VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
MATEMATINĖS INFORMATIKOS KATEDRA

Vytautas Jankauskas *parašas*
Bioinformatikos studijų programa

Lietuviško rankraščio atpažinimas ir taikymas

Bakalauro baigiamasis darbas

Vadovas: lektorius **Irus Grinis** *parašas*

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.

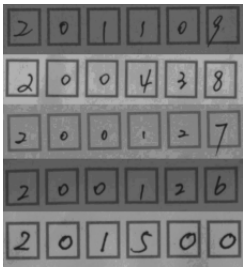
Kompiuterinis vaizdinės medžiagos apdorojimas išsikelto tikslo įvykdymui yra vadinamas kompiuterine rega [1]. Ji iš esmės turi tokią pat paskirtį kaip ir žmogaus rega. Tačiau kompiuteriai, skirtingai nei žmonės, vaizdinę medžiagą supranta tik kaip ilgą skaičių seką. Dirbtinės regos sistemos neturi jokio objektų atpažinimo modelio, nežino kurioje vietoje reikėtų fokusuoti vaizdą ar kurias jo dalis ignoruoti. Taip pat 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 [7].

- Vienas iš seniausių kompiuterinės regos taikymų yra pašto indeksų atpažinimas ant vokų ir siuntinių, leidžiantis sutaupyti daug laiko paštų darbuotojams [8].



2 pav.: Pašto indeksų pavyzdžiai iš Kinijos pašto.

- 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 [5].
- 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 bakalaurinio 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 įrankiais naudojamais vaizdo normalizavimui, triukšmo pašalinimui.
- Susipažinti su elementariais mašininio mokymosi įrankiais ir algoritmais.
- Sukurti mokymosi duomenų aibę ir su ja apmokyti teksto atpažinimo algoritmą.

- Sukurti 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 įgyti bazines žinias 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 Kompiuterinės regos biblioteka OpenCV

Pagrindinis šio bakalauro darbo įrankis naudotas programos prototipo kūrimui yra OpenCV biblioteka [9]. 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 daugiausia skirti vaizdų apdorojimui, taip pat mašininiam mokymui. Nors ir parašyta C++ programavimo kalba, OpenCV turi C, Python, Java ir MATLAB kalbų sąsajas taip dar palengvindama naudojimąsi. Taip pat palaikomos Windows, Linux, Mac OS ir Android operacinės sistemos. Daugiausiai 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 [11]. 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 erdvė aprašomos spalvos pagal jose esantį pilkos spalvos kiekį ir pagal jų šviesumo stiprumą. Dar kompiuterinėje regoje dažnai naudojama ir pilkumo tonų (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 pilkumo tonų spalvų erdvėje. Jame, pagal pasirenkamą arba algoritmo nustatomą spalvos vertę kiekvienas taškas įgauna nulio arba vieneto vertę. Tai reiškia, kad už slenkstinę vertę didesnę vertę turintys taškai įgys juodos spalvos reikšmę, ir atvirkščiai.

3 pav.: Pilkumo tonų atvaizdas kairėje, slenkstinis – viduryje. Invertuotas slenkstinis – dešinėje [12].

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

```
# 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).

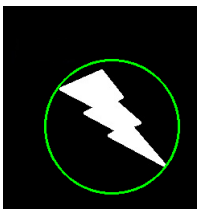
  Ardymo operacija veikia panašiai kaip ir dirvožemio ardymas, tik ji ardo atvaizdo priekinio plano objekto kraštus. Struktūrinis elementas slenkamas per paveikslėlio taškus. Originaliam paveikslėlio taškui (turinčiam vertę 0 arba 1) bus priskiriama vertė 1 tik tuo atveju, jei visi taškai po struktūriniu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

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 išplečiame priekinio vaizdo objekto plotą. 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 bet triukšmas nebeatsiranda iš naujo.

5 pav.: Kairėje – originalus atvaizdas. Dešinėje – po išplėtimo operacijos.

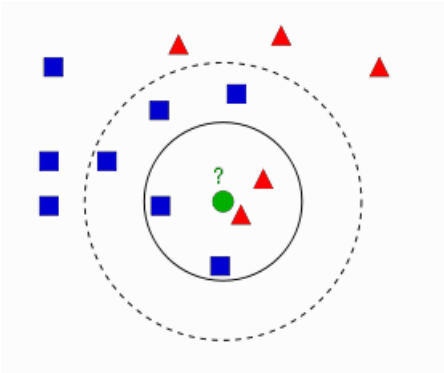
- Kontūrų radimas kompiuterinėje regoje dažniausiai naudojamas aptinkant įvairius objektus, juos atpažįstant ar nustatant jų formas. Kontūrą galima apibūdinti kaip visus vienodos spalvos ar intensyvumo taškus (aplink jų pakraščius) apibrėžiančią liniją. Norint gauti tikslesnius objekto kontūrus, rekomenduojama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



6 pav.: Kontūrų aptikimas binariniame atvaizde.

- 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 [13]. Pav. 4 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.

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. 4 grafike esantį žalią apskritimą pagal 4 artimiausius kaimynus, jis būtų priskirtas raudonųjų trikampių klasei, kadangi šie yra arčiau nei mėlynieji kvadratai.

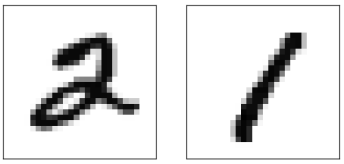
2 Problemos sprendimas

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.dat* – tekstinis failas. Jame surašyti žodžiai, kuriuos algoritmas gali atpažinti. Naudojamas teksto atpažinimo patikslinimui.
- *lettertags.txt* – 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 lietuvišką ranka parašytą 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, kokia klasei elementas yra priskirtas. Vienas žinomiausių tokių rinkinių yra MNIST ranka rašytų skaičių rinkinys. Jį sudaro 60 000 mokymo elementų ir 10 000 elementų, skitų testavimui [14]. 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.

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 [15].

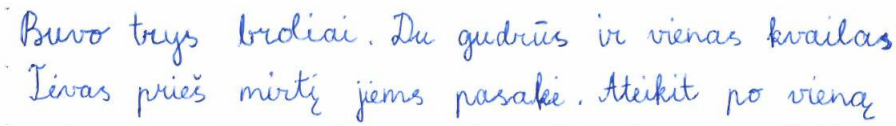
- Kiekviena raidė yra parašyta šio darbo autoriaus. Šis apribojimas įvestas, nes planuojama rinkinio apimtis nebus didelė, tad ir teksto rašymo stilius turi būti panašus visuose elementuose.

- 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 pilkumo tonų. 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)
```

xxx 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)
```

xxx 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)
```

xxx 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 kontūro 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 viena 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:
2.5500000000000000e+02 2.5500000000000000e+02 2.5500000000000000e+02 ...
# Atvaizdo žymos skaitinė reikšmė:
5.3000000000000000e+01
```

xxx kodo fragmentas. Galutinis mokymosi duomenų formatas.

2.2 Vaizdo apdorojimas

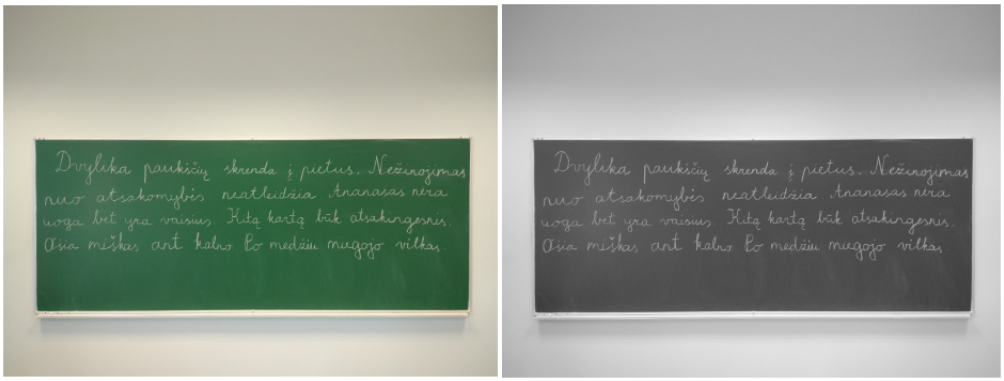
Pirmas žingsnis programos kūrime buvo surinkti kokybišką vaizdinę medžiagą. Todėl šiame darbe naudoti atvaizdai buvo fiksuojami esant geroms, pastovioms apšvietimo sąlygoms. Tokiu būdu norėta išvengti papildomo triukšmo paveikslėliuose. Taip pat buvo stengiamasi išrinkti pakankamai kontrastingus priekinio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

2 kodo fragmentas. Atvaizdo įkėlimas į programą.

Kitame žingsnyje naudojama spalvų erdvės transformacija iš RGB erdvės į pilkumo tonų. Tokiu būdu gaunamas juodai baltas paveikslėlis.



12 pav.: Kairėje – paveikslėlis RGB erdvėje, dešinėje – pilkumo tonų.

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.dat* faile. Jį sudaro atsitiktinai parinkti sintaksiškai nesudėtingi lietuvių kalbos žodžiai, kurie nebūtinai sudaro prasmingus žodžių junginius. Žodynas algoritme naudojamas kiekvieno atpažinto žodžio patikslinimui.

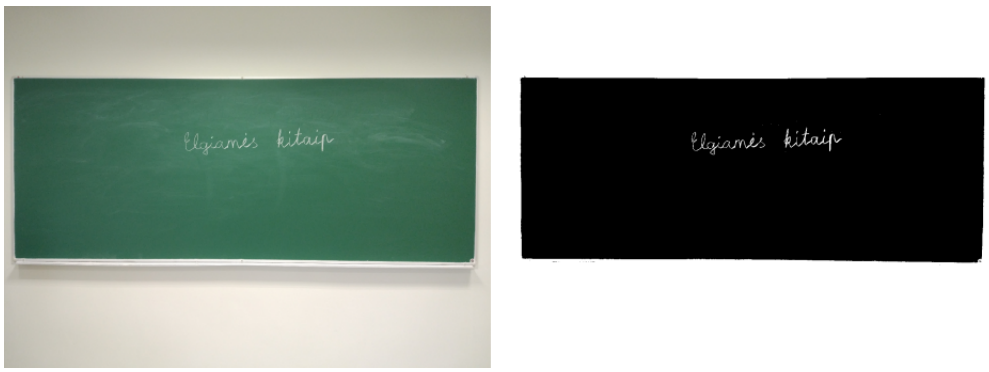
Vykdant programą pirma atliekamas mašininio mokymo kNN algoritmo apmokymas. Tam naudojamos OpenCV bibliotekoje esančios algoritmo mokymo modelio sukūrimo ir apmokymo funkcijos.

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

xxx 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 pilkumo tonų 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 atkrepamas 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)
```

xxx kodo fragmentas. KNN algoritmo artimiausio kaimyninio elemento paieška.

- Grįžtama prie to paties žodžio kontūro. Padidinamas apkrepamo atvaizdo plotis ir aukštis ir 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 mažiau-

sią skaitinę vertę turintis 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 atlikta keletas eksperimentų jo veikimo išbandymui. Nuspręsta, kad prototipo tikslumas bus tikrinamas naudojant atitinkamomis sąlygomis gautus atvaizdus su juose parašytais lietuviškais žodžiais.

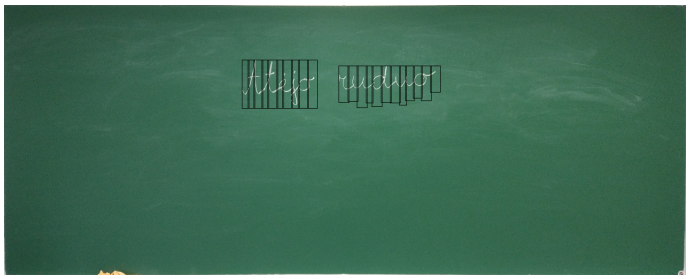
Pasirinktos eksperimento sąlygos – universiteto auditorija, kurioje yra pastovus dirbtinis apšvietimas. Taip pat norint gauti kontrastingą vaizdą tekstas balta kreida užrašomas tamsiai žalioje lentoje. Lentą supa šviesus fonas.

14 pav.: Eksperimentinio atvaizdo pavyzdys.

Eksperimentui įvykdyti iš sudaryto žodyno atsitiktine tvarka buvo pasirinkti 100 žodžių. Iš jų sudaryti įvairūs sakiniai ar žodžių junginiai. Jie buvo užrašomi ant žalios lentos įvairia tvarka ir kiekiu. Taip gauti reikalingi atvaizdai.

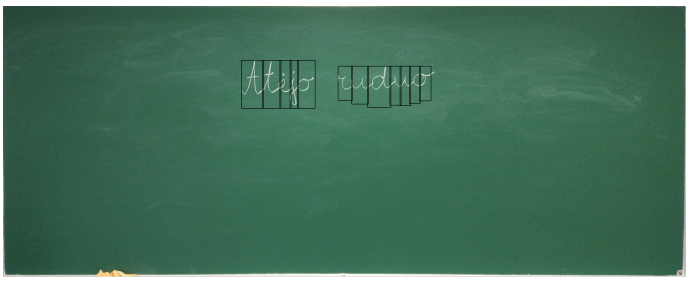
3.1 Tinkamų parametrų nustatymas

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.

!!!!!!!!!!!!!!!! po visais paveikslėliais sudėti šaltinius.

3.3 B2 pavadinimas

Poskyris B2 turi du skirsnius

3.3.1 B21 pavadinimas

Tekstas....

3.3.2 B22 pavadinimas

Tekstas....

3.4 B3 pavadinimas

Tekstas su nauja formule $y = x^3$...

4 Išvados

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] 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.

[6] Y. LeCun ir kt., *Comparison of learning algorithms for handwritten digit recognition*, Tarptautinė neuroninių tinklų konferencija, 1995. http://yann.lecun.com/exdb/publis/pdf/lecun-95b.pdf

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

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

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

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

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

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

[13] 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

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

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

Santrauka

Trumpa darbo santrauka...

Summary

Short english summary...

Priedas Nr. 1

long stuff