV biblioteka [10]. Ši biblioteka – vienas populiariausių įrankių kompiuterinės regos programoms kurti. Ji yra atvirojo kodo, turi aktyvią vartotojų bendruomenę. Biblioteką sudaro daugiau nei 2500 algoritmų, iš kurių dauguma skirti vaizdų apdorojimui, taip pat mašininiam mokymui. Nors ir parašyta C++ programavimo kalba, OpenCV turi C, Python, Java ir MATLAB kalbų sąsajas, taip palengvindama naudojimąsi. Taip pat palaikomos Windows, Linux, Mac OS ir Android operacinės sistemos. Dažniausiai naudojama kuriant realiu laiku veikiančias programas.

Buvo pasirinkta naudoti Python programavimo kalbos sąsają, kadangi ji yra paprasta naudoti ir darbo autorius jau turėjo patirties programuojant šia kalba. OpenCV biblioteka turi labai įvairių funkcijų ir pritaikymo būdų. Kadangi darbe buvo naudota tik nedidelė jų dalis, būtent jos yra apžvelgtos toliau.

- Atvaizdo spalvų erdvės transformacijos. Spalvų erdvė yra metodas, kuriuo galima nustatyti, kurti ir atvaizduoti spalvas. Kiekvieną atvaizdą galima apibūdinti bet kurioje spalvų erdvėje [14]. Priklausomai nuo išsikelto uždavinio, svarbu pasirinkti tinkamą spalvų formatą. Dažniausiai sutinkamas formatas, įprastas kompiuteriuose, televizoriuose, skaitmeninėse vaizdo kamerose yra RGB. Juo nurodoma, koks kiekis raudonos, žalios ir mėlynos spalvos reikalingas, norint atitikti konkrečią spalvą. Tuo tarpu HSV erdvėje spalvos aprašomos pagal jose esantį pilkos spalvos kiekį ir pagal jų šviesumo stiprumą. Dar kompiuterinėje regoje dažnai naudojama ir pilkos spalvos pustonių (angl. grayscale) erdvė. Joje aprašoma tik juodos spalvos vertė.

- Kita svarbi, dažnai naudojama vaizdo apdorojimo operacija yra atvaizdo slenksčio (angl. image threshold) nustatymas. Šio metodo metu kaip įvestis naudojamas paveikslėlis pilkos spalvos pustonių spalvų erdvėje. Jame, pagal pasirenkamą arba algoritmo nustatomą spalvos vertę kiekvienas taškas įgauna nulio arba vieneto vertę. Tai reiškia, kad didesnę už slenkstinę vertę turintys taškai įgis juodos spalvos reikšmę, ir atvirkščiai.

3 pav. Pilkumo tonų atvaizdas kairėje, slenkstinis – viduryje. Invertuotas slenkstinis – dešinėje [15].

- Vartotojo veiksmų fiksavimas, konkrečiau – paspaustų mygtukų fiksavimas yra dar viena OpenCV bibliotekos funkcija, kuria naudotis labai paprasta.

```python
# Importuoti OpenCV biblioteką:
import cv2
# Kintamajam key priskiriame pirmo paspausto mygtuko reikšmę
key = cv2.waitKey(0)
# Jei vartotojas paspaudžia "Enter", išspausdinti:
if key == 13:
        print("Paspaudėte Enter.")
```

1 kodo fragmentas. Mygtukų paspaudimo naudojimo pavyzdys.

- Morfologinės transformacijos yra paprastos operacijos, atliekamos remiantis atvaizdo forma. Dažniausiai šioms operacijoms atlikti naudojami binariniai atvaizdai. Kaip įvesties elementai morfologinėms transformacijoms naudojamas norimas atvaizdas ir papildomas struktūrinis elementas, kuris ir nusprendžia, kaip bus vykdoma operacija. Struktūrinis elementas išties yra kvadratinė matrica, dažniausiai užpildyta vienetais. Pagrindinės operacijos yra ardymas (angl. erosion) ir išplėtimas (angl. dilation) [11].

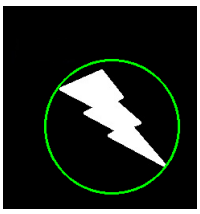
  Ardymo operacija veikia panašiai kaip ir dirvožemio ardymas, tik ji ardo atvaizdo priekinio plano objekto kraštus. Struktūrinis elementas slenkamas per paveikslėlio taškus. Originaliam paveikslėlio taškui (turinčiam vertę 0 arba 1) bus priskiriama vertė 1 tik tuo atveju, jei visi taškai po struktūriniu elementu bus lygūs 1. Kitu atveju paveikslėlio taškui priskiriama 0 vertė (jis suardomas). Taip sumažinamas priekinio vaizdo objekto plotas.

4 pav. Kairėje – originalus atvaizdas. Dešinėje – po ardymo operacijos [11].

  Išplėtimo operacija yra atvirkščia ardymui. Šiuo atveju, jei bent viena taško vertė struktūriniame elemente yra lygi vienam, tuomet ir originalus taškas atvaizde bus lygus vienam. Taigi, šiuo metodu išplečiamas priekinio vaizdo objekto plotas. Dažniausiai, kai norima sumažinti triukšmą paveikslėlyje, po ardymo vykdomas išplėtimas. Taip ardymas pašalina baltus triukšmo taškus atvaizde, bet ir sumažina priekinio objekto plotą. Tam, kad objekto plotas vėl padidėtų, atliekamas išplėtimas, bet triukšmas nebeatsiranda iš naujo.

5 pav. Kairėje – originalus atvaizdas. Dešinėje – po išplėtimo operacijos [11].

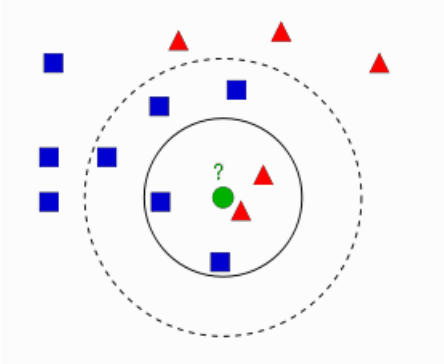
- Kontūrų radimas kompiuterinėje regoje dažniausiai naudojamas aptinkant įvairius objektus, juos atpažįstant ar nustatant jų formas [12]. Kontūrą galima apibūdinti kaip visus vienodos spalvos ar intensyvumo taškus (aplink jų pakraščius) apibrėžiančią liniją. Norint gauti tikslesnius objekto kontūrus, rekomenduojama šią operaciją atlikti binariniam atvaizdui.



6 pav. Kontūrų aptikimas binariniame atvaizde [12].

- Mašininio mokymo algoritmai. OpenCV biblioteka turi funkcijų įvairioms mašininio mokymo operacijoms atlikti – statistinius modelius, KNN (angl. K–Nearest Neighbors) algoritmą, SVM (angl. Support Vector Machine) algoritmą, sprendimų medžių, atsitiktinių medžių algoritmus, neuroninius tinklus ir kitus.

  Plačiau apžvelgiamas KNN algoritmas. Jis yra vienas paprasčiausių klasifikacijos algoritmų, naudojamų prižiūrimam mokymui (angl. supervised learning). Pagrindinė algoritmo idėja yra ypatybių erdvėje surasti artimiausią atitikmenį testiniams duomenims [16]. Pav. 7 dvi skirtingas duomenų klases vaizduoja mėlyni kvadratai ir raudoni trikampiai. Kiekvienas duomuo turi po 2 ypatybes (grafike tai parodo x ir y koordinatės ir taško pozicija). Naujas testinis duomuo yra žalias apskritimas. Tad proceso, vadinamo klasifikacija metu algoritmas turi nuspręsti, kuriai duomenų klasei jį priskirti. Jei žiūrima tik į vieną artimiausiai esantį kaimyną, tai testinis duomuo būtų priskirtas raudonų trikampių klasei (kaip ir jei ieškoma 2 artimiausių kaimynų). Klasifikuojant pagal 3 kaimynus, duomuo vis tiek būtų priskirtas raudonųjų klasei. Naudojant šį algoritmą nereikėtų rinktis lyginio kaimynų skaičiaus, nes tuomet rezultatas gali būti lygus ir reikėtų ieškoti kitų savybių norint teisingai klasifikuoti. Pasirinkus 5 ar daugiau kaimynų, matoma, kad tokiu atveju žaliasis apskritimas būtų priskirtas mėlynųjų klasei. Toks klasifikavimas žiūrint į duomenų išsidėstymą atrodo teisingas.




7 pav. KNN algoritmo vizualizacija 2 duomenų klasėse [16].

Galima tarti, kad KNN algoritmui įtaką daro tik artimiausių kaimynų skaičius. Tačiau būtina paminėti, kad klasifikuojant kiekvieną naują duomenų aibės narį, esamiems duomenims priskiriami skirtingi svoriai. Jie nustatomi pagal atstumą iki naujojo nario. Remiantis šia savybe, galima tarti, kad klasifikuojant pav. 7 grafike esantį žalią apskritimą pagal 4 artimiausius kaimynus, jis būtų priskirtas raudonųjų trikampių klasei, kadangi šie yra arčiau nei mėlynieji kvadratai.

## 1.2 Python programavimo kalbos paketai

### 1.2.1 NumPy

NumPy yra vienas svarbiausių ir populiariausių Python programavimo kalbos paketų, naudojamų moksliniams skaičiavimams. Šis paketas turi labai funkcionalų N dimensijų masyvo objektą, taip pat – naudingas tiesinės algebros, atsitiktinių skaičių naudojimo galimybes. NumPy naudojamas kaip efektyvi kelių dimensijų talpykla bendriniams duomenims saugoti. Pakete galima apibrėžti ir naujus sutartinius duomenų tipus [20].

```
1 # NumPy paketo importavimas:
2 import numpy as np
3 # Sukuriamas masyvas, kurio pradinės vertės yra atsitiktinės:
4 samples =  np.empty((1,300))
```

2 kodo fragmentas. Numpy duomenų masyvo su atsitiktiniais duomenimis sukūrimas.

### 1.2.2 Sys ir os paketai

Šie paketai visuomet paruošti naudojimui. Jų nereikia papildomai atsisiųsti kaip daugumos kitų.

Sys paketas skirtas sukurti prieigai prie programos vykdymo interpretatoriaus (angl. interpreter) naudojamų kintamųjų [21].

```
1 # Komandų eilutės argumentai, perduodami Python skriptui:
2 systemargs = sys.argv
```

3 kodo fragmentas. Sys paketo naudojimas komandinės eilutės argumentams gauti.

Os paketas leidžia naudoti nuo operacinės sistemos priklausančias funkcijas. Naudingas failų skaitymui ir rašymui, darbui su direktorijomis, laikinų failų valdymui [22].

```
1 # Naujo aplanko failų sistemoje sukūrimas:
2 os.mkdir(os.path.join('letters', str(key)))
```

4 kodo fragmentas. Os paketo naudojimas naujam aplankui sukurti.

### 1.2.3 Difflib paketas

Difflib suteikia klases bei funkcijas, reikalingas sekų palyginimui. Gali būti naudojamas įvairių formatų failų, skaitinių verčių ar teksto palyginimui [23].

```
1 # Žodžių, panašiausių į „žodis“ paiešką masyve „žodynas“.
2 # Cutoff parametras nurodo norimą panašumo koeficientą.
3 something = difflib.get_close_matches("žodis", zodynas, n=3, cutoff= 0.35)
```

5 kodo fragmentas. Panašių žodžių paieška naudojant difflib paketą.

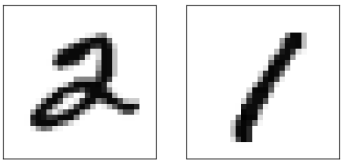
# 2 Problemos sprendimas

Norint sukurti programos, gebančios atpažinti lietuvišką rankraštį, prototipą ir jį pritaikyti, buvo pasirinkta Python programavimo kalba. Naudojantis įvariais kalbos moduliais bei OpenCV kompiuterinės regos biblioteka atlikti išsikelti uždaviniai ir įgyvendintas pagrindinis darbo tikslas – sukurtas ir praktiškai išbandytas prototipas. Šiame skyriuje aprašomas tų uždavinių sprendimas, jų rezultatai, kilusios problemos, pastabos ir pasiūlymai. Programos prototipas susideda iš 3 Python kalba parašytų programinių failų, tekstinio žodyno failo, prototipo metu sukurto mokymo duomenų rinkinio ir iš jo sugeneruotų algoritmo apmokymo failų:

- *classification.py* – programos dalis, kuri naudojama mokymosi duomenų rinkinio sukūrimui.
- *learningset.py* – programos dalis, kurioje mokymosi duomenys sudedami į 2 atskirus tekstinius mokymosi failus atvaizdams ir jų žymėms.
- *testBoard.py* – programos dalis, naudojama rašmenų aptikimui ir atpažinimui.
- *dictionary.data* – tekstinis failas. Jame surašyti žodžiai, kuriuos algoritmas potencialiai gali atpažinti. Naudojamas teksto atpažinimo patikslinimui.
- *lettertags.data* – lietuviškos abėcėlės raidžių sąrašas, kuris (pagal eilučių numerius) atitinka mokymosi duomenų aplankų pavadinimus.

## 2.1 Mokymo duomenų ruošimas

Kadangi šio darbo tikslas yra programa, gebanti atpažinti ranka parašytą lietuvišką tekstą, tam reikalingi duomenys, kuriais galima apmokyti algoritmą prižiūrimo mašininio mokymo būdu. Kiekvienas šiam būdui reikalingas duomenų elementas turi būti tinkamai paruoštas: to paties duomenų tipo, vienodo dydžio, su žyma, kokiai klasei elementas yra priskirtas. Vienas žinomiausių tokių rinkinių yra MNIST ranka rašytų skaičių rinkinys. Jį sudaro 60000 mokymo elementų ir 10000 elementų, skirtų testavimui [17]. Kiekvienas elementas sudarytas iš 2 dalių – 28 taškų aukščio ir 28 taškų pločio binarinio atvaizdo (viso turi 784 taškus) ir žymės, kokiam skaitmeniui atvaizdas yra priskirtas.



8 pav. MNIST duomenų rinkinio elementų pavyzdžiai [18].

Lietuvių kalbos rašmenims tokio tipo paruošto duomenų rinkinio nėra, todėl vienas iš baigiamojo darbo uždavinių ir buvo sukurti bent jau bandomąjį duomenų rinkinį, atitinkantį šias sąlygas:

- Rinkinį sudaro tik lietuvių kalbos abėcėlės raidės. Dabartinėje abėcėlėje yra 32 raidės. Kiekviena raidė gali būti didžioji arba mažoji. Remiantis dabartinės lietuvių kalbos žodynu, nėra žodžių, prasidedančių Ų ir Ę, todėl šios didžiosios raidės neįtraukiamos į rinkinį [19].

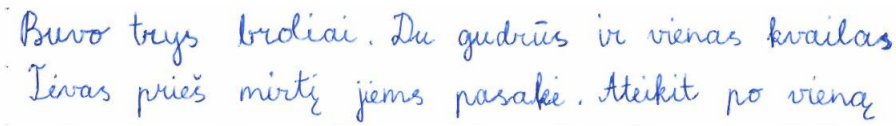
- Kiekviena raidė yra parašyta šio darbo autoriaus. Šis apribojimas įvestas, nes planuojama rinkinio apimtis nebus didelė, tad ir teksto rašymo stilius turi būti panašus visuose elementuose.

- Norint išvengti dar didesnio darbo kompleksiškumo, į rinkinį neįtraukiami jokie skyrybos ženklai.

- Rinkinyje yra po 50 kiekvienos raidės atvaizdų. Kiekvienas iš jų yra juodai baltas (binarinis) 30 taškų aukščio ir 10 taškų pločio (iš viso turi 300 taškų). Toks atvaizdo dydis pasirinktas atsižvelgiant į galimas raidžių aukščio ir pločio charakteristikas.

- Kiekvienas paveikslėlis, kartu su žyme, rodančia, kokiai raidei yra priskirtas, sudaro vieną duomenų rinkinio elementą.

Pirmiausia reikalinga tekstinė medžiaga buvo parašyta ir nuskenuota. Teksto turinys parinktas taip, kad neturėtų sintaksiškai sudėtingų žodžių, o jo skyryba buvo supaprastinta siekiant išvengti kablelių, brūkšnių ir kitų skyrybos ženklų.




9 pav. Teksto, naudoto duomenų rinkinio kūrimui, ištrauka.

Vėliau vykdomos vaizdo apdorojimo operacijos. Apie jas plačiau rašoma 2.2 poskyryje „Vaizdo apdorojimas“, todėl čia pateikta tik jų vykdymo tvarka ir programos kodo fragmentai.

Atvaizdas įkeliamas į programą. Tuomet vykdoma spalvų erdvės transformacija ir erdvė pakeičiama iš RGB į pilkos spalvos pustonių. Tada pagal prisitaikančią slenkstinę vertę (angl. adaptive treshold) paveikslėlis paverčiamas binariniu.

```
# Įkeliame paveikslėlį:
img = cv2.imread('learn2.png')
# Keičiame spalvų erdvę iš RGB į grayscale:
img = cv2.cvtColor(newimage,cv2.COLOR_BGR2GRAY)
# Prisitaikančios slenkstinės vertės funkcija:
thresh1 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
        cv2.THRESH_BINARY,15,24)
```

6 kodo fragmentas. Mokymosi duomenų atvaizdo paruošimas apdorojant vaizdą.

Kitas žingsnis yra atlikti morfologines transformacijas, kad būtų išryškinti teksto kontūrai. Atlikti bandymai su įvairiais struktūrinio elemento dydžiais, skirtinga ardymo ir išplėtimo operacijų vykdymo tvarka ir jų iteracijų skaičiumi. Galiausiai nustatytos optimaliausios šių parametrų vertės mokymosi duomenų atvaizdams apdoroti.

```
# Struktūrinio elemento (2x2 dydžio) sukūrimas ir atliktos 2 ardymo operacijos iteracijos:
kernel2 = np.ones((2,2),np.uint8)
eroded = cv2.erode(thresh1,kernel2,iterations = 2)
```

7 kodo fragmentas. Mokymosi duomenų atvaizdo morfologinė ardymo transformacija.

Verta paminėti, kad šiuo atveju ardymo operacija atlikta norint išryškinti teksto kontūrus, nes tekstas binariniame paveikslėlyje buvo juodos spalvos, o jo fonas – baltos.

Išryškintame binariniame atvaizde atliekama objektų kontūrų paieška.

```
#Kontūrų suradimas:
im2, contours, hierarchy = cv2.findContours(eroded,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

8 kodo fragmentas. Kontūrų suradimas atvaizde.

Rastų kontūrų koordinatės išsaugomos duomenų masyve. Tuomet jie filtruojami – pagal plotą atmetami neatitinkantys raidžių ar žodžių užimamo ploto. Likusieji kontūrai dar kartą filtruojami norint atmesti esančius raidžių viduje. Tam naudojama darbo autoriaus parašyta *removeInnerContours* funkcija.

10 pav. Atvaizdas, kuriame vienas kontūras yra kito viduje.

Išfiltravus netinkamus kontūrus pradedamas mokymosi duomenų atvaizdų karpymas. Iteruojamas žodžių kontūrų masyvas. Kiekvienos iteracijos metu apdorojamas naujas žodis. Algoritmas pradeda darbą nuo numatyto pločio atkarpos. Tuomet vartotojas, naudodamasis klaviatūros navigaciniais klavišais nustato tikslią konkrečios raidės poziciją.

11 pav. Žodžio sukarpymas į pavienes raides.

Pasirinktas atvaizdas yra pažymimas spaudžiant atitinkamą raidės klavišą klaviatūroje. Atvaizdas sumažinamas iki 30 taškų aukščio ir 10 taškų pločio. Tuomet jis išsaugomas aplanke, kurio pavadinimas ir reiškia raidės žymą.

Tokiu būdu buvo atrinkta ir sužymėta iš viso 3100 programos mokymo atvaizdų, po 50 vienai raidei.

Turint reikalingus mokymo duomenų atvaizdus vykdoma *learningset.py* programa, kuri visų atvaizdų taškų skaitines vertes surašo į tekstinį failą *generalsamplesNEW.data*. Viena eilutė faile atitinka vieną atvaizdą ir turi 300 skaitinių reikšmių. Kiekvieno atvaizdo žyma analogiškai įrašoma į tekstinį *generalresponsesNEW.data* failą. Jame viena eilutė atitinka vieną žymą.

```
# Atvaizdo taškų skaitinės reikšmės:
2.5500000000000000000e+02 2.5500000000000000000e+02 2.5500000000000000000e+02 ...
# Atvaizdo žymos skaitinė reikšmė:
5.3000000000000000000e+01
```

9 kodo fragmentas. Galutinis mokymosi duomenų formatas.

## 2.2 Vaizdo apdorojimas

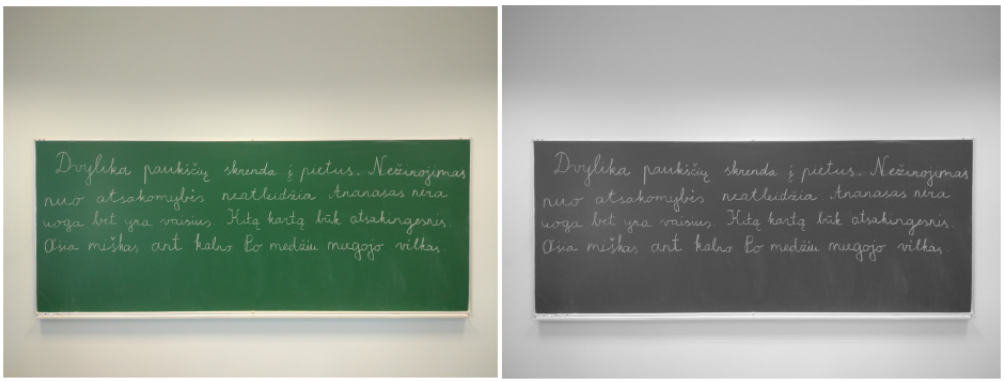
Pirmas žingsnis programos kūrime buvo surinkti kokybišką vaizdinę medžiagą. Todėl šiame darbe naudoti atvaizdai buvo fiksuojami esant geroms, pastovioms apšvietimo sąlygoms. Tokiu būdu norėta išvengti papildomo triukšmo paveikslėliuose. Taip pat buvo stengiamasi išrinkti pakankamai kontrastingus priekinio plano objektus ir foną.

Darbe naudojami atvaizdai pirmiausiai įkeliami į programą naudojant OpenCV bibliotekos funkcijas. Taip pat sumažinama jų skiriamoji geba.

```
# Importuoti OpenCV biblioteką:
import cv2
# Įkelti atvaizdą į programą:
img = cv2.imread('gb3.png')
# Sumažinti skiriamąją gebą:
newx,newy = img.shape[1]/3.5,img.shape[0]/3.5
newimage = cv2.resize(img,(int(newx), int(newy)))
```

10 kodo fragmentas. Atvaizdo įkėlimas į programą.

Kitame žingsnyje naudojama spalvų erdvės transformacija iš RGB erdvės į pilkos spalvos pustonių. Tokiu būdu gaunamas juodai baltas paveikslėlis.




12 pav. Kairėje – paveikslėlis RGB erdvėje, dešinėje – pilkos spalvos pustonių. Fotografuota VU MIF, 409 auditorijoje.

## 2.3 Rankraščio atpažinimo algoritmas

Turint mokymo duomenis išbandomas rankraščio atpažinimas. Jis vykdomas naudojant *testBoard.py* faile esantį algoritmą, kurio veikimas aprašytas šiame poskyryje. Vykdant programą vartotojui kaip komandos argumentą reikia nurodyti norimo analizuoti atvaizdo vardą.

Programą nuspręsta apriboti 250 žodžių žodynu, kuris laikomas *dictionary.data* faile. Jį sudaro atsitiktinai parinkti sintaksiškai nesudėtingi lietuvių kalbos žodžiai, kurie nebūtinai sudaro prasmingus žodžių junginius. Žodynas algoritme naudojamas kiekvieno atpažinto žodžio patikslinimui.

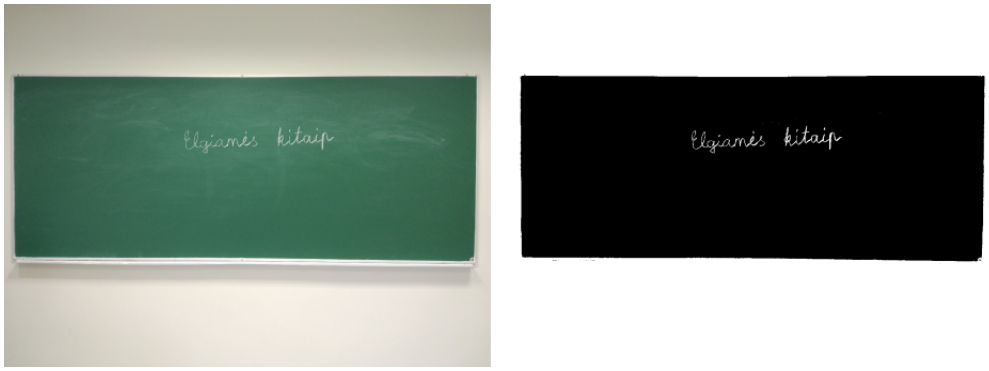
Vykdant programą pirma atliekamas mašininio mokymo KNN algoritmo apmokymas. Tam naudojamos OpenCV bibliotekoje esančios algoritmo mokymo modelio sukūrimo ir apmokymo funkcijos.

```
# Mokymo duomenų rinkinio failų įkėlimas:
samples = np.loadtxt('generalsamplesNEW.data',np.float32)
responses = np.loadtxt('generalresponsesNEW.data',np.float32)
responses = responses.reshape((responses.size,1))
# KNN algoritmo modelio sukūrimas ir apmokymas:
global model
model = cv2.ml.KNearest_create()
model.train(samples, cv2.ml.ROW_SAMPLE, responses)
```

11 kodo fragmentas. KNN algoritmo modelio apmokymas.

Modelio apmokymo funkcijos įvestyje naudojami iš mokymo duomenų sugeneruoti tekstiniai failai *generalsamplesNEW.data* ir *generalresponsesNEW.data*.

Vėliau vykdomas elementarus įvesties atvaizdo apdorojimas. Paveikslėlis yra įkeliamas į programą, jo skiriamoji geba sumažinama, o spalvų erdvė pakeičiama į pilkos spalvos pustonių erdvę. Tada atliekama binarinio slenksčio nustatymo operacija.



13 pav. Kairėje – originalus paveikslėlis, dešinėje – po binarinio slenksčio nustatymo.

Binariniame atvaizde atliekama objektų kontūrų paieška. Atrenkamas objektas, kuriame yra užrašytas tekstas ir atvaizdas apkerpamas. Gauto paveikslėlio spalvos invertuojamos. Antrą kartą nustatomas jo slenkstis, tačiau šįkart – pagal prisitaikančią slenkstinę vertę.

Atliekamos morfologinės vaizdo transformacijos – viena išplėtimo, dvi ardymo ir vėl viena išplėtimo operacijų iteracijos. Paruoštame atvaizde vėl vykdoma kontūrų paieška. Kaip ir ruošiant mokymo duomenų rinkinį, kontūrai filtruojami paliekant tik tuos, kurie galimai atitinka žodžius.

Gautas žodžių kontūrų masyvas iteruojamas analizuojant kiekvieną iš elementų. Žodžių skaldymas į pavienes raides vykdomas *adjustWidth* funkcijoje:

- Nuo kontūro pradžios atkerpamas nustatyto pločio ir aukščio atvaizdas.
- Atvaizdas, jo koordinatės ir matmenys originaliame paveikslėlyje išsaugomi ir perduodami į funkciją *getRoi*.
- Joje atvaizdas apdorojamas ir paverčiamas į tokią pat išraišką, kaip ir mokymosi duomenų rinkinio elementai.
- KNN algoritmas panaudojamas nustatyti, į kokią raidę panašiausias ir kiek nuo jos nutolęs įvesties paveikslėlis. Funkcija kaip atsakymą grąžina į analizuojamą atvaizdą panašiausio elemento žymę ir atstumą iki jo (kitaip tariant – kiek ir į kokią raidę jis panašiausias). Šios vertės išsaugomos.

```
1 # Surandamas artimiausias kaimyninis elementas:
2 retval, results, neigh_resp, dists = model.findNearest(roismall, k = 1)
```

12 kodo fragmentas. KNN algoritmo artimiausio kaimyninio elemento paieška.

- Grįžtama prie to paties žodžio kontūro. Padidinamas apkerpamo atvaizdo plotis ir aukštis, o gautas paveikslėlis vėl analizuojamas naudojant KNN algoritmą.
- Procedūra kartojama, kol pasiekiamas maksimalus nustatytas vienos raidės aukštis ir plotis originaliame atvaizde. Tuomet visi rezultatai apdorojami ir išrenkamas mažiausią skaitinę vertę turintis (arčiausiai esantis) elementas. Jis laikomas panašiausiu į kažkurią raidę ir yra išsaugomas.
- Kadangi gautos raidės koordinatės ir plotis originaliame atvaizde žinomi, pagal juos nustatoma, nuo kur algoritmui pradėti naują paiešką. Tokiu būdu išanalizuojamas visas vienas kontūras.
- Išsaugoti rezultatai sujungiami į žodį, pagal kurį žodyne randami trys į jį panašiausi atitikmenys. Programa juos visus išspausdina. Viena atpažinimo iteracija baigta, analogiškai vienas po kito analizuojami kiti paveikslėlyje esantys žodžiai.

# 3 Eksperimentas

Sukūrus rankraščio programos prototipą buvo atliktas eksperimentas jo veikimo išbandymui. Nuspręsta, kad prototipo veiksmingumas bus tikrinamas naudojant atitinkamomis sąlygomis gautus atvaizdus su juose parašytais lietuviškais žodžiais.

Pasirinktos eksperimento sąlygos – universiteto auditorija, kurioje yra pastovus dirbtinis apšvietimas. Taip pat norint gauti kontrastingą vaizdą tekstas balta kreida užrašomas tamsiai žalioje lentoje. Lentą supa šviesus fonas. Visos šiame eksperimente naudojamos nuotraukos užfiksuotos Vilniaus Universiteto Matematikos ir Informatikos fakulteto 409 ir 415 auditorijose.

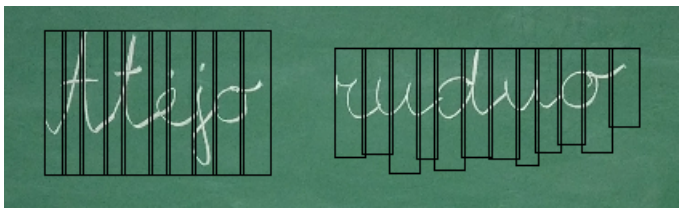
14 pav. Eksperimentinio atvaizdo pavyzdys.

Eksperimentui įvykdyti iš darbo autoriaus sudaryto žodyno (esančio faile *dictionary.data*) atsitiktine tvarka buvo pasirinkti 100 žodžių. Iš jų sudaryti įvairūs (nebūtinai prasmingi) sakiniai ar žodžių junginiai. Jie įvairiu kiekiu ir tvarka buvo užrašomi ant žalios lentos. Taip gauti reikalingi atvaizdai.

Programos prototipas į komandinę eilutę grąžina dviejų tipų atsakymus. Tarpiniai rezultatai parodo kokias raides žodyje atpažino KNN algoritmas, jų ribos (apibrėžtos stačiakampiais) vaizduojamos originaliame atvaizde. Galutiniai rezultatai gaunami žodyne ieškant panašiausio į gautą žodį atitikmens. Paieška žodyne grąžina tris į užklausą panašiausius variantus. Šis apribojimas įvestas, kadangi laikoma, jog daugiau nei trys galimi rezultatų variantai suteiktų algoritmui nepakankamą tikslumą atpažįstant žodžius. Jei žodyne nerandamas žodžio atitikmuo, laikoma, jog galutinis eksperimento rezultatas yra KNN algoritmo atpažintas žodis.

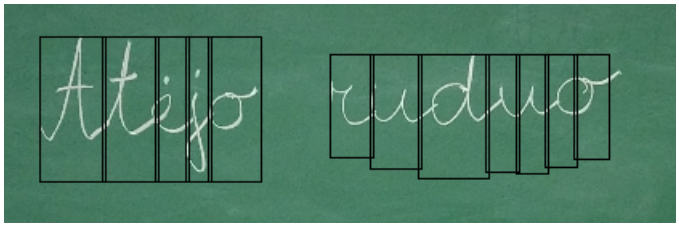
## 3.1 Tinkamų parametrų nustatymas

Pirmieji atvaizdų analizės bandymai parodė, kad galimus raidžių dydžius (plotį ir aukštį) reikia nustatyti eksperimentiniu būdu. Priešingu atveju žodžiai segmentuojami į raides labai netiksliai ir gauti rezultatai neturi jokios prasmės.



15 pav. Netikslus žodžių segmentavimas.

Keletą kartų pakartojant eksperimentą ir tikslinant raidžių dydžius, matoma, kad žodžiai sukarpomi į raides žymiai tiksliau.

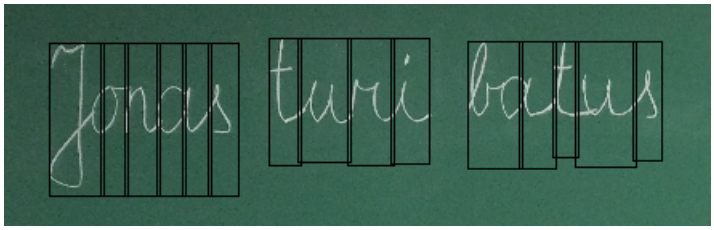


16 pav. Tikslesnis žodžių skirstymas į raides.

## 3.2 Eksperimento eiga

Nustačius tinkamus parametrus pradėti vykdyti bandymai naudojant eksperimentinius atvaizdus. Šiame poskyryje detaliau analizuojama bandymų eiga.

Pirmojo bandymo metu analizuojamas atvaizdas, kuriame parašyti žodžiai „Jonas“, „turi“, „batus“.

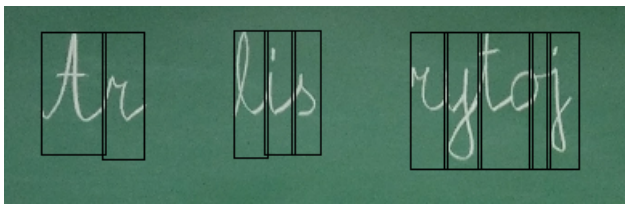


17 pav. Pirmojo bandymo atvaizdo žodžių segmentavimas į atskiras raides.

Tarpiniai rezultatai rodo, kad KNN algoritmas atpažino žodžius „Jonccs“, „tuni“, „hetus“. Pastebima, kad dalis raidžių buvo identifikuotos klaidingai – „a“ kaip „c“ ir „c“, „r“ kaip „n“, „b“ kaip „h“ ir „a“ kaip „e“.

Įvykdžius paiešką žodyne, iš pirmo karto atpažinti žodžiai „Jonas“ ir „turi“. Tačiau „hetus“ atpažintas klaidingai, tad ir galutinis paieškos rezultatas neteisingas – gauti variantai „pietus“, „lietus“ ir „sotus“.

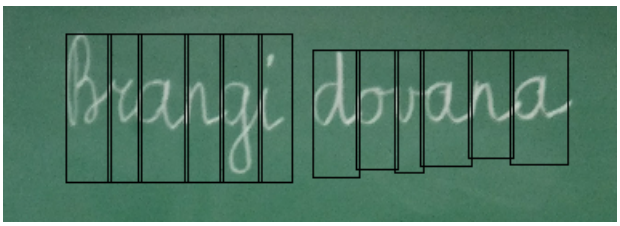
Antrojo bandymo metu analizuojamas atvaizdas *test2.png*. Jame parašyti žodžiai „Ar“, „lis“ ir „rytoj“.



18 pav. Antrojo bandymo atvaizdo žodžių padalijimas į raides.

Gauti tarpiniai rezultatai – „ar“, „lės“ ir „ryboj“. Šiuo atveju „i“ identifikuota kaip „ė“ ir „t“ – kaip „b“. Paieška žodyne grąžina „ar“ bei „rytoj“ kaip pirmąjį galimą variantą. Tuo tarpu „lis“ yra grąžinamas tik kaip trečiasis variantas.

Dalyje bandymų rankraštis atpažįstamas sėkmingai ir beveik be klaidų. Pavyzdžiui, analizuojamas *test9.png* atvaizdas. Jame ant lentos parašyti žodžiai „Brangi“ ir „dovana“.



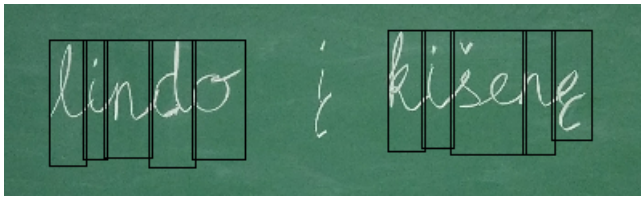
19 pav. Sėkmingo atvaizde esančių žodžių skirstymo į raides pavyzdys.

Įvykdžius programą iš gauto vaizdo matoma, kad suskirstymas į raides atliktas sėkmingai. Tarpiniai rezultatai – „Brungi“ ir „dovana“. Sėkmingai atpažintos visos raidės, išskyrus vieną – vietoje „a“ buvo identifikuota „u“. Tačiau įvykdžius paiešką žodyne ši klaida ištaisoma ir abu žodžiai atpažinti teisingai.

Toliau analogiškai vykdomi bandymai su kitais atvaizdais ir fiksuojami gauti rezultatai.

## 3.3 Algoritmo trūkumai

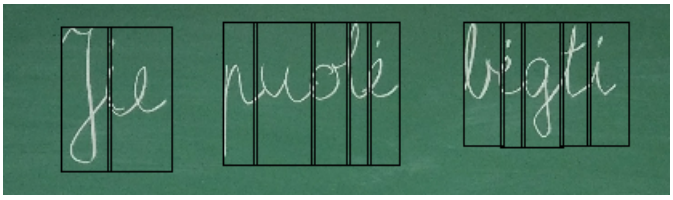
Ne visi bandymai buvo tokie sėkmingi. Dalis jų atskleidė algoritmo trūkumus. Analizuojant atvaizdą *test12.png* į iteruojamą kontūrų masyvą įtraukti žodžiai „Lindo“, „kišenę“, tačiau neįtrauktas „į“.



20 pav. Neteisingai atmestas kontūras, kurio viduje yra žodis.

Kontūras neįtrauktas į teksto atpažinimo algoritmą, nes yra per mažo dydžio. Taip atsitinka dėl to, kad nustatyti galimi žodžio dydžio parametrai neatitinka žodžio „į“ aukščio ir pločio.

Atliekant bandymus nustatyta, kad algoritmas neatpažįsta dalies raidžių, nors ir žodis segmentuojamas sėkmingai. Analizuojant *test14.png* paveikslėlį teisingai atpažįstamas tik žodis „bėgti“. KNN algoritmas neteisingai nustato reikšmes žodžiui „puolė“, nors jis ir suskirstytas tinkamai. Dėl šios priežasties paieška žodyne irgi vykdoma pagal neteisingą užklausą. Vietoj teisingo atsakymo gaunami variantai „ne“, „vėl“ ir „per“. Taigi, algoritmas ne visada veikia tinkamai.



21 pav. Bandymo metu žodžiai suskirstomi į raides tinkamai, bet atpažinimas nesėkmingas.

## 3.4 Eksperimento rezultatai

Eksperimento metu programos prototipo veikimas buvo išbandytas analizuojant 21 atvaizdą, kuriuose buvo užrašyta 100 atsitiktinai atrinktų žodžių iš sudaryto žodyno. Algoritmas teisingai atpažino 74 žodžius, kai iš žodyno paieškos buvo grąžinamas tik vienas žodis, labiausiai panašus į duotąjį. Septyni žodžiai buvo teisingai atpažinti, kai žodyno paieška grąžino du panašiausių žodžių variantus. Taip pat vienas žodis buvo tinkamai atpažintas, kai paieškos algoritmas grąžino 3 galimus variantus. Dėl algoritmo trūkumų apskritai nebuvo atpažinta 18 žodžių. Išsamūs rezultatai pateikiami 1-oje lentelėje.

1 lentelė. Eksperimento rezultatai.

| **Failo vardas** | **Atpažinta kaip:** | | | **Neatpažinta žodžių** | **Bendras žodžių skaičius** |
|---|---|---|---|---|---|
| | **1-as variantas** | **2-as variantas** | **3-as variantas** | | |
| test1.png | 2 | 0 | 0 | 1 | 3 |
| test2.png | 2 | 0 | 1 | 0 | 3 |
| test3.png | 3 | 0 | 0 | 0 | 3 |
| test4.png | 3 | 0 | 0 | 0 | 3 |
| test5.png | 2 | 1 | 0 | 3 | 6 |
| test6.png | 8 | 0 | 0 | 1 | 9 |
| test7.png | 3 | 0 | 0 | 0 | 3 |
| test8.png | 2 | 0 | 0 | 1 | 3 |
| test9.png | 2 | 0 | 0 | 0 | 2 |
| test10.png | 7 | 0 | 0 | 1 | 8 |
| test11.png | 2 | 0 | 0 | 0 | 2 |
| test12.png | 2 | 0 | 0 | 1 | 3 |
| test13.png | 7 | 1 | 0 | 1 | 9 |
| test14.png | 1 | 0 | 0 | 2 | 3 |
| test15.png | 6 | 2 | 0 | 1 | 9 |
| test16.png | 3 | 0 | 0 | 0 | 3 |
| test17.png | 1 | 1 | 0 | 1 | 3 |
| test18.png | 5 | 0 | 0 | 1 | 6 |
| test19.png | 2 | 0 | 0 | 1 | 3 |
| test20.png | 6 | 1 | 0 | 2 | 9 |
| test21.png | 5 | 1 | 0 | 1 | 7 |
| **Viso:** | **74** | **7** | **1** | **18** | **100** |

Buvo apskaičiuotas vidurkis, kokiu tikslumu vidutiniškai atpažįstami lietuvių kalbos žodžiai iš šio darbo autoriaus pasirinkto 250 žodžių žodyno. Tam naudota ši euristinė formulė:

$$E(X_n) = \frac{x_1 + \frac{1}{2}x_2 + \frac{1}{3}x_3}{100}$$

$x_1$ – teisingai atpažintų žodžių skaičius, kai grąžinamas tik vienas variantas.

$x_2$ – teisingai atpažintų žodžių skaičius, kai grąžinami du galimi variantai.

$x_2$ – teisingai atpažintų žodžių skaičius, kai grąžinami trys galimi variantai.

Formulėje prie $x_2$ ir $x_3$ kintamųjų nurodyti koeficientai $\frac{1}{2}$ ir $\frac{1}{3}$. Jie suteikia skirtingas vertes algoritmo grąžinamiems atitinkamai dviems ir trims panašiausiems žodžiams. Taip yra todėl, jog laikoma, kad algoritmui grąžinant du žodžių variantus ir atsitiktinai renkantis vieną teisingą – teisingo žodžio gavimo tikimybė yra lygi $\frac{1}{2}$. Analogiškai, tikimybė yra lygi

$\frac{1}{3}$, kai grąžinami trys variantai.

Naudojant bandymų rezultatus apskačiuota, kad tikimybinis vidurkis, jog žodis bus atpažintas teisingai yra 0.7783.

# Išvados

Šiame darbe buvo apžvelgti kompiuterinės regos praktiniai taikymai. Pristatyta populiari ir plačiai naudojama bei daug funkcijų turinti kompiuterinės regos biblioteka OpenCV. Taip pat trumpai apžvelgtos jos funkcijos, naudojamos vaizdų apdorojimui ir mašininiam mokymui. Buvo suformuluotas darbo tikslas – atpažinti lietuvių kalbos rankraštį. Minėtas tikslas praktinėje dalyje buvo sprendžiamas sukuriant lietuvių kalbos žodžių atpažinimo algoritmą, parašytą Python programavimo kalba. Algoritmo įgyvendinimui naudotos OpenCV bibliotekos vaizdų apdorojimo, mašininio mokymo funkcijos. Taip pat – Python programavimo kalbos numpy, os, sys, difflib, time paketai. Buvo sukurtas ir mašininiam mokymui panaudotas mokymosi duomenų rinkinys, sudarytas iš lietuvių kalbos raidžių atvaizdų.

Algoritmo veikimas buvo išbandytas atliekant praktinį eksperimentą. Atvaizdai, naudoti eksperimente gauti keliose Vilniaus Universiteto Matematikos ir Informatikos fakulteto auditorijose. Tuose atvaizduose užfiksuoti auditorijos lentoje užrašyti lietuviško teksto fragmentai. Pritaikius KNN mašininio mokymo algoritmą ir tikslinamąją paiešką riboto dydžio žodyne programos prototipas teisingai atpažino žodžius tikimybiniu vidurkiu 0.7783.

Atliekant praktinį eksperimentą išryškėjo ir programos prototipo trūkumai. Šiuo metu algoritmas tinkamai veikia tik su darbo autoriaus parašytu lietuvišku rankraščiu. Taip pat analizuojami atvaizdai turi būti užfiksuoti esant pastoviam dirbtiniam apšvietimui. Žodžių suskirstymas į raides ir pačių raidžių identifikavimas ne visada vyko tiksliai dėl sąlyginai nedidelio mokymosi duomenų rinkinio dydžio ir pasirinkto sąlyginai nesudėtingo mašininio mokymo KNN algoritmo.

Sukurto programos prototipo praktinio panaudojimo potencialas yra didelis, todėl ateityje jį galima tobulinti. Geresniam algoritmo veikimui reikalingas bent keliolika kartų didesnis mokymosi duomenų rinkinys. Taip pat galima pakeisti KNN mašininio mokymo algoritmą į efektyvesnį, pavyzdžiui, neuroninį tinklą. Kitas rankraščio atpažinimo tikslumo pagerinimo būdas gali būti papildomas mašininio mokymo naudojimas žodžių atpažinimui arba raidžių susijungimo vietoms aptikti taip patikslinant žodžių skirstymą į raides. Algoritmo panaudojimo galimybes išplėstų ir lankstesnis vaizdo apdorojimas. Tokiu būdu būtų panaikintas reikalavimas fiksuoti atvaizdus specifinėmis sąlygomis. Taip pat programai reikalinga grafinė vartotojo sąsaja norint priimtiniau atvaizduoti rankraščio atpažinimo rezultatus. Prototipas gali būti panaudotas ir kaip pagrindas tokiu pat principu veikiančiai išmaniųjų telefonų programai sukurti.

Šis baigiamasis bakalauro darbas parodo naudingą kompiuterinės regos taikymo galimybę ugdymo įstaigose ar kitose srityse, kur būtina vėlesniam laikui išsaugoti ranka užrašytą informaciją. Tokiu būdu galima išvengti būtinybės užrašus rašyti ranka ir taip pat sutaupyti laiko – palengvinti mokymosi bei darbo procesus.

# Literatūra


[1] G. Bradski ir A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*, O'Reilly Media, Inc., 2008, p. 1–8.

[2] N. Linder ir kt., *A malaria diagnostic tool based on computer vision screening and visualization of Plasmodium falciparum candidate areas in digitized blood smears*, PLoS One 9.8 (2014): e104855.

[3] E. Christos-Nikolaos Anagnostopoulos ir kt., *License plate recognition from still images and video sequences: A survey*, IEEE Transactions on intelligent transportation systems 9.3 (2008), p. 377-391.

[4] J. Pradeep, E. Srinivasan ir S. Himavathi, *Diagonal based feature extraction for handwritten character recognition system using neural network*, Electronics Computer Technology (ICECT), 2011 3rd International Conference IEEE, 2011.

[5] S. Lu ir kt., *Cost-sensitive neural network classifiers for postcode recognition*, International Journal of Pattern Recognition and Artificial Intelligence 26.05 (2012): 1263001.

[6] C. Chen ir kt., *A cost-effective people-counter for a crowd of moving people based on two-stage segmentation*, Journal of Information Hiding and Multimedia Signal Processing 3.1 (2012): 12-25.

[7] Y. LeCun ir kt., *Comparison of learning algorithms for handwritten digit recognition*, International conference on artificial neural networks. Vol. 60. 1995, http://yann.lecun.com/exdb/publis/pdf/lecun-95b.pdf.

[8] D. Kostadinov, *Privacy Implications of Automatic License Plate Recognition Technology*, http://resources.infosecinstitute.com/privacy-implications-automatic-license-plate-recognition-technology/.

[9] Shujing Lu ir kt., *Cost-sensitive neural network classifiers for postcode recognition*, International Journal of Pattern Recognition and Artificial Intelligence, 2012.

[10] OpenCV kompiuterinės regos bibliotekos žiniatinklio puslapis, http://opencv.org/about.html.

[11] OpenCV bibliotekos morfologinės transformacijos, http://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html.

[12] OpenCV bibliotekos kontūrų radimo funkcijų aprašymas, http://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html.

[13] OpenCV spalvų erdvės transformacijos, http://docs.opencv.org/3.2.0/df/d9d/tutorial_py_colorspaces.html.

[14] A. Ford ir A. Roberts, *Colour Space Conversions*, 1998, http://sites.biology.duke.edu/johnsenlab/pdfs/tech/colorconversion.pdf.

[15] Atvaizdo slenkstinės vertės nustatymas, http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html.

[16] KNN algoritmo paaiškinimas, http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html#knn-understanding.

[17] MNIST ranka rašytų skaičių duomenų rinkinys, http://yann.lecun.com/exdb/mnist/.

[18] MNIST duomenų iliustracija, http://neuralnetworksanddeeplearning.com/images/mnist_2_and_1.png.

[19] Dabartinės lietuvių kalbos žodynas, http://lkiis.lki.lt/dabartinis.

[20] Python numpy paketo vartotojo gidas, https://docs.scipy.org/doc/numpy-dev/user/quickstart.html.

[21] Python sys paketo dokumentacija, https://docs.python.org/3/library/sys.html.

[22] Python os paketo dokumentacija, https://docs.python.org/3/library/os.html.

[23] Python difflib paketo dokumentacija, https://docs.python.org/2/library/difflib.html.

# Santrauka

Bakalauro darbe „Lietuviško rankraščio atpažinimas ir taikymas“ apžvelgta ir pristatyta kompiuterinės regos ir mašininio mokymo biblioteka OpenCV bei jos funkcijos. Darbe buvo siekiama sukurti programos prototipą, kuris atpažįsta lietuvių kalbos žodžius, parašytus ranka. Prototipas pritaikytas atpažinti universiteto auditorijoje lentoje parašytą tekstą ir vartotojui pateikti rezultatą kompiuterio ekrane.

Darbo tikslas buvo pasiektas. Iš pradžių sukurtas mokymosi duomenų rinkinys iš lietuviškų raidžių atvaizdų. Ruošiant vaizdą analizei, prototipo algoritmas naudoja įvairias vaizdo apdorojimo funkcijas. KNN mašininio mokymo algoritmas naudojamas raidžių identifikavimui, o Python difflib paketo funkcijos naudojamos norint patikslinti, koks žodis atitinka atpažintą raidžių seką.

Buvo atliktas praktinis eksperimentas, kurio metu išanalizuotas prototipo veikimo tikslumas. Taip pat buvo išsiaiškinti prototipo trūkumai bei galimi patobulinimai.

Raktiniai žodžiai: kompiuterinė rega, mašininis mokymas, lietuviško rankraščio atpažinimas, vaizdo apdorojimas, Python, OpenCV, difflib, KNN, mokymosi duomenų rinkinys.

# Summary

A computer vision and machine learning library OpenCV is presented and reviewed in the bachelor thesis „Lithuanian handwriting recognition and its application“. The main goal of the thesis is to create a prototype of a program, which is capable to recognize a lithuanian handwriting. The prototype is adapted to recognize a text written on a green board in a classroom of the university and it shows the result on the screen of the computer.

The goal was reached. In the beginning, the learning data set was created. It consists of images of the lithuanian alphabet letters. The prototype uses various image processing functions to prepare the image for the analysis. A KNN machine learning algorithm is used to identify letters and a Python difflib package helps to match a word with the recognized sequence of letters.

There was a practical experiment done in order to analyse the accuracy of the prototype. Also, the weaknesses of the prototype were discovered and the possible improvements for the prototype were presented.

Keywords: computer vision, machine learning, lithuanian handwriting recognition, image processing, Python, OpenCV, difflib, KNN, learning data set.

# Priedas

## Programinės įrangos naudojimo instrukcijos

Šio darbo elektroninė versija, programos prototipo failai ir eksperimente bei mokymo duomenų ruošime naudoti atvaizdai pateikiami elektroninėje laikmenoje kartu su šiuo darbu. Visi programiniai failai bei jų versijos pasiekiamos ir žiniatinklyje, adresu: https://github.com/vytjan/final-ml.

Programos prototipo veikimas išbandytas naudojant kartu pateiktus atvaizdus ir šias programinės įrangos komponentų versijas:

- Operacinė sistema – Linux Ubuntu 16.04 versija.
- Python 3.5.2 versija.
- Python NumPy paketo 1.11.0 versija.

Darbo failus sudaro „src“ ir „doc“ aplankai. Aplanke „doc“ yra šio rašto darbo *pdf* ir *tex* formato failai, o „images“ aplanke – darbe naudojamos iliustracijos. *tex* formato failas kompiliuojamas į *pdf* dokumentą vykdant komandą:
*pdflatex -synctex=1 -interaction=nonstopmode*
*–shell-escape "Bakalauro_darbas_Vytautas_Jankauskas".tex*

„src“ aplanke patalpinti darbo praktinėje dalyje naudoti failai. „letters-50“ aplanke patalpintas mokymosi duomenų rinkinys, sukurtas naudojant *classification.py* programą. Programa *learningset.py* sugeneravo *generalsamplesNEW.data* ir *generalresponsesNEW.data* failus. Pagal juos apmokomas KNN algoritmas. *dictionary.data* faile yra darbo autoriaus sudarytas 250 žodžių žodynas. *lettertags.data* faile surašytos lietuvių kalbos abėcėlės raidės, naudojamos mokymo duomenų žymių dekodavimui. Eksperimente naudotas *testBoard.py* algoritmas, kuriame analizuoti „test_images“ direktorijoje esantys atvaizdai. Bandymų rezultatai įrašyti *statistika.ods* faile.

Programiniai failai vykdomi komandinėje eilutėje. Programos langas, vaizduojantis atvaizdą, išjungiamas grįžimo klavišu (angl. escape). Programų paleidimo instrukcijos (esant „src“ direktorijoje):

- Mokymosi duomenų ruošimo programos argumentas yra norimo *png* formato atvaizdo vardas iš „learning_data“ aplanko:
  *python3 classification.py learning_data/learn1.png*
- Duomenų rinkinio vertimas į tekstinius failus:
  *python3 learningset.py*
- Eksperimente naudotos žodžių atpažinimo programos argumentas yra analizuojamo *png* formato atvaizdo vardas iš „test_images“ aplanko:
  *python3 testBoard.py test_images/test1.png*