UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET
ODSJEK ZA TELEKOMUNIKACIJE

Razvoj aplikacije za optičko prepoznavanje karaktera (OCR)

ZAVRŠNI RAD

- PRVI CIKLUS STUDIJA -

Student:
Almedina Kerla

Mentor:
Doc. dr Emir Sokić, dipl.ing.el.

Sarajevo,
septembar 2019.

Sažetak

U okviru ovog rada opisani su osnovni koraci za razvoj aplikacije za optičko prepoznavanje karaktera i testirane su mogućnosti već postojećeg OCR softvera *Tesseract-a*. Analizirane su osnovne metode obrade slike potrebne za rad OCR softvera i tipovi segmentacije koji se koriste kod ovih sistema.

Aplikacija omogućava prepoznavanje karaktera uz korištenje programskog jezika *C++* i biblioteka iz *OpenCV-a* i *Tesseract-a*. Testiranje je vršeno za nekoliko osnovnih kriterija određivanja kvalitete OCR aplikacije: brzina, preciznost, ograničenja u vidu dimenzija slike i raznovrsnosti fontova i osjetljivost na vanjske faktore poput osvjetljenja. Pomoću *Qt Creator-a* razvijen je i korinički interfejs koji omogućava jednostavnije praćenje rezultata.

Ključne riječi: OCR, prepoznavanje karaktera, Tesseract

Abstract

This paper describes the basic steps for developing an Optical Character Recognition application and the capabilities of Tesseract, an already existing OCR software, were tested. Basic image processing methods required for OCR software and the types of segmentation used in these system were analyzed

The character recognition application was developed using the *C++* programming language and libraries from *OpenCV* and *Tesseract*. Testing has been performed for several basic criteria for determining the quality of an OCR application: speed, precision, limitations in image dimensions and font variety, and sensitivity to external factors such as lighting. Using Qt Creator, a user interface has also been developed that makes it easier to track results.

Keywords: OCR, character recognition, Tesseract

Elektrotehnički fakultet, Univerzitet u Sarajevu
Odsjek za telekomunikacije
Doc. dr Emir Sokić, dipl.el.ing
Sarajevo, 04.03.2019.

Postavka zadatka završnog rada I ciklusa:
Razvoj aplikacije za optičko prepoznavanje karaktera (OCR)

Prilikom vizuelne inspekcije u industrijskom okruženju veoma često se nameće potreba za sistemom za automatsko prepoznavanje karaktera (eng. *optical character recognition* - OCR), kako bi se provjeravala ispravnost odgovarajućih natpisa na proizvodima ili izvršila klasifikacija (npr. provjera datuma ili serije). U okviru rada je potrebno napraviti aplikaciju za automatsko optičko prepoznavanje karaktera na osnovu digitalnih slika, i to na osnovu postojećih biblioteka i open-source aplikacija (OpenCV, Tessaract i sl.)

Postavka zadatka, koncept i metode rješavanja: U okviru rada potrebno je:

- dati osnovni pregled literature vezane za problem optičkog prepoznavanja karaktera,
- opisati uobičajene pristupe u rješavanju problema OCR-a
- dati pregled postojećih komercijalnih/besplatnih biblioteka/aplikacija za OCR,
- izvršiti eksperimentalnu analizu (vrijeme i tačnost izvršavanja) nad odgovarajućim OCR implementacijama,
- testirati OCR sistem u realnom okruženju (ETFCam).

Polazna literatura:

[1] Batchelor, Bruce G., and Paul F. Whelan. Intelligent vision systems for industry. Springer Science & Business Media, 2012.

[2] Torras, Carme, ed. Computer vision: theory and industrial applications. Springer Science & Business Media, 2012.

[3] Demant, Christian, C. Demant, and Bernd Streicher-Abel. Industrial image processing. Springer-Verlag, 1999.

[4] E.R. Davies. Computer vision: principles, algorithms, application, learning. Academic Press, 2017.

[5] Sonka, Milan, Vaclav Hlavac, and Roger Boyle. Image processing, analysis, and machine vision. Cengage Learning, 2014.

[6] Kim L. Boyer, Sudeep Sarkar. Perceptual organization for artificial vision systems, Springer Science & Business Media, 2000.

[7] A.Chaudhuri, K. Mandaviya. Optical character recognition systems for different languages with soft computing. Springer ,2017.

[8] Stephen V. Rice, George Nagy, Thomas A. Nartker. Optical character recognition: an illustrated guide to the Frontier. Springer Science & Business Media, 1999.

[9] M.Cheriet, N.Kharma, C. Liu, C. Suen. Character recognition systems. Wiley - Interscience, 2007.

[10] Mori S., Nishida H., Yamada H., Optical character recognition. Wiley – Interscience, 1999.

Doc. dr Emir Sokić, dipl.el.ing

Univerzitet u Sarajevu
Elektrotehnički fakultet
Odsjek za telekomunikacije

Izjava o autentičnosti radova

Završni rad
I ciklusa studija

Ime i prezime: Almedina Kerla
Naslov rada: Razvoj aplikacije za optičko prepoznavanje karaktera (OCR)
Vrsta rada: Završni rad Prvog ciklusa studija
Broj stranica: 41

Potvrđujem:

- da sam pročitao dokumente koji se odnose na plagijarizam, kako je to definirano Statutom Univerziteta u Sarajevu, Etičkim kodeksom Univerziteta u Sarajevu i pravilima studiranja koja se odnose na I i II ciklus studija, integrirani studijski program I i II ciklusa i III ciklus studija na Univerzitetu u Sarajevu, kao i uputama o plagijarizmu navedenim na web stranici Univerziteta u Sarajevu;
- da sam svjestan univerzitetskih disciplinskih pravila koja se tiču plagijarizma;
- da je rad koji predajem potpuno moj, samostalni rad, osim u dijelovima gdje je to naznačeno;
- da rad nije predat, u cjelini ili djelimično, za stjecanje zvanja na Univerzitetu u Sarajevu ili nekoj drugoj visokoškolskoj ustanovi;
- da sam jasno naznačio prisustvo citiranog ili parafraziranog materijala i da sam se referirao na sve izvore;
- da sam dosljedno naveo korištene i citirane izvore ili bibliografiju po nekom od preporučenih stilova citiranja, sa navođenjem potpune reference koja obuhvata potpuni bibliografski opis korištenog i citiranog izvora;
- da sam odgovarajuće naznačio svaku pomoć koju sam dobio pored pomoći mentora i akademskih tutora/ica.

Sarajevo, datum .....

Potpis:

_____

Almedina Kerla

Sadržaj

Popis slika

Popis tabela

Poglavlje 1

Uvod

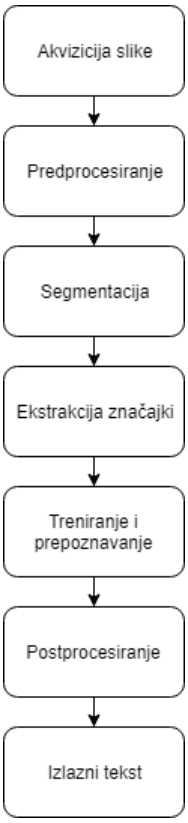
Optičko prepoznavanje znakova (OCR) proces je klasifikacije optičkih uzoraka sadržanih u digitalnoj slici koji odgovaraju alfanumeričkim ili drugim znakovima. Ova tehnologija omogućava automatsko prepoznavanje znakova kroz optički mehanizam. U slučaju ljudskih bića, naše oči su optički mehanizam. Slika koju ljudi vide očima je ulaz za mozak. OCR je tehnologija koja funkcioniše poput ljudske sposobnosti čitanja. Učinkovitost OCR-a direktno ovisi o kvaliteti ulaznih dokumenata. OCR je dizajniran za obradu slika koje se gotovo u potpunosti sastoje od teksta, sa vrlo malo netekstualnih dijelova dobivenih na slici snimljenoj kamerom.

Potrebe za optičkim prepoznavanjem karaktera se javljaju iz perioda kada digitalizacija nije bila razvijena kao danas. Usluge OCR-a početkom ovog stoljeća su se prvobitno plaćale i sam softver je bio veoma skup. U proteklih 20 godina, OCR tehnologija se brzo pokazala važnom u mnogim industrijama, medicini, računovodstvu i slično. Razlog za to je bila potreba pretraživanja slika ili skeniranih dokumenata. Najčešću primjenu OCR sistemi nalaze u bankarskim aplikacijama. Sisteme karakteriše čitanje ograničenog niza znakova, obično brojeva i nekoliko posebnih simbola. Dizajnirani su za čitanje podataka poput brojeva računa, identifikacije kupaca, brojeva članaka, iznosa novca itd. [3]

U uvodnom dijelu rada bit će objašnjeni osnovni principi rada svih sistema za optičko prepoznavanje karaktera i opća struktura neovisno da li se koristio dodatni softver koji već ima implementirane neke od ovih faza. Također, bit će prikazano poređenje različitih OCR i istaknute prednosti i mane svakog od njih. Na osnovu dobijenih rezultata i literature OCR sistem koji će se razvijati u pozadini je baziran na Tesseract-u. S obzirom da je osnovni parametar za procjenu kvaliteta bilo kojeg OCR softvera tačnost, poglavlje 2 objašnjava najčešće uzroke nastanka grešaka, te kako takve greške izbjeći. Jedan od načina kako smanjiti količinu grešaka je upotreba odgovarajućih tehnika predprocesiranja. Predprocesiranje slike koje će se detaljnije obrađivati su ekvalizacija histograma, binarizacija, uklanjanje šuma, korekcija nagiba, dilatacija. Idući korak u sklopu prepoznavanja karaktera čini segmentacija. Osnovna obilježja koja su karakteristična za segmentaciju u sklopu OCR aplikacija bit će prikazana u poglavlju 4. Rad također uključuje i praktični dio, čija realizacija je detaljnije objašnjena u poglavlju 5.

1.1 Optičko prepoznavanje karaktera

OCR tehnologija nam omogućuje pretvorbu različitih vrsta dokumenata kao što su skenirani papirni dokumenti, pdf datoteke ili slike snimljene digitalnim fotoaparatom u podatke koji se mogu uređivati i pretraživati. Slike snimljene digitalnim fotoaparatom razlikuju se od skeniranih dokumenata ili slika, često imaju oštećenja kao što su izobličenje na ivicama i prigušeno svjetlo, što otežava većini OCR aplikacija da pravilno prepoznaju tekst. [6]. OCR softver koji



Slika 1.1: Strukrura OCR aplikacija

će se koristiti za prepoznavanje karaktera u ostatku rada je Tesseract. Tesseract je odabran zbog mogućnosti proširivanja baze podataka (treniranjem) i fleksibilnosti. Za uspješno prepoznavanje karaktera potrebno je izvršiti sve faze prikazane na slici 1.1, odnosno akvizicija slike, predprocesiranje, segmentacija, izdvajanje značajki, treniranje i prepoznavanje i postprocesiranje kako bi na izlazu iz aplikacije dobili tekst.

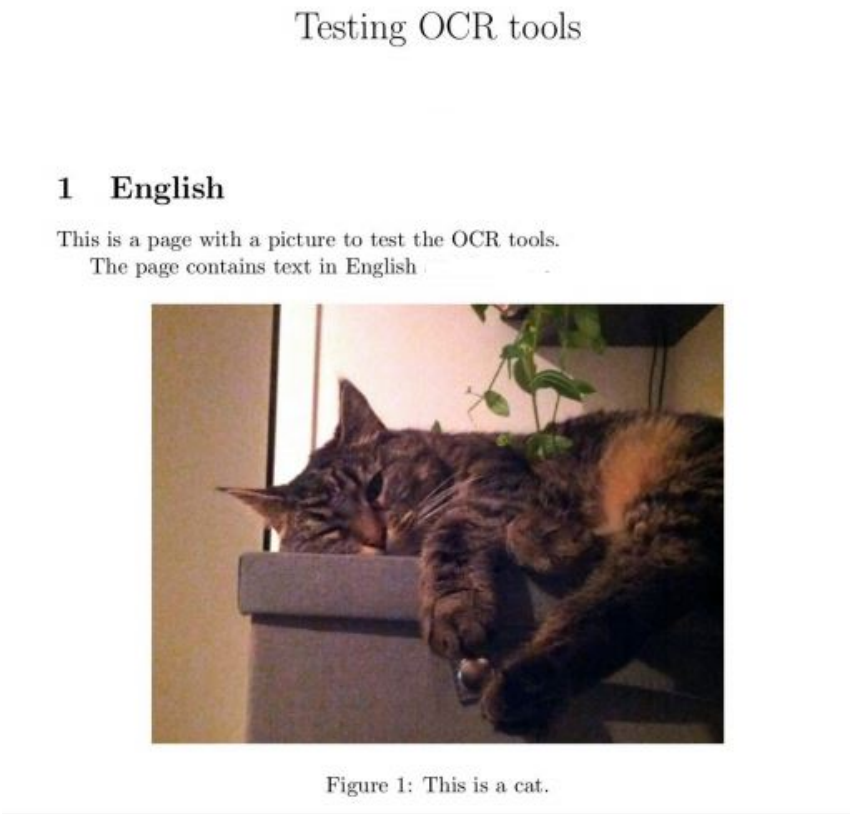
OCR sistemi postali su jedni od najuspješnijih aplikacija tehnologije u prepoznavanju uzoraka i području umjetne inteligencije. Iako postoje mnogi komercijalni sistemi za izvođenje OCR-a za različite aplikacije, raspoloživi sistemi još uvijek nisu u mogućnosti da se takmiče sa mogućnostima ljudskog čitanja sa željenim stepenom tačnosti.

1.2 Poređenje postojećih besplatnih OCR softvera

Postoji veliki broj softvera za optičko prepoznavanje karaktera. Dva faktora koja čine najveću razliku među njima su jednostavnost upotrebe i cijena. Neki od njih zahtijevaju složenije programiranje što ih ne plasira kao prvi izbor pri odabiru OCR softvera. Također, veliki broj OCR softvera je besplatno, samim time u ovom poglavlju obrađivat će se samo besplatni (eng. *open source*) alati. Tri besplatna softvera koja se najčešće koriste su: OCRopus, Tesseract i GOCR.

OCRocpus je OCR alat izdan pod licencom Apache. Prepoznavanje se može unaprijediti pomoću mašinskog učenja, gdje se OCRopus može obučiti za prepoznavanje novih fontova ili jezika.

Tesseract je OCR alat otvorenog koda koji je HP počeo razvijati 1985. HP je nastavio razvoj sve do 1995. godine, a otkako je 2005. godine izdan pod licencom Apache, razvoj je sponzorisao Google.[1] Tesseract može prepoznati tekst na preko 60 različitih jezika, koji se moraju


Testing OCR tools

1 English

This is a page with a picture to test the OCR tools.
The page contains text in English

Figure 1: This is a cat.

Slika 1.2: Testna slika za provjeru preciznosti OCR softvera[1]

preuzeti i dodati Tesseract-u ručno. Ako za skeniranje nije naveden nijedan jezik, Tesseract će kao zadani koristiti engleski jezik, ali može i koristiti više jezika pri svakom skeniranju. Tesseract podržava mašinsko učenje, koje se može koristiti ako jezik ili font koji se skenira ne postoje.

GOCR je OCR alat otvorenog koda koji se objavljuje u sklopu GNU GPL. Slike je potrebno konvertovati da bi se mogle koristiti s GOCR-om jer može pročitati samo nekoliko formata slike.

OCRopus daje najbolje rezultate za različite tipove slika (slika sa tekstom, iskrivljen tekst, rukopis, loše osvjetljenje, jak šum i slično), dok GOCR je najlošiji u skoro svim područjima. Problem kod OCRopus-a je sporo izvršavanje, što je uzrokovalo da danas prednost ima Tesseract. Najbolje rezultate u vidu procenta greške na testiranoj slici 1.2 je imao Tesseract, to se može vidjeti i u tabeli 1.1. [1]

Tabela 1.1: Poređenje OCR softvera

OCR alat	Operativni sistem	Programski jezik	Broj jezika	Količina grešaka
Tesseract	Windows, MAC, Linux	Python, C, C++	60+	2.08%
GOCR	Windows, MAC, Linux	C	20+	72.15%
OCRopus	MAC, Linux	Python, C++	Latinična slova	37.96%

Poglavlje 2

Uzroci nastanka grešaka

U posljednjem desetljeću značajno je porasla stopa upotrebe čitača za ručno i mašinski ispisane znakove. Rani OCR uređaji zahtijevali su skupe skenere i specijalni elektronički ili optički hardver: IBM-ov optički čitač stranica za čitanje otkucanih izvještaja o troškovima socijalne zaštite koštao je više od tri miliona dolara. U isto vrijeme javljaju se mikroprocesori za računare i skenere, [2] što je rezultiralo ogromnim smanjenjem troškova. Danas OCR softver je često dodatak skenerima koji koštaju otprilike kao pisač ili faks-uređaj.

Iako se OCR široko koristi, njegova tačnost još uvijek je daleko od idealne. Čak i 99% tačnosti znači 30 grešaka na stranici od 3000 znakova. Samim time, greške koje nastanu se moraju ručno ispravljati ili se dokument odbacuje. Greške možemo grupisati u četiri glavne kategorije u odnosu na uzroke njihovog nastanka:

- oštećenje slike,
- slični simboli,
- znakovi interpukcije,
- tipografija.

2.1 Oštećenje slike

Greške usljed oštećenja slike se mogu javiti i tokom procesa štampanja. Porozni papir uzrokuje širenje tinte. Prevučeni, sjajni papir ne upija mastilo ili toner i podložan je razmazivanju. Štampači velike brzine, kao oni koji se koriste za printanje novina, nerijetko mogu proizvesti nejasan print.[2] Kopiranje stranice, posebno na starijim uređajima za kopiranje, rezultira daljim gubitkom informacija. Čak i uz modernu tehnologiju kopiranja, kopiranje kopija brzo eskalira pogoršanje. Na slici 2.1 većina defekata u slikanju već je bila prisutna na papiru. Usljed različitog predprocesiranja dolazi do različitog prepoznavnja teksta kao: *:ite:*, *sitos* i *sitoa*.

Ipak, proces skeniranja uvodi svoje vlastite nesavršenosti, posebno u odvajanju teksta od pozadine. Količina svjetlosti koja se reflektuje od bijeli papir je samo oko dvadeset puta veća od svjetlosti koja se reflektuje od čvrsti, tamni papir. [2] Skeneri su mnogo osjetljiviji od ljudi na nizak kontrast i varijacije u pozadini. Mnogi OCR sistemi mogu podesiti prag skenera na osnovu preliminarnog skeniranja stranice, a neki čak mogu postaviti i različite vrijednosti za dijelove stranice. Na stranicama sa visokim kontrastom, izbor praga nije kritičan.

Skeniranje crno-bijele slike omogućava detaljniju analizu digitalnih uzoraka i smanjuje vjerovatnoću greške na materijalu sa slabim kontrastom. Također se može tvrditi da monohromatski prikaz povećava efektivnu prostornu rezoluciju skenera, te donosi niže vjerovatnoće grešaka

Slika 2.1: Primjer greške koja se javlja usljed printanja[2]

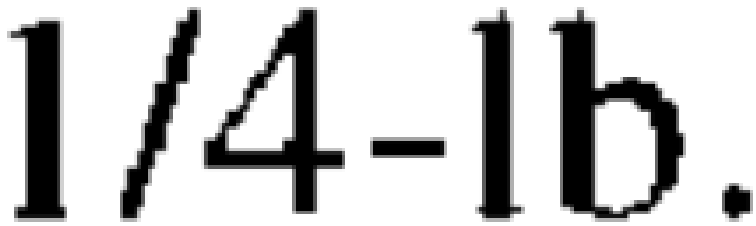
na visokom kontrastu, za razliku od usko razmaknutog teksta. Stranica se obično uzorkuje i horizontalno i vertikalno brzinom od 240 ili 300 tačaka po inču - dpi (eng. *dots per inch*). Danas se teži većem uzorkovanju, kao posljedica toga javlja se da neki OCR paketi mogu iskoristiti 400 dpi ili 600 dpi slika. Drugi samo poduzrokuju ili interpoliraju sliku na 300 dpi.

2.2 Slični simboli

Svi OCR uređaji prepoznaju znakove prvenstveno na osnovu njihovog oblika. Prvenstveno se posmatra cijeli dokument, odnosno šira slika a zatim se fokus stavlja na dijelove znakova sa sličnim oblicima gdje najčešće dolazi do konfuzije kod prepoznavanja. Među obilježjima oblika koji se koriste za razlikovanje slova su omjer širine i visine znakova, omjeri širina horizontalnih, vertikalnih i kosih linija, veličina i oblik serifa i tako dalje.[2]

Neki simboli izgledaju toliko slično da ih je teško razlikovati čak i kad su čisto ispisani i nemaju oštećenja na slikama. Na primjer, ako je nepoznati simbol izdužen i kružnog oblika, to bi mogao biti broj 0 ili slovo O. Ako je susjedni znak broj, veća je vjerovatnoća da je nula. Izbor između 1, L i I za nepoznati vertikalni simbol je posebno težak. Posebni simboli poput $ i % su moderni izumi koji dodatno komplikuju tačnost OCR sistema.

Ovaj problem je testiran na slici 2.2. Promjenom dimenzija bloka za adaptivnu binarizaciju dobijeni su različiti rezultati. Za blok dimenzije veće od 29 piksela tekst sa slike je ispravno prepoznat, odnosno kao *"1/4-lb."*, dok za dimenzije bloka 17 i 11 rezultati su *"1/4-Ib."* i *"\/4-Ib."* respektivno.



Slika 2.2: Primjer problema kod detekcije broja 1 [2]

Neke od grešaka se mogu tolerisati usljed psihovizualne redundanse, jer čitatelj svako slovo ili broj ne razmatra posebno. Dizajneri tipova fonta prave značajnu razliku između različitih simbola u istom slovu. Stoga je mnogo vjerojatnije da simbol nalikuje drugom simbolu iz druge porodice fontova, nego onome u vlastitom fontu.

2.3 Znakovi interpukcije

Kapitalizacija i interpunkcija su smjernice u pisanom materijalu. U narativnom i deskriptivnom tekstu oko 60% svih interpunkcijskih znakova sastoji se od tačaka i zareza, a zarezi su obilniji od tačaka. Međutim, u tehničkim materijalima broj tačaka je veći od zareza, jer se koriste i skraćenice i decimalni brojevi. [7] Male dimenzije interpukcijskih znakova dodatno unose problem kod ispravne detekcije. Usljed treniranja OCR sistema neki od znakova se razlikuju na osnovu pozicije u odnosu na osnovnu liniju teksta npr. apostrofi i zarezi.

2.4 Tipografija

Korisnik OCR-a nema kontrolu nad tipografijom i često se suočava s izgledima, tipovima pisama i veličinama slova koje su daleko od idealnog za OCR. Tipovi pisama posebno dizajnirani za OCR, kao što su OCR-A i OCR-B fontovi (slika 2.3), imaju uniformne širine linija i izražene razlike između sličnih simbola kao što su 0 i 0 ili 1 i l. [8]

Slika 2.3: Izgled fontova OCR-A i OCR-B [3]

Engleska abeceda ima samo 26 slova, ali se u ne-tehničkom materijalu najčešće koristi oko 80 znakova (velika i mala slova, brojevi, interpunkcija i neki posebni simboli). Svi ovi simboli su uključeni u skup od 96 znakova - ASCII (eng. *American Standard Code for Information Interchange*), koji je usvojen od strane američkih proizvođača računala.[3] Korištenje pogrešnog jezika za prepoznavanje teksta uzrokuje greške jer se detekcija vrši na osnovu cijele riječi, a ne pojedinačnih slova. Ukoliko se za prepoznavanje teksta sa slike 2.4 u Tesseract-u koriste jezici koji nisu engleski, dobijeni rezultati neće biti tačni. Stoga za portugalski jezik riječ koju dobijemo kao rezultat je $O]7IceoftLyeDe6m$, a za bosanski $Office\#theDean$.

Slika 2.4: Testirana rečenica na osjetljivost jezika [2]

Razmak između redova teksta također se izražava u tačkama. Set tipa 10/14 znači da između blokova veličine 10-pt (tačaka) postoji vertikalni razmak od 4-pt (tačke). Dobra čitljivost

i povećanje tačnosti OCR-a, zahtijeva najmanje 2 razmaka. Grafički sistem kodiranja koji koristimo za prikazivanje jezika je složen, ali dobro prilagođen ljudskim sposobnostima. Pisma koja ne koriste latinična slova još uvijek stvaraju velike probleme za OCR sisteme i zahtijevaju posebnu obradu, najčešće dodatno treniranje OCR softvera.

* * *

Kvalitet sistema za optičko prepoznavanje karaktera se mjeri na osnovu broja grešaka koje nastanu kao posljedica loše akvizicije slike ili predprocesiranja. Sistem ne može garantovati 100% tačnost s obzirom da ne može pretpostaviti kakva će biti ulazna slika. OCR softver poput Tesseract-a za neke slike može izvršiti ekstrakciju znakova bez greške.

Poglavlje 3

Predprocesiranje slike

Početni korak u digitalnoj obradi slike je predprocesiranje. Faza predprocesiranja ima za cilj izdvojiti relevantne tekstualne dijelove i pripremiti ih za segmentaciju i prepoznavanje. Prema tome, u ovom postupku ulazni i izlazni parametar je slika, odnosno izlazna slika je izmijenjena verzija originala i prilagođena je daljnjim postupcima. Osnovni zadatak obrade snimljenih podataka je smanjenje varijacija koje uzrokuju redukciju brzine prepoznavanja i povećanje kompleksnosti sistema. Važnost faze predprocesiranja leži u sposobnosti da ispravi probleme koji se mogu pojaviti usljed faktora poput: kvalitet skenera, rezolucija skeniranja, tip printanog dokumenta (laserski ili fotokopirano), font koji se koristi u tekstu, jezička kompleksnost i korišteni rječnik.

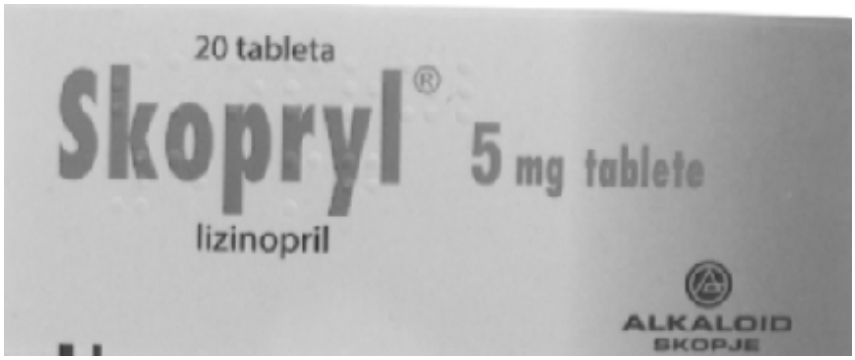
Prva faza optičkog prepoznavanja karaktera je akvizicija koja uključuje skeniranje ili digitalizaciju štampanih znakova u oblik pogodan za naknadne elektronske obrade uz minimalnu degradaciju. S obzirom da se snimljena slika može naći u raznim stanjima neke od tehnika koje je neophodno koristiti su: binarizacija, dilatacija, korekcija nagiba, uklanjanje šuma pomoću filtera i segmentacija stranice. Na slici 3.1 je prikazana originalna slika za koju će biti vršeno izdvajanje karaktera.

3.1 Ekvalizacija histograma

Kada je kontrast slike veoma nizak, ekvalizacija histograma predstavlja standardan postupak za poboljšanje njene kvalitete. Histogram predstavlja diskretnu funkciju koja je reprezentacija frekvencije pojavljivanja svih intenziteta na određenoj slici. Naime, kod slika koje imaju uzak histogram i ne zauzimaju čitav opseg mogućih intenziteta, postoji mnogo piksela sa sličnim vrijednostima te stoga nije moguće razaznati detalje, bilo da je to previše tamna, svijetla ili blijeda slika. Ekvivalizacija histograma teži automatskoj regulaciji kontrasta u vrlo svijetlim ili tamnim područjima i proširenju srednjih razina sive boje prema krajevima skale histograma zbog toga što je kod većine slika histogram oblika Gausove funkcije.

Za transformacijsku funkciju je potrebno da bude monotono rastuća iz prostog razloga što informacija koju nosi slika ne zavisi od pojedinih vrijednosti piksela već od njihovih relativnih odnosa sa ostalim pikselima, time očuvanjem ovog uslova sprečavaju neželjene distorzije. [9] Iako teorijski ekvalizacijom histogram postaje ravan, odnosno sve vrijednosti intenziteta postaju jednako vjerovatne, navedeno nije moguće ostvariti iz razloga što je u digitalnoj obradi slike skup vrijednosti koje pikseli mogu poprimiti konačan. Za digitalnu sliku, transformacijska

(a) Slika prije predprocesiranja



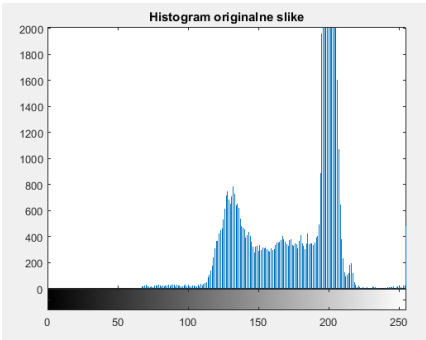
(b) Monohromatska slika

Slika 3.1: Originalna slika

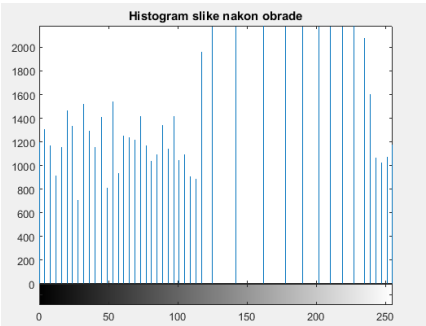
funkcija je data relacijom 3.1.

$$s_k = T(r_k) = (L-1)\sum_{i=0}^{k} p(r_i) = \frac{L-1}{M \cdot N}\sum_{i=0}^{k} n_i \tag{3.1}$$

Gdje je M·N ukupan broj piksela na slici, n_i broj piksela sa intenzitom i, dok ostali parametri imaju isto značenje kao i u prije navedenim transformacijama. Za ovu transformaciju također važi da preslikava ulazne intenzitete $r \in [0, L-1]$ u izlazne intenzitete $s \in [0, L-1]$. Transformacijska funkcija je zapravo kumulativna funkcija razdiobe vjerovatnoće za slučajnu varijablu r. [9]



(a) Originalna slika



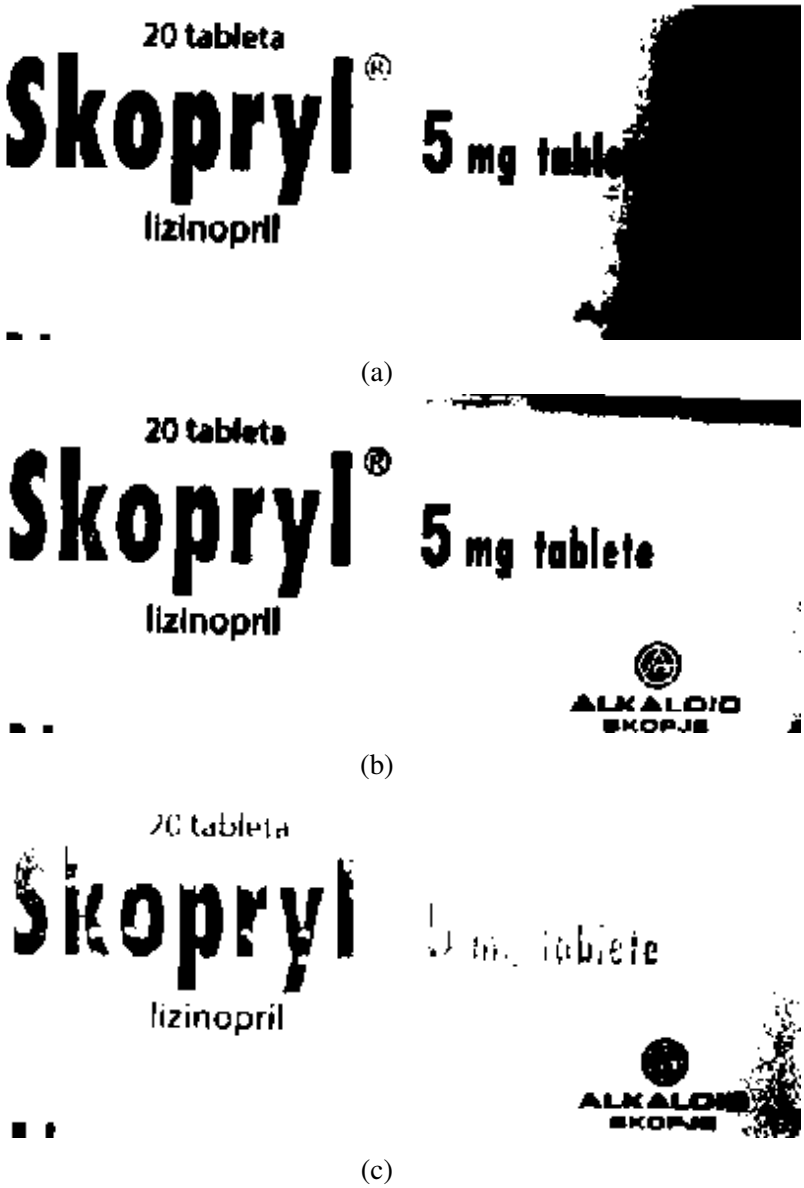
(b) Slika nakon ekvalizacije

Slika 3.2: Prikaz efekta ekvalizacije histograma

3.2 Binarizacija

Binarizacija slike (eng. *thresholding*) je najjednostavniji način segmentacije na osnovu intenziteta slike. Ovaj proces se obično primjenjuje na monohromatske slike, s toga je potrebno izvršiti konverziju slike iz RGB modela u BW model. Binarizaciju možemo kategorisati u dvije glavne kategorije: globalna i lokalna. Globalna metoda za određivanje praga binarizacije bira jednu vrijednost za cijelu sliku dokumenta. Ovaj proces se često zasniva na estimaciji nivoa pozadine iz histograma slike. S druge strane, lokalni adaptivni prag koristi različite vrijednosti praga za svaki piksel u odnosu na okolno područje tog piksela. [10]

Na slici 3.3 su prikazana 3 različita tipa binarizacije: Otsu, adaptivna i binarna.



(a)

(b)

(c)

Slika 3.3: Tipovi binarizacije: a) Otsu, b) adaptivni, c) binarni

Najbolje performanse u oblasti optičkog prepoznavanja karaktera pokazano imaju dvije metode: Otsu i adaptivni način segmentacije. Otsu metoda pretvara monohromatsku sliku u binarnu. Algoritam pretpostavlja da slika sadrži dvije klase piksela na osnovu bimodalnog histograma. Otsu metoda traži prag koji minimizira varijancu unutrašnje klase, definisane u

jednačini 3.2 kao težinska suma dviju klasa. Ovaj način binarizacije je pogodan za skenirane dokumente.

$$\sigma_{\omega}^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \tag{3.2}$$

Težinski koeficijenti ω_i su vjerovatnoće da su dvije klase odvojene pragom t i σ_i su varijanse ovih klasa. Prednost Otsu algoritma je što minimizacija unutrašnje klase ujedno znači i maksimizacija vanjske klase, predstavljeno jednačinom 3.3:

$$\sigma_b^2(t) = \sigma^2 - \sigma_{\omega}^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2 \tag{3.3}$$

gdje je μ_i srednja vrijednost klase.

Prilikom akvizije slike upotrebom kamere osvjetljenje nije uniformno po cijeloj slici što uzrokuje da histogram nije bimodalan, stoga za potrebe binarizacije slike Otsu metoda ne daje dovoljno dobre rezultate. Značajno poboljšanje se dobija korištenjem lokalne adaptivne binarizacije. Metoda lokalne binarizacije računa pojedinačne pragove za svaki piksel pomoću susjednih piksela.[10] Ovim se metodama obično mogu postići dobri rezultati čak i na veoma degradiranim dokumentima, ali su često spori upravo iz razloga što se mora za svaki piksel treba izvršiti izračunavanje praga.

Intenzitet piksela na poziciji (x,y) monohromatske slike je $g(x,y) \in [0,255]$. Kod ovog tipa cilje je izračunati vrijednost praga $t(x,y)$ za svaki piksel tako da je:

$$o(x) = \begin{cases} 0, & \text{if } g(x,y) < t(x,y). \\ 255, & \text{inače.} \end{cases} \tag{3.4}$$

Iz jednačine 3.5 određivanje vrijednosti praga se računa pomoću srednje vrijednosti m(x,y) i standardne devijacije s(x,y) intenziteta piksela koji se nalaze unutar prozora dimenzija wxw centriranog oko piksela na lokaciji (x,y).

$$t(x,y) = m(x,y)\left[1 + k\left(\frac{s(x,y)}{R} - 1)\right)\right] \tag{3.5}$$

gdje je R maksimalna vrijednost standardne devijacije, za monohromatske slike $R = 128$, k je parametar koji može uzimati vrijednost u opsegu $[0.2, 0.5]$. Lokalna srednja vrijednost $m(x,y)$ i standardna devijacija prilagođavaju vrijednost praga prema kontrastu lokalnih susjednih piksela.

Kada je u nekom području slike visok kontrast, $s(x,y) \approx R$ što rezultira $t(x,y) \approx m(x,y)$. Međutim, kada je kontrast u lokalnom susjedstvu prilično nizak prag $t(x,y)$ je manji od srednje vrijednosti i time uspješno uklanja relativno tamna područja pozadine. Parametar k kontroliše vrijednost praga u lokalnom prozoru tako da što je veća vrijednost k, to je niži prag od srednje vrijednosti $m(x,y)$. Najčešća vrijednost parametra k je 0.34. [10]

3.3 Uklanjanje šuma

Napredak u tehnologiji proizveo je uređaje za akviziciju slika sa značajnim poboljšanjem. Moderna tehnologija je omogućila smanjenje nivoa šuma na gotovo zanemarive nivoe, međutim još uvijek postoje izvori šuma koji se ne mogu eliminisati. Kod OCR sistema uklanjanje šuma za posljedicu ima poboljšanje pouzdanosti i robusnosti faza segmentacije i prepoznavanja karaktera. U binarnim slikama uobičajna manifestacija šuma je u obliku izolovanih piksela, tzv. „*salt and pepper*“ šuma.

Većina metoda obnove slika bazira se na konvoluciji koja se primjenjuje na cijelu sliku. Uzroci degradacije mogu biti: defekti optičkih leća, nelinearnost senzora, kretanje objekta, pogrešan fokus, itd. Cilj restauracije slike je rekonstrukcija originalne slike iz njene degradirane verzije. Kako bismo omogućili preciznije prepoznavanje karaktera iz skeniranog dokumenta, potrebno je smanjiti djelovanje šuma. Postojeće tehnike za uklanjanje šuma možemo posmatrati kroz tri kategorije: filtriranje, morfološke operacije i modeliranje šuma.[11]

Filtriranje: ima za cilj da ukloni šum nastao lošom brzinom uzorkovanja u fazi prikupljanja podataka. U tu svrhu se može dizajnizati niz filtera prostornog ili frekvencijskog domena. Osnovna ideja je određivanje konvolucijske maske koja će dodijeljivati vrijednost pikselu na osnovu vrijednosti susjednih piksela. Filtri se mogu dizajnirati za zaglađivanje, izoštravanje, uklanjanje teksture iz pozadine i podešavanje kontrasta.

Morfološke operacije: osnovna ideja iza morfoloških operacija je zamjena operacije konvolucije logičkim operacijama tokom filtriranja dokumenta. One mogu biti dizajnirane tako da povezuju slomljene i razgrađuju spojene linije, odnosno koriste se za uklanjanje šuma nastalog zbog loše kvalitete tinte i dokumenta.

Modeliranje šuma: djelovanje šuma se smanjuje i pomoću nekoliko metoda kalibracije. Postoji vrlo malo rada na modeliranju šuma uvedenog optički izboličenjem, s toga nema široku primjenu u OCR-u.

3.4 Korekcija nagiba

Kada se tekstualni dokument unosi u skener bilo mehanički ili ručno, nekoliko stepeni nagiba ili iskrivljenja je neizbježno. Usljed vjerovatnoće rotacije ulazne slike i osjetljivosti velikog broja metoda analize slika na rotaciju, treba se obaviti korekcija dokumenta. Ovaj efekat se vizuelno javlja kao nagib linija teksta u odnosu na x-osu. Iskrivljenje može biti i namjerno dizajnirano da naglasi važne detalje u dokumentu, ali s obzirom da je u većini situacija slučajno, treba biti eliminisano jer značajno smanjuje tačnost procesa segmentacije i optičkog prepoznavanja znakova. Stoga, jedan od primarnih zadataka sistema za analizu slike otkrivanje nakošenja i ispravljanje istog. Tehnike detekcije isrkivljenja možemo grubo kategorisati u sljedeće grupe: analiza profila projekcije, grupisanje na osnovu najbližeg susjeda, Hough-ova transformacija, kros-korelacija i morfološke transformacije. [11]

Metode zasnovane na analizi profila projekcije su najjednostavne. Stranica dokumenta se projektuje pod nekoliko uglova i određuju se odstupanje broja crnih piksela po projektovanoj liniji. Nakon što je otkriven ugao ukošenja, tekstualni dokument se mora rotirati kako bi se ispravila ova iskrivljenost. Proces rotacije mora biti brz i precizan. Poznavajući ugao zakrivljenja, nove koordinate se određuju koristeći jednačinu 3.6.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta \\ -\sin\theta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.6}$$

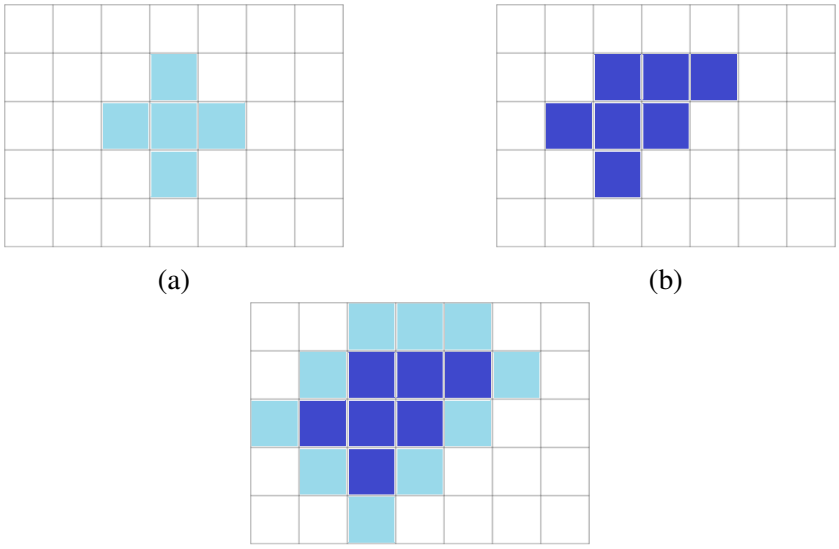
3.5 Dilatacija

Dilatacija i erozija su morfološke operacije koje povećavaju ili smanjuju veličine objekta. Dilatacija čini objekat većim dodavanjem piksela oko njegovih ivica dok erozija uklanja rubne

piksele.

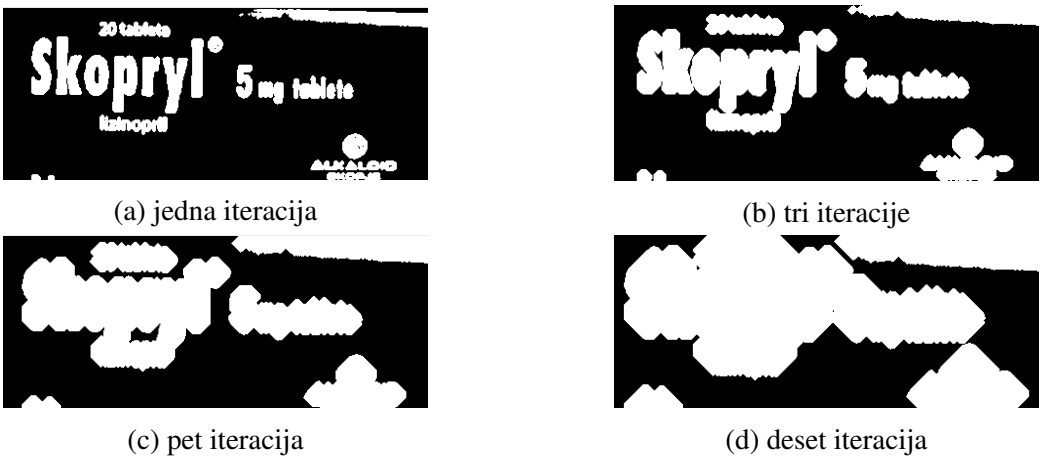
Dilatacija je proširivanje slike A sa strukturnim elementom B. Od strukturnog elementa ovisi rezultat dilatacije. Modifikovana slika posmatra se kao kombinacija dva skupa A i B pomoću vektorskog dodavanja, odnosno translacijom elementa B preko svakog piksela slike A. [12] Dilatacija skupa A strukturnim elementom B će biti predstavljena kao A$\bigoplus$B kao:

$$A \bigoplus B = \{c \in \mathbb{E}^N \,|\, c = a + b \quad \text{za} \quad a \in A \quad \text{i} \quad b \in B\} \tag{3.7}$$



Slika 3.4: Matrični prikaz dilatacije strukturnog elementa 3.4a nad baznom matricom 3.4b. Matrica 3.4c je rezultirajuća matrica

Vizuelnom analizom slike možemo primijetiti da su slova uvijek grupisana zajedno u blokove koji se formiraju od tekstualnih paragrafa. Ove paragrafe je neophodno učiniti očiglednijim što se postiže dilatacijom slike. Na slici 3.5 je prikazan proces dilatacije za posmatrani članak gdje se vidi način formiranja blokova povećanjem stepena iteracije.



(a) jedna iteracija

(b) tri iteracije

(c) pet iteracija

(d) deset iteracija

Slika 3.5: Rezultati dilatacije

* * *

Najvažniji korak za preciznost prilikom očitavanja karaktera predstavlja predprocesiranje slike. Ukoliko je u cilju poboljšati kontrast slike najjednostavnije je koristiti ekvalizaciju histograma. Međutim, slike koje imaju dobar kontrast ovaj korak se neće izvršavati. U zavisnosti od prirode slike obavljat će se različite faze predprocesiranja.

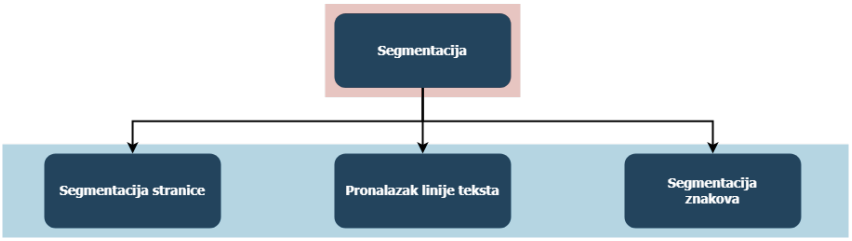
Poglavlje 4

Segmentacija

Slika dobivena nakon faze predprocesiranja sadrži dovoljnu količinu informacija o obliku i nizak nivo šuma. Po završetku predprocesiranja potrebno je pronaći elemente slike koji se dalje obrađuju. Sljedeća komponenta prepoznavanja karaktera je segmentacija. Segmentacija predstavlja proces lociranja regije štampanog ili pisanog teksta. Kada se segmentacija primijeni na sliku, pikseli koji imaju iste karakteristike, najčešće vizualne, svrstavaju se u jednu grupu. Kod OCR-a segmentacija se može odnositi na izoliranje paragrafa, riječi ili pojedinačnog znaka. Važnost segmentacije se ogleda u činjenici da stepen do kojeg može doći u odvajanju različitih linija u znakovima direktno utječe na stopu prepoznavanja.

Segmentacija slike je jedan od najvažnijih koraka koji vode do analize obrađenih podataka čiji je glavni cilj podijeliti sliku na dijelove koji imaju jaku korelaciju s objektima koji se nalaze na slici. Možemo težiti potpunoj segmentaciji, što rezultira skupom nepovezanih regija koje odgovaraju objektima u ulaznoj slici ili djelomičnoj segmentaciji, u kojoj regije ne odgovaraju izravno objektima slike. [6]

Najčešći problem koji se javlja u ovoj fazi OCR-a je konfuzija koja se javlja između teksta i grafike usljed spojenih ili razdvojenih znakova. Obično, ovakvi problemi su uzrokovani skeniranjem ili kada su znakovi povezani sa slikama koje se nalaze u dokumentu ili slici. Ukoliko je dokument tamna fotokopija ili ako je nizak prag skeniranja pojavit će se spojeni znakovi. Suprotno, razdvojeni znakovi će se pojaviti ukoliko je dokument svijetla fotokopija ili ako je visok prag skeniranja. Segmentaciju ćemo posmatrati kroz tri glavna dijela prikazana na slici 4.1.



Slika 4.1: Tri osnovna tipa segmentacije

4.1 Segmentacija stranice

Da bi se identifikovali regioni teksta u dokumentu, potrebno je izvršiti analizu segmentacije stranice, gdje se odvajaju smislene komponente i odgovarajući atributi kao što su tekstualni

blokovi, slike, tabele i slično. Ove informacije se zatim unose u sistem za pronalaženje i prepoznavanje linija teksta.

Postojeći algoritmi za analizu dokumenta mogu se podijeliti u tri glavne kategorije: pristupi odzdo prema gore (eng. *bottom-up*), odozgo prema dolje (eng. *top-down*) i hibridne tehnike. Top-down metoda rekurzivno izdvaja velike regije u manje podregije, dok tehnika bottom-up grupiše piksele od interesa i spaja ih u veće blokove ili povezane komponente. Hibridne tehnike su kombinacija top-down i bottom-up metoda. Top-down pristup može obraditi samo jednostavne dokumente karakterističnog formata, odnosno ne uspijeva obraditi dokumente koji sadrže komplikovane geometrijske strukture. Najčešće za obradu se koriste binarne i sive slike sa uniformnim pozadinama. [13]

4.2 Pronalazak linije teksta

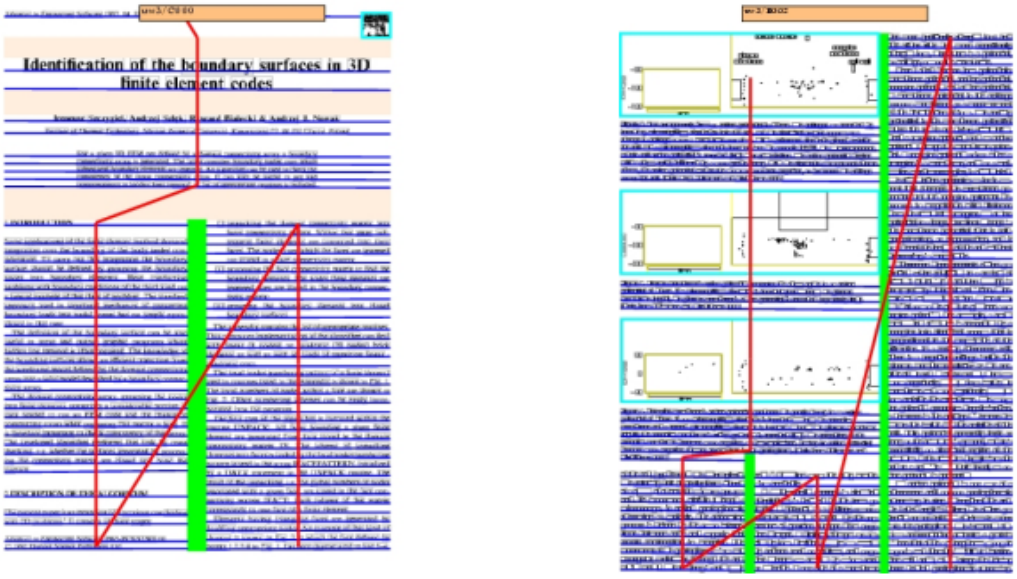
Podrazumijeva proces dodjeljivanja iste oznake prostorno usklađenim jedinicama, tj. pikselima, povezanim komponentama ili karakterističnim tačkama. Nadalje, na temelju dobivenih oznaka, tekst je podijeljen na različita područja od kojih svaki predstavlja redak teksta. Nakon što je završena segmentacija tekstualnih linija, ona pruža bitne informacije za uzastopne korake slike dokumenata kao što su otkrivanje i ispravljanje iskrivljenja, izdvajanje značajki teksta i prepoznavanje znakova. Stoga je to preduvjet za daljnji proces analize slike dokumenta. Iako su neke tehnike otkrivanja tekstualnih linija uspješne u printanim dokumentima, obrada rukopisnih dokumenata ostala je ključni problem u optičkom prepoznavanju znakova (OCR). Većina metoda segmentacije tekstualnih linija temelji se na pretpostavkama da je udaljenost između susjednih linija teksta značajna, kao i da su linije teksta razumno ravne. Međutim, te pretpostavke nisu karakterizirane za rukom pisane dokumente. Stoga je segmentacija teksta vodeći izazov u analizi slike dokumenta.

Metode koje se primenjuju za pronalaženje linija obično pretpostavljaju da je tekst horizontalno orijentisan. Zadatku segmentacije se pristupilo definiranjem pravogaonika ili linije za pretraživanje (slika 4.2) unutar kojih (u slučaju pravougaonika) ili uz koje su (u slučaju linija pretraživanja) otkriveni pojedinačni znakovi ili konture objekta. [14] To je tipičan način korištenja regija interesa (eng. *region of interests*) u obradi slika: kao ograničenje područja unutar kojeg se objekti slika trebaju segmentirati.

4.3 Segmentacija znakova

Izbor algoritma segmentacije koji se koristi ključni je faktor u odlučivanju o tačnosti OCR sistema. Ako postoji dobra segmentacija znakova, tačnost prepoznavanja će biti visoka. Segmentacija riječi u karaktere postaje veoma teška usljed loše akvizicije slike. Segmentacija karaktera je dugo bila kritična oblast procesa OCR-a. Dobar dio nedavnog napretka u čitanju neograničenog štampanog i pisanog teksta može se pripisati boljem proučavanju segmentacije.

Segmentacija znakova je operacija koja nastoji razložiti sliku niza znakova u podslike pojedinih simbola. To je jedan od procesa odlučivanja u sistemu za optičko prepoznavanje znakova. Odluka o segmentaciji nije lokalna odluka, već je ovisna o prethodnim i naknadnim odlukama. [15] Stvaranje dobrog podudaranja sa simbolom iz rječnika je neophodno za pouzdano prepoznavanje. Čak i niz zadovoljavajućih uzoraka može se smatrati netačnim ako kontekstualni zahtjevi na izlazu sistema nisu zadovoljeni. Na primjer *"cl"* često može vrlo sličiti *"d"*, ali obično takav izbor neće predstavljati kontekstualno važeći rezultat. [13]



Slika 4.2: Primjer redoslijeda čitanja teksta. Plave linije označavaju segmente teksta, a crvena linija povezuje središte uzastopnih segmenata teksta u redoslijedu čitanja. [4]

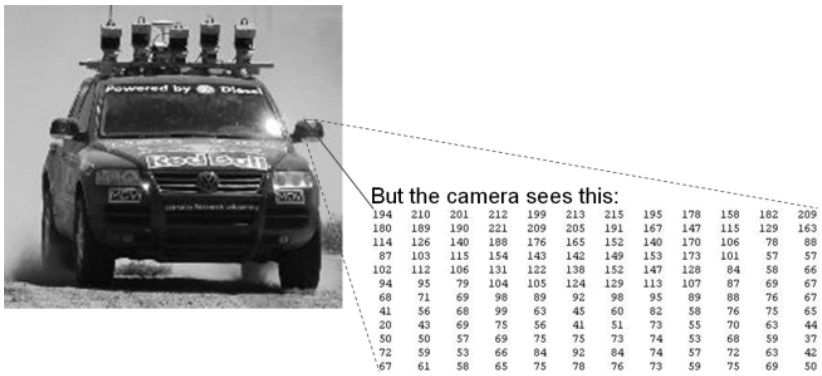
* * *

Neophodno je izolirati entitete koji čine tu sliku, prepoznati ih i odrediti odnos koji postoji između njih. Bez dobrog razgraničenja tih entiteta, nemoguće je napraviti bilo kakav smisao scene koja je uhvaćena u datoj slici. Ali problem s kojim se suočava svaka tehnika segmentacije je da se kvaliteta segmentacije koja se može procijeniti samo na temelju interpretacije slike.

Poglavlje 5

Implementacija u OpenCV-u pomoću Tesseract-a

OpenCV je besplatna (eng. *open source*) biblioteka koja nudi osnove za eksperimente i aplikacije računarskog vida (eng. *computer vision*). Također, sadrži interfejse za snimanje, obradu i prezentaciju slika. OpenCV se široko koristi u akademiji i industriji. Biblioteka je pisana u C i C++ jezicima i može raditi na brojnim platformama poput Linux-a, Windows-a, Androida i slično.[5] Jedan od ciljeva OpenCV-a je da obezbijedi infrastrukturu kompjuterske vizije koja je jednostavna za upotrebu i brzo kreiranje aplikacija. Biblioteka OpenCV-a sadrži više od 500 funkcija koje obuhvaćaju mnoga područja, uključujući pregled tvorničkih proizvoda, medicinske snimke, sigurnost, korisnički interfejs, kalibraciju kamere, stereo viziju i robotiku. Cilj OpenCV-a je pružanje osnovnih alata za rješavanje problema računarske vizije.



Slika 5.1: Crno-bijela slika iz perspektive računara[5]

Identifikacija teksta na određenoj sceni i prepoznavanje njenog sadržaja postaje sve važnije. Neke aplikacije uključuju prepoznavanje natpisnih pločica, prepoznavanje saobraćajnih znakova za automobile koji sami voze, skeniranje knjiga za digitalizaciju sadržaja i tako dalje. Pronalaženje teksta koji se slučajno pojavljuje na slici predstavlja veliki problem. Postoji nekoliko faktora koji dodatno utječu na skenirani tekst, poput trodimenzionalnost, raznovrsnost, osvjetljenje i sjene i zamagljenost.[5]

Trodimenzionalnost: Tekst može biti bilo koje dimenzije i orijentacije. Također, tekst se može djelimično preklapati ili biti prekinut. U odnosu na dimenzije slike riječi se mogu nalaziti na bilo kojem dijelu, što ukazuje da ukoliko su dimenzije slike veće to proces pretraživanja duže traje.

Raznovrsnost: Pozadina na kojoj se nalazi posmatrani sadržaj može biti uniformno predstavljena jednom bojom ili složenim elementima koji dodatno otežavaju proces proces detekcije. Tekst može biti ispisan različitim fontovima i bojama. Neki fontovi nemaju jasno definisane granice kontura.

Osvjetljenje i sjene: Položaj sunčeve svjetlosti i boja mijenjaju se tijekom vremena. Različiti vremenski uvjeti kao što su magla ili kiša mogu izazvati šum. Osvjetljenje može biti problem čak i u zatvorenim prostorima, jer se svjetlo reflektira preko obojenih predmeta i udara u tekst.

Zamagljenost: Tekst se može pojaviti u regiji koja nije prioritet objektiva za automatsko fokusiranje. Zamagljenost je također uobičajeno u pokretnim kamerama ili u prisutnosti magle.

5.1 Inicijalizacija Tesseract-a

Direktna primjena OCR softvera kao što je Tesseract na tekstualne regije u boji ili sive boje zbog niske kvalitete i niske rezolucije takvih regija ne daje pouzdane rezultate. Uobičajeno rješenje u literaturi sastoji se od nekako poboljšanja i binarizacije teksta prije nego što se pošalje OCR softveru. Tesseract je implementirao algoritme ispravljanja nagiba i segmentacije teksta, kao i većina OCR biblioteka danas. To omogućava da se unaprijedi otkrivanje teksta u mnogim slučajevima. Na primjer, ako se kreira OCR aplikacija za stare dokumente, zadani prag koji koristi Tesseract može stvoriti tamnu pozadinu.

Upotrebom kombinacije *Qt Creator*, *OpenCV* i *Tesseract* se može razviti GUI (eng. *graphical user interface*) ujedno i detektovati tekst koji se pojavljuje u videu s web kamere ili analizirati fotografisane slike. To omogućava stvaranje širokog spektra aplikacija, od pristupačnosti marketingu, pa čak i polja robotike. Primjer GUI-a koji je korišten u ovom radu je na slici 5.2

Budući da je Tesseract već integriran u verzije OpenCV 3.0 i veće, još uvijek vrijedi proučiti njegov API (eng. *application programming interface*) jer omogućava kontrolu preko Tesseract parametara. [16] Inicijalizacija Tesseract-a se vrši funkijom *init* sa prototipom:

int Init(const char datapath, const char* language, OcrEngineMode oem)*

Značenje svakog od argumenata od funkcije se nalazi u prilogu A.1. [16] Važno je naglasiti da se init funkcija može izvršiti mnogo puta. Ako je omogućen drugi jezik ili način rada, Tesseract će obrisati prethodnu konfiguraciju i ponovo početi. Ako su navedeni isti parametri, Tesseract je dovoljno pametan da jednostavno ignoriše naredbu. Funkcija init vraća 0 za uspjeh i -1 za neuspjeh.

Zatim je potrebno odrediti način segmentacije strane:

void SetPageSegMode(tesseract :: PSM_SINGLE_BLOCK)

Moguće vrijednosti parametra *SetPageSegMode* su date u prilogu A.2. [16] Algoritam OpenCV 3.0 koristi analizu povezanih komponenti. Ovaj pristup se bazira na grupisanju piksela u regije u kojima pikseli imaju slična svojstva. Te regije imaju veće šanse da budu identificirane kao karakteri. Prednost ovog pristupa je što ne ovisi o osobinama teksta (orijentacija, razmjera i fontovi), a omogućavaju segmentaciju regije što se može koristiti za dijeljenje teksta pogodan za OCR. Detaljan pregled strukture rada Tesseract-a je objašnjem u poglavlju 5.2.

Slika 5.2: Izgled aplikacije

Treniranje Tesseract-a

Budući da je klasifikator u stanju lako prepoznati oštećene znakove, on nije istreniran na rad sa takvim karakterima. Klasifikator Tesseract-a je obučen na samo 20 uzoraka svakog od 94 znaka iz 8 fontova u jednoj veličini, ali sa 4 atributa (normalni, podebljani, kurziv, podebljani kurziv), što znači da ukupno ima 60160 istreniranih uzoraka. Ovo je značajna suprotnost drugim objavljenim klasifikatorima, poput Calera klasifikatora sa više od milion uzoraka, i Bairdovog klasifikatora sa 1175000 uzoraka.[17]

5.2 Arhitektura Tesseract-a

Analiza izgleda stranice za Tesseract dizajnirana je da ne ovisi o jeziku, ali ostatak softvera razvijen je za engleski jezik. Ostali komercijalni softveri su isključivo namijenjeni za prepoznavanja crnog teksta na bijeloj pozadini, stoga jedan od dizajnerskih ciljeva Tesseract-a bio je da prepoznavanje bijelog teksta na crnoj pozadini bude jednako dobro kao u suprotnom slučaju. Ovo je vodilo dizajn u smjeru analize povezanih komponenti (eng. *connected components - CC*) i radeći na obrisima komponenti.

Prvi korak nakon analize povezanih komponenti je pronalazak blobova u tekstu. Blob predstavlja klasifirajuću jedinicu koja može biti jedna ili više povezanih komponenti koje se preklapaju u vidu ugniježdenih obrisa ili rupa.[18] Nakon odlučivanja koji obrisi čine blobove, blok za pronalazak linija teksta detektuje horizontalne linije teksta zahvaljujući vertikalnom preklapanju susjednih znakova u liniji teksta. Algoritam za detekciju osnovne linije teksta je najbolje razvijen za engleski jezik, što omogućava pronalazak teksta i u slučajevima kada je veći nagib teksta.

Nakon pronalaženja linija teksta, provjerava se uniformnost razmaka među znakovima i po-

kreće se jedan od dva različita algoritma segmentacije riječi. Većina procesa prepoznavanja djeluje na svaku riječ neovisno, nakon čega slijedi završna faza razlučivanja nejasnog prostora (eng. *fuzzy space*). Linije teksta su podijeljene u riječi u ovisnosti prema vrsti razmaka između znakova. U većini slučajeva, jedan blob odgovara jednom karakteru. Prvi zadatak bloka za prepoznavanje riječi je klasifikacija svakog od blobova, zatim se vrši pretraživanje riječi u rječniku u skladu sa kombinacijom dobijenih blobova. Ako rezultat nije dovoljno dobar, sljedeći korak je sjeckanje loše prepoznatih znakova, čime se poboljšava pouzdanost klasifikatora. [17] U idućem koraku, rezultirajuća segmentacija sa najboljim pretraživanjem u rječniku spaja sjeckane fragmente znakova ili dijelove znakova koji su u originalnoj slici razbijeni na više povezanih komponenti. Nakon svakog pretraživanja klasificiraju se sve nove kombinacije bloba, a rezultati klasifikatora ponovo se provjeravaju u rječniku.

Riječi na slici obrađuju se dva puta. Na prvom prolazu uspješne riječi, odnosno koje se nalaze u rječniku, prenose se u adaptivni klasifikator za treniranje. Kada adaptivni klasifikator ima dovoljno uzoraka može dati rezultate klasifikacije, nekada čak i nakon prvog prolaza. Adaptivni klasifikator dobiva priliku za tačnije prepoznavanje teksta u nastavku posmatranog teksta. Budući da adaptivni klasifikator može naučiti nešto korisno pri kraju teksta, radi se ponovni prolaz u cilju tačnog prepoznavanja.

5.3 Potrebe za adaptivnom binarizacijom

Slika u sivim tonovima najprije se pretvara u binarnu pomoću prilagodljivog praga. Adaptivni prag je nužan za pretvaranje slike u sivim tonovima u binarnu sliku jer je teško pretvoriti neke slike u binarne primjenom konstantnog globalnog praga. Adaptivni prag pomoći u sljedećim situacijama:

- Originalne slike su ispisane sa vrlo tankim linijama. Tanke linije u kombinaciji s ograničenom rezolucijom skenera nestaju nakon primjene globalnog praga.
- Loše reproducirane kopije. Osvjetljenje teksta može imati isti učinak kao i istanjenje linije.
- Tekst ispisan na pozadini u boji. Takav se tekst može lako izgubiti globalnim pragom.

U prethodno navedenim slučajevima primjena konstantnog praga ne bi bilo korisno.[4] Uobičajno korišenje praga čija je vrijednost polovina skale sive za pretvaranje monohromatske slike u binarnu ne daje uvijek dobre rezultate.

Postoji modifikovana tehnika binarizacije u kojoj je razina praga sama po sebi promjenjiva. U OpenCV-u je ova metoda implementirana u funkciji cv::adaptiveThreshold() prototipa:

void cv::adaptiveThreshold(src, dst, maxValue, adaptiveMethod, thresholdType, blockSize, C);

Detaljna značenja parametara ove funkcije se nalaze u prilogu A.3. Ova funkcija omogućava dvije različite vrste adaptivnih pragova ovisno o postavkama *adaptiveMethod*. U oba slučaja, adaptivni prag T(x,y) postavlja se računanjem težinskog prosjeka regije. oko svake lokacije piksela minus konstanta *C*. Dimenzije regije su određene parametrom *blockSize*. Ako je metoda postavljena na $cv :: ADAPTIVE\_THRESH\_MEAN\_C$, tada je težinski prosjek svih piksela jednak. Ako je postavljeno na $cv :: ADAPTIVE\_THRESH\_GAUSSIAN\_C$, težinski prosjek piksela u regiji oko (x,y) se određuje prema Gaussovoj funkciji njihove udaljenosti od te središnje tačke.

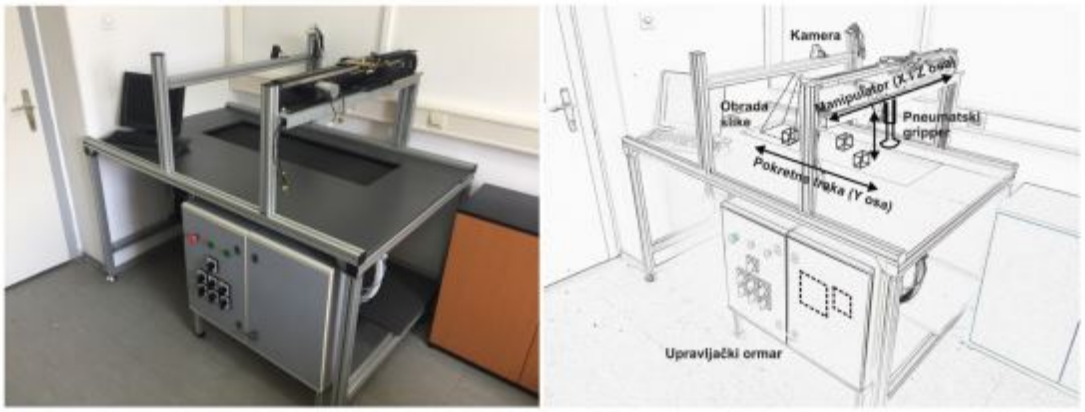
Slika 5.3: Arhitektura Tesseract-a

Testiranjem je pokazano da je ovo najbolji način binarizacije slike ukoliko se za prepoznavanje karaktera koristi softver Tesseract. Za binarizaciju u nastavku rada korištena je adaptivna binarizacija kod koje je težinski prosjek piksela jednak za sve susjedne piksele.

5.4 Testiranje Tesseract-a

Testiranje brzine izvršavanja, preciznosti, veličine slova, varijabilnosti fontova i osvjetljenja se vršilo na računaru specifikacija:

Procesor: AMD E2-9000e RADEON R2
Instalirani RAM: 4 GB + 4GB
Tip sistema: 64 - bit
Operativni sistem: Ubuntu 18.04 (VM)



Slika 5.4: ETFCam

Dio eksperimentalnih rezultata koji su zahtijevali povezivanje sa kamerom je rađeno na modelu *ETFCam v1.0*, prikazan na slici 5.4, na kojem se nalazi kamera *Sony XC-55*. Testiranja su vršena na kutijama tableta koje su postavljene na pokretnu traku. Međutim, usljed ograničenja brzine frejmova i brzine pokretne trake svi objekti su ručno postavljeni bez uključivanja pokretne trake.

Brzina izvršavanja

Većina tehnika koje koristi Tesseract prilično su standardne u području OCR (izgled stranice, pronalazak linije teksta, izdvajanje karaktera i nekoliko faza klasifikacije), ali ima velike zahtjeve za memorijom. To je razlog zbog kojeg nova verzija Tesseract 4.0 na većini uređaja se izvršava sporije u odnosu na verziju Tesseract 3.0. Brzina izvršavanja Tesseract-a prvenstveno ovisi o specifikacijama uređaja na kojem se izvršava aplikacija i količine teksta koji se obrađuje. Brzina testiranja slike 5.5a na drugom računaru je skoro 6 puta bolja u odnosu na rezultate dobijene u tabeli 5.1.

S obzirom da je pri svakom pokretanju aplikacije brzina prepoznavanja karaktera varirala, mjerenje je vršeno na testnim slikama 5.5 određivanjem srednje vrijednosti kroz 10 pokretanja Tesseract-a. Iz tabele 5.1 se vidi značajan utjecaj količine teksta na vrijeme izvršavanja. Također, slike malih dimenzija usporavaju aplikaciju za red par stotina milisekundi. Stoga ukoliko je brzina izvršavanja jedan od najvažnijih parametara promjena veličine slike može dovesti do poboljšanja.

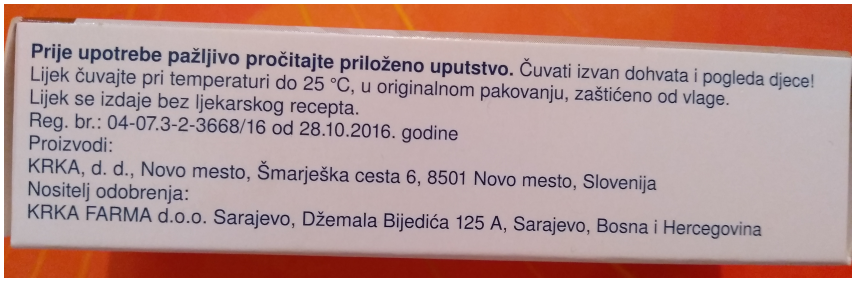
Tabela 5.1: Brzina izvršavanja Tessseract-a

Broj riječi:	1-10 (mala količina teksta)	50+ (srednja količina teksta)	100+ (velika količina teksta)
Brzina *[s]*	0.41493	10.6857	26.298

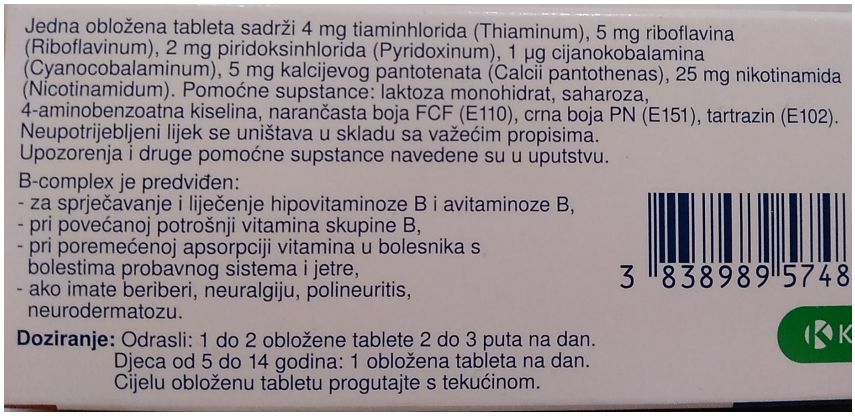
Veličina slova

Ekstremno umanjena ili povećana slova onemogućavaju prepoznavanje bilo kakvih karaktera. Za velike dimenzije dolazi do pojave znakova koji se ne nalaze na slici, najčešće su to sljedeći znakovi "-,".’+*". U slučaju slika jako malih dimenzija ne dolazi do detekcije bilo kakvih

(a) Mala količina teksta



(b) Srednja količina teksta



(c) Velika količina teksta

Slika 5.5: Testne slike za brzinu izvršavanja

karaktera i izlaz Tesseract-a ne daje nikakve rezultate. Pored samih dimenzija na mogućnost prepoznavanja teksta utiče i rezolucija slike. Na osnovu promjene dimenzija i rezolucije slike 5.6 dobijene su granične vrijednosti za očitavanje sadržaja slike. S obzirom da je riječ o maksimalnim i minimalnim vrijednostima tačnost teksta na izlazu Tesseract-a je znatno lošija u poređenju sa originalnom slikom dimenzija $252x103$, ostale dimenzije su dobijene skaliranjem originalne slike. Dobijeni rezultati su prikazani u tabeli 5.2.

Iz tabele 5.2 se vidi da povećanje rezolucije ne doprinosi samo povećanju preciznosti već i mogućnosti detekcije teksta. Za male dimenzije je očekivano da će prednost imati i veća rezolucija, međutim postoje i slučajevi kada ni povećanje rezolucije ne daje optimalne rezultate. Kako su osnovni zahtjevi za uspješnu analizu slike u polju prepoznavanja karaktera preciznost i brzina, neophodno je izbjegavati granične vrijednosti veličine teksta. Ukoliko se ulazna slika očekuje u takvim dimenzijama Tesseract se može dodatno istrenirati tako da može detektovati tekst i na takvim slikama pomoći *traineddata* datoteke za određeni jezik.

Dodatni problem koji proizilazi je što postoji mogućnost da veličina slova nije konstantna

Slika 5.6: Testna slika za maksimalnu i minimalnu veličinu slova

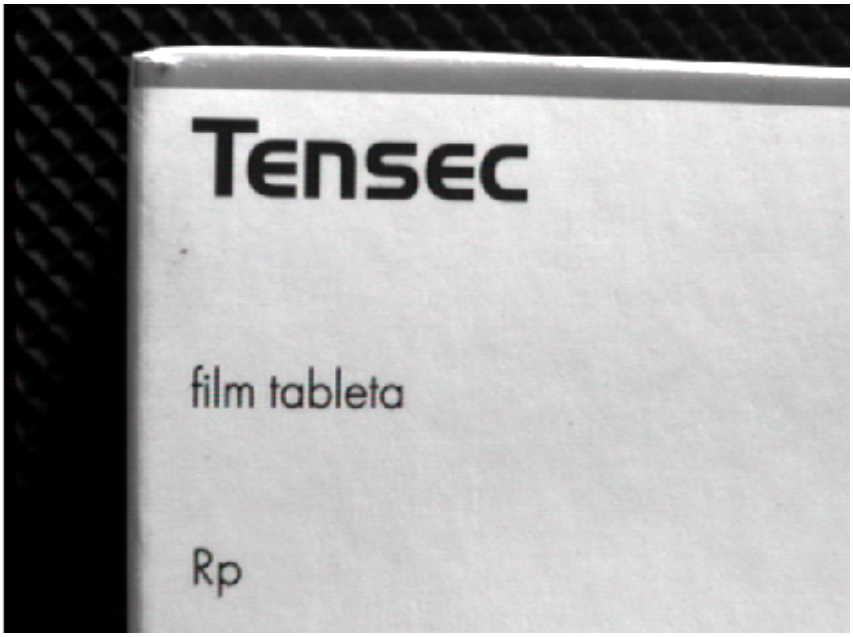
Tabela 5.2: Granične vrijednosti dimenzija slike u ovisnosti o rezoluciji

Rezolucija	Maksimalna širina	Maksimalna dužina	Minimalna širina	Minimalna dužina
300 *[dpi]*	4.892 *[px]*	2.000 *[px]*	27 *[px]*	11 *[px]*
400 *[dpi]*	5.626 *[px]*	2.300 *[px]*	27 *[px]*	11 *[px]*
72 *[dpi]*	4.000 *[px]*	1.635 *[px]*	25 *[px]*	10 *[px]*

na svim dijelovima slike, primjer za takav slučaj je prikazan na slici 5.7. Izlazni tekst dobijen kao rezultat Tesseract-a je:

TENSEC
tilrn tableto /
Rp:

Rezultat nije zadovoljavajući, jer je uspješno pročitana samo prva riječ, dok tačnost ostalih je značajno narušena.



Slika 5.7: Slika sa promjenljivom veličinom slova

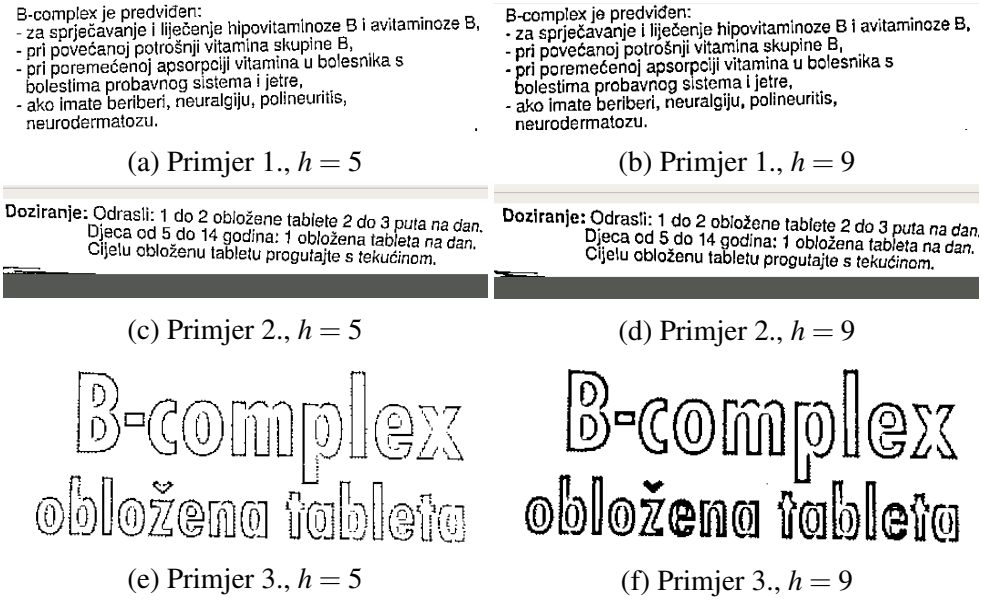
Količina grešaka

Nijedan od softverskih programa za prepoznavanje karaktera ne garantuje 100% tačnost. Za ispravno prepoznavanje karaktera, Tesseract zahtijeva ulaznu sliku visoke kvalitete i rezolucije reda 300-400 dpi. Dodatni šumovi koji se javljaju kao posljedica lose kvalitete slike će biti prepoznati najčešće kao crtice, tačke, zarezi ili apostrofi. Filtriranje predstavlja jedan od načina rješenja ovog problema.

Na slikama 5.8 prikazano je kako različit broj piksela koji se koristi za adaptivnu binarizaciju (označen sa parametrom h) može utjecati na izlazni tekst. Stoga, svaka slika iziskuje posebnu vrijednost za adaptivnu binarizaciju. Količinu grešaka možemo odrediti izrazom 5.1, gdje Err predstavlja procentualnu količinu grešaka, N_{errror} broj pogrešnih karaktera i N_{total} ukupan broj karaktera.

$$Err = \frac{N_{error}}{N_{total}} \times 100 \tag{5.1}$$

Primjenjujući izraz 5.1 nad slikama 5.8 sa dimenzijama bloka za binarizaciju predstavljene parametrom h, dobijeni su rezultati prikazani tabelom 5.4. Ukoliko je prosječan broj grešaka do 2% prepoznavanje karatera se smatra idealnim. U slučaju da je prosječan broj pogrešnih karaktera preko 10% rezultat se odbacuje i potrebno je poboljšati predprocesiranje. Iz tabele 5.4 se vidi važnost ispravne binarizacije. Posmatrajući rezultate Tesseract-a kod slika 5.8e i 5.8f prosječan broj pogrešno pročitanih karaktera se razlikuje za 20%, iako je razlika u dimenzijama bloka susjednih piksela koji se posmatraju za određivanje praga binarizacije 4. Također i kod ostalih primjera vidimo znatno bolje rezultate u zavisnosti koja vrijednost dimenzija bloka odgovara toj slici.



(a) Primjer 1., $h = 5$ (b) Primjer 1., $h = 9$

(c) Primjer 2., $h = 5$ (d) Primjer 2., $h = 9$

(e) Primjer 3., $h = 5$ (f) Primjer 3., $h = 9$

Slika 5.8: Testne slike za određivanje količine grešaka

Varijabilnost fontova

Pored svih prethodno navedenih faktora koji utiču na kvalitet rada OCR softvera, veliku ulogu ima i sam tip slova koji se nalazi na slici odnosno font. Neki tipovi slova mogu davati izrazito loše rezultate, u tom slučaju jedna od mogućnosti je treniranje Tesseract-a za tačno željeni font,

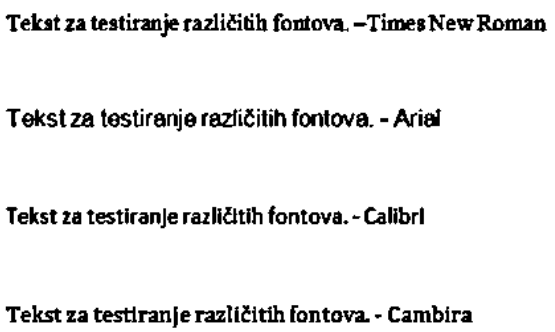
Tabela 5.3: Prosječan broj pogrešnih karaktera

	Broj pogrešnih karaktera	Ukupan broj karaktera	Prosječan broj pogrešnih karaktera
Slika 5.8a	3	279	1.08 %
Slika 5.8b	7	279	2.51 %
Slika 5.8c	0	162	0.00 %
Slika 5.8d	1	162	0.62 %
Slika 5.8e	5	25	20.00 %
Slika 5.8f	0	25	0.00 %

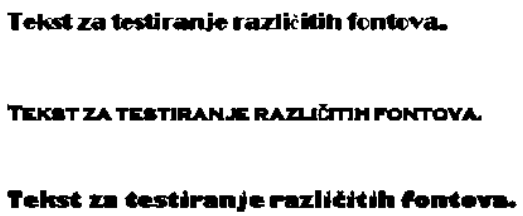
što će osigurati znatno veći procenat uspješno prepoznatih karaktera. Kombinacija više različitih fontova koji se znatno razlikuju poput *serif* i *script* smanjuje mogućnost ispravne detekcije. Tačnost prepoznavanja karaktera testirano je na četiri tipa slova serif i sans-serif (slika 5.9a), dekorativnim (slika 5.9b) i kurzivnim (slika 5.9c). Serif i sans-serif su davali najbolje rezultate, odnosno sva slova su ispravno prepoznata. Za razliku od serif i sans-serif tipa slova, dekorativna slova zahtijevaju precizniju obradu u fazi predprocesiranja uz preciznost od 99.16%, dok kurzivna slova su imale znatno veću stopu grešaka od 10%, što je minimalna dozvoljena.

Tabela 5.4: Rezultati aplikacije nad slikama 5.9

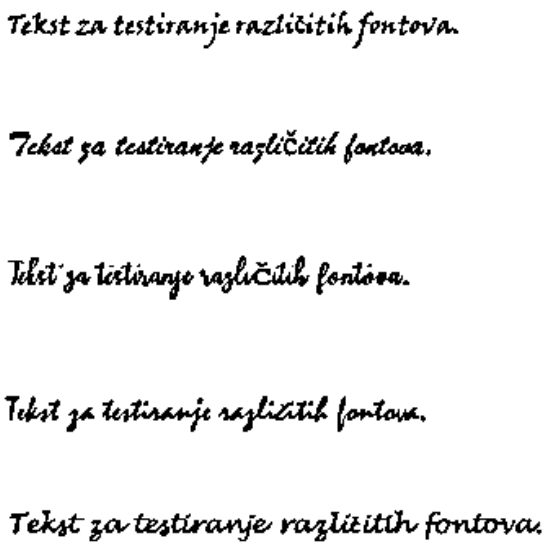
Testirani font	Rezultat	Tačnost [%]
Slika 5.9a prvi red	Tekst za testiranje različitih fontova. Times New Roman	98.18 %
Slika 5.9a drugi red	Tekstza testiranje različitih fontova. - Arial	97.87 %
Slika 5.9a treći red	Tekst za testiranje različitih fontova. - Calibri	100.00%
Slika 5.9a četvrti red	Tekst za testiranje različitih fontova. - Cambira	100.00%
Slika 5.9b prvi red	Tekst za testiranje različitih fentcva.	95.00 %
Slika 5.9b drugi red	Tekst za testiranje različitih fontova.	100.00 %
Slika 5.9b treći red	Yekst za testiranje različitih fontova.	97.50 %
Slika 5.9c prvi red	Tekst za testiranjeraztititih feuteve.	17.50 %
Slika 5.9c drugi red	"Teka ya tsttrarje sapližižik foes.	>20.00 %
Slika 5.9c treći red	Het pe tectanye waglidill fantom.	>20.00 %
Slika 5.9c četvrti red	Tekap bestinanjeregh EEL footocn.	>20.00 %
Slika 5.9c peti red	Tekst za testiranjerazličitih fontova.	97.50 %



(a) Serif i Sans-Serif tip slova



(b) Dekorativni tip slova



(c) Kurzivni tip slova

Slika 5.9: Eksperimentalni rezultati za testiranje različitih fontova

Osvjetljenje

Vanjski faktor koji utječe na kvalitet slike, samim time i na preciznost sistema, je osvjetljenje posmatranog objekta. Sposobnost hardvera da nadoknadi promjene prosječne vrijednosti piksela kada se promijeni osvjetljenje scene varira uzimajući u obzir različite modele fotoaparata, objektiva i osvjetljenja. Ako hardver ne može nadokanditi te promjene, prosječna razina monohromatske slike premjestit će se na jedan od krajeva histograma, 0 ili 255, ovisno o nedostatku ili višku osvjetljenja. U oba slučaja slika sadržati više šuma, a kontrast će se smanjiti. Oba stanja rezultiraju gubitkom detalja na slici.

Defekti na slikama nakon faze predprocesiranja su manje vidljivi kod tamnijih slika što se ogleda i u rezultatima dobijenim na slikama 5.10. Razlog za to leži u činjenici da se adaptivna binarizacija izvršavala nad fiksnim blokom veličine 29 piksela, što je davalo idealne rezultate

pri normalnom osvjetljenju. Ukoliko je riječ o jačem osvjetljenju, veličinu tog bloka je potrebno povećati. Histogram potpuno tamne ili veoma osvijetljene slike nakon binarizacije predstavlja delta funkciju na bijelom kraju skale histograma. Iz toga možemo zaključiti da će slika biti potpuno bijela i sve informacije na slici izgubljene.

U slučaju da je riječ o tamnoj slici bez djelovanja šuma, ovaj problem se može riješiti ekvalizacijom histograma. Međutim, kamere koje se koriste danas zasigurno će uzrokovati pojavu šuma pri niskom osvjetljenju. Prilikom jakog osvjetljenja kamere zamućuju sliku, što opet dovodi do gubitka kvalitete slike.

Slika 5.10: Eksperimentalni rezultati dobijeni za različito osvjetljenje

* * *

Implementacija OCR aplikacije je rađena u OpenCV okruženju. Rješavanjem problema lokalne adaptivne binarizacije značajno je doprinijelo kvaliteti dobijenih slika. Tesseract se pokazao kao dobar alat za prepoznavanje znakova za slike jako velikih i jako malih dimenzija, također ostvaruje dobre rezultate pri različitim vrstama fontova (osim kurzivna). Brzina izvršavanja varira u odnosu na specifikacije računara, dok visok stepen tačnosti zahtijeva pojedinačno podešavanje parametara za svaku sliku.

Zaključak

Potrebe za optičkim prepoznavanjem karaktera sve više rastu kako se digitalizacija razvija. OCR danas je potreban čak i u svakodnevnom životu za pretraživanje knjiga i dokumenata.

U okviru ovog rada implementirana je aplikacija za razvoj karaktera korištenjem jednog od *open source* softvera. Poređenjem rezultata najčešće korištenih softvera i preporukama literature, odabran je Tesseract. Zatim ispitivanjem nekih od osnovnih tipova grešaka na digitalnim slikama, ustanovljeno je da predprocesiranje ima značajan utjecaj na sam finalni rezultat OCR sistema. Odnosno, ukoliko je predprocesiranje ispravno odrađeno može se obezbijediti jako visok stepen preciznosti, ali isto tako ukoliko je faza predprocesiranja loša, može uzrokovati da Tesseract ne prepozna postojanje bilo kakvih karaktera na slici. Najveći izazov pri predprocesiranju predstavlja binarizacija slike. Značajan gubitak informacija može nastati usljed neuniformnog osvjetljenja, što predstavlja veliki problem za binarizaciju. Testiranjem brojnih funkcija za binarizaciju koje se nalaze u bibliotekama OpenCV-a, najbolji rezultati su dobijeni lokalnom adaptivnom binarizacijom. Nakon predprocesiranja slijedi faza segmentacije, gdje se izdvajaju regije teksta, a potom i pojedinačnih karaktera.

Tesseract će veoma precizno izdvojiti tekst, pojava šuma će uzrokovati čitanje karaktera koji se ne nalaze na slici. Brzina Tesseract-a će ovisiti o karakteristikama računara na kojem se vrši prepoznavanje karaktera. Na vrijeme izvršavanja će utjecati i količina teksta. OCR aplikacije se uglavnom koriste za čitanje karaktera sa cijelog papira, stoga je neophodno da se prepoznavanje karaktera brzo izvršava. Google je do sada razvio 4 verzije Tesseract-a, zbog čega softver daje dobre rezultate nad slikama sa velikim rasponom dimenzija slike i različitim vrstama fontova.

Smjernice za daljnji rad

U budućnosti razvoja ove aplikacije potrebno je riješiti problem prepoznavanja kurzivnog teksta, što se može uraditi kreiranjem vlastite baze podataka za treniranje (eng. *traineddata*). Na taj način se može omogućiti i prepoznavanje karaktera rukopisa.

Potrebno je također automatizovati aplikaciju, s obzirom da u ovoj fazi još uvijek zahtijeva dosta ručnog rada posebno kada je riječ o upotrebi aplikacije korištenjem kamere u realnom svijetu (ETFCam). Pokretna traka na uređaju se kreće mnogo brže od brzine kamere. Jedno od rješenja je povezivanje sa senzorom koji se nalazi na uređaju ETFCam. Akvizija slike bi se vršila kada senzor detektuje predmet na pokretnoj traci. Drugo rješenje je promjena kamere čija brzina će biti uporediva sa brzinom pokretne trake.

Prilozi

Prilog A

Prototipi funkcija

A.1 Inicijalizacija Tesseract-a

Prototip funkcije za inicijalizaciju Tesseract-a:

int Init(const char datapath, const char* language, OcrEngineMode oem)*

Argumenti ove funkcije:

- **datapath**: Ovo je putanja do datoteke tessdata osnovnog direktorija. Put mora završavati s znakom kose crte /. Tessdata direktorij sadrži instalirane datoteke za jezike. Prosljeđivanjem nullptr ovom parametru Tesseract će tražiti u svom instalacijskom folderu.
- **language (jezik)**: Riječ od tri slova s oznakom jezika (npr. eng za engleski, deu za njemački ili bos za bosanski). Tesseract podržava učitavanje višejezičnih kodova znakom +. Dakle, bos + eng će učitati i engleski i bosanski jezik, s tim da se prednost daje jeziku koji je prvi naveden, u ovom slučaju to je bosanski. Jezike koje koristimo prethodno moramo instalirati jer uz instalaciju Tesseract-a jedini ugrađeni jezik je engleski. Konfiguracija jezika može odrediti da se dva ili više jezika moraju učitati zajedno. Tilda se koristi za zabrane korištenja kombinacija jezika. Na primjer, upotreba bos + eng osigurava da se engleski ne učitava sa bosanskim, čak i ako je konfigurisan za to.
- **OcrEngineMode** predsatvljaju OCR algoritme koji se koriste. Mogu imati jednu od sljedećih vrijednosti:
  - OEM_TESSERACT_ONLY: Koristi samo Tesseract. To je najbrža metoda, ali ima i manju preciznost.
  - OEM_CUBE_ONLY: Sporiji, ali precizniji algoritam. Radi samo ukoliko je korišteni jezik istreniran za podršku ovom načinu rada sistema. U slučaju da se u tessdata folderu nalazi .cube datoteka za korišteni jezik onda je omogućena upotreba ovog algoritma.
  - OEM_TESSERACT_CUBE_COMBINED: Kombinacija prethodna algoritma ostvaruje najbolje rezultate za klasifikaciju. Posljedica toga je najsporije vrijeme izvršavanja.
  - OEM_DEFAULT: Pokušava zaključiti strategiju koja se temelji na konfiguracijskoj datoteci jezika, konfiguracijskoj datoteci komandne linije, ili u općem slučaju koristi OEM_TESSERACT_ONLY.

A.2 Segmentacija stranice upotrebom Tesseract-a

Prototip funkcije pomoću kojeg se određuje način segmentacije stranice:

$$void\ SetPageSegMode(tesseract :: SegmentationMode)$$

Parametar *SegmentationMode* određuje način segmentacije stranice, vrijednosti koje može uzimati su:

- PSM_OSD_ONLY: Koristeći ovaj način, Tesseract samo pokreće svoje algoritme prethodne obrade kako bi otkrio orijentaciju i detekciju teksta.
- PSM_AUTO_OSD: Tesseract automatski izvršava segmentaciju stranice sa orijentacijom i detekcijom teksta
- PSM_AUTO_ONLY: Vrši samo segmentaciju stranice
- PSM_AUTO: Vrši segmentaciju i OCR
- PSM_SINGLE_COLUMN: Pretpostavlja da se tekst u kolonama javlja u različitim veličinama
- PSM_SINGLE_BLOCK_VERT_TEXT: Tretira sliku kao jedan jedinstveni blok vertikalno poravnanog teksta.
- PSM_SINGLE_BLOCK: Predstavlja zadanu konfiguracija koja smatra da postoji jedan blok teksta.
- PSM_SINGLE_LINE: Slika sadrži samo jedan redak teksta.
- PSM_SINGLE_WORD: Slika sadrži samo jednu riječ.
- PSM_SINGLE_WORD_CIRCLE: Slika samo jedna riječ koja se nalazi u krugu.
- PSM_SINGLE_CHAR: Slika sadrži jedan znak

A.3 Adaptivna binarizacija

Prototip funkcije koja se koristi za adaptivnu binarizaciju je:

void cv::adaptiveThreshold(src, dst, maxValue, adaptiveMethod, thresholdType, blockSize, C);

Argumenti ove funkcije su:

- **src** - ulazna monohromatska slika
- **dst** - odredišna slika iste veličine i istog tipa kao *src*
- **maxValue** - nenulta vrijednost dodijeljena pikselima za koje je uvjet A.1 ili A.2 zadovoljen
- **adaptiveMethod** - algoritam adaptivnog praga za upotrebu, *ADAPTIVE_THRESH_MEAN_C* ili *ADAPTIVE_THRESH_GAUSSIAN_C*.

- **thresholdType** - vrsta binarizacije koji mora biti $THRESH\_BINARY$ ili $THRESH\_BINARY\_INV$.
- **blockSize** - veličina susjednih piksela koja se koristi za izračunavanje vrijednosti praga za piksel. Može imati samo neparnu vrijednost počevši od 3.
- **C** - konstanta koja se oduzima od srednje vrijednosti ili težinske srednje vrijednosti. Obično je pozitivan broj, ali može biti i nula ili negativno.

$$THRESH\_BINARY : dst(x,y) = \begin{cases} \text{maxValue}, & \text{if } src(x,y) > T(x,y). \\ 0, & \text{inače.} \end{cases} \tag{A.1}$$

$$THRESH\_BINARY\_INV : dst(x,y) = \begin{cases} 0, & \text{if } src(x,y) > T(x,y). \\ \text{maxValue}, & \text{inače.} \end{cases} \tag{A.2}$$

Prilog B

Kodovi za implementaciju OCR-a

Program B.1: Funkcija za binarizaciju slike

```
void MainWindow::binary()
{
    int thresh_value = ui->spinbox->value();
    adaptiveThreshold(slika,slika, 255, ADAPTIVE_THRESH_MEAN_C,
        CV_THRESH_BINARY, thresh_value,12);
}
```

Program B.2: Funkcije za korištenje kamere

```
//Pokretanje kamere
void MainWindow::on_camerabtn_clicked()
{
    capture = VideoCapture(0);
    if(capture.isOpened() == false)
    {
        std::cout << "Kamera␣nije␣otvorena"<<std::endl;
    }
    timer->start(1);
}

void MainWindow::updateFrame()
{
    capture >> frame;
    QImage prikaznaSlika = QImage((const unsigned char*)frame.data,
        frame.cols,frame.rows,frame.step,QImage::Format_RGB888);
    prikaznaSlika= prikaznaSlika.scaled(ui->lbl_image->size(), Qt::
        KeepAspectRatio);
    k1=double(prikaznaSlika.height())/frame.rows;
    k2 = double(prikaznaSlika.width()) / frame.cols;
    ui->lbl_image->setPixmap(QPixmap::fromImage(prikaznaSlika));
    ui->lbl_image->setAlignment(Qt::AlignTop | Qt::AlignLeft);
}

//Zaustavljanje kamere
void MainWindow::on_camerastopbtn_clicked()
{
    capture.release();
    timer->stop();
}
```

Program B.3: Funkcija za postavljanje slike sa računara

```
void MainWindow::on_btn_image_clicked()
{
    QString filename = QFileDialog::getOpenFileName(this, tr("Choose
        "), "", tr("Images␣(*.tiff␣*.png␣*.jpeg␣*.jpg␣␣*.bmp␣*.gif␣
        *.tif)"));
    if(QString::compare(filename, QString()) != 0 )
    {
        bool valid = image.load(filename);

        if(valid)
        {
            image = image.scaledToWidth(ui->lbl_image->width(), Qt::
                SmoothTransformation);
            image = image.scaledToHeight(ui->lbl_image->height(), Qt
                ::SmoothTransformation);
            image = image.scaled(ui->lbl_image->size(), Qt::
                KeepAspectRatio);
            ui->lbl_image->setPixmap(QPixmap::fromImage(image));
            ui->lbl_image->setAlignment(Qt::AlignTop | Qt::AlignLeft
                );
            slika = Qimage2Mat(image);
            oldSlika=slika;
            ui->checkBox_cs->setChecked(false);
            ui->checkBox_histeq->setChecked(false);
            ui->checkBox_median->setChecked(false);
            timesClicked = 0;
        }
    }

}
```

Program B.4: Funkcija za mijenjanje slike u QLabel-u

```
void MainWindow::update_image()
{
    QPixmap imgIn = Mat2QPixmap(slika);
    ui->lbl_image->setPixmap(imgIn);
    ui->lbl_image->show();
}
```

Program B.5: Funkcija za prepoznavanje karaktera

```
void MainWindow::identifyText()
{
    Pix* image1 = mat8ToPix(&slika);
    tesseract::TessBaseAPI *api = new tesseract::TessBaseAPI();
        if (api->Init(NULL, "eng+bos")) {
                fprintf(stderr, "Could␣not␣initialize␣tesseract.\n");
                exit(1);
        }
    double e1 = getTickCount();
    api->SetImage(image1);
    outText = api->GetUTF8Text();
    double e2 = getTickCount();
    double time = (e2 - e1)/getTickFrequency();
    new_time = new_time + time;
    timesClicked++;
```

```
16      double averageTime = new_time/timesClicked;
17      std::cout<<"OCR␣output:␣"<<std::endl <<  outText<< std::endl;
18      std::cout<< "Vrijeme␣izvrsavanja␣trenutno:␣"<< time <<std::endl;
19      std::cout<< "Vrijeme␣izvrsavanja␣prosjecno:␣" << averageTime <<
           std::endl;
20      std::cout<< "Broj␣izvrsavanja␣Tesseract-a:␣" << timesClicked <<
           std::endl;
21  }
```

Program B.6: Funkcije za rezanje i rotaciju slike

```
1   //Gonji lijevi piksel izrezane slike
2   void MainWindow::on_btn_point_clicked()
3   {
4       ui->spinBox_x->setValue(x_koordinata);
5       ui->spinBox_y->setValue(y_koordinata);
6       new_x_koordinata=x_koordinata;
7       new_y_koordinata=y_koordinata;
8   }
9
10  //Donji desni piksel izrezane slike
11  void MainWindow::on_btn_point2_clicked()
12  {
13      ui->spinBox_x2->setValue(x_koordinata);
14      ui->spinBox_y2->setValue(y_koordinata);
15      width_koordinata =abs(x_koordinata - new_x_koordinata);
16      height_koordinata = abs(y_koordinata - new_y_koordinata);
17  }
18
19  //Rezanje slike
20  void MainWindow::crop()
21  {
22      cv::Rect myROI(int(new_x_koordinata/k1), int(new_y_koordinata/k2
           ), int(width_koordinata/k1), int(height_koordinata/k2));
23      cv::Mat novaSlika2 = slika(myROI);
24      slika = novaSlika2.clone();
25  }
26
27  //Rotiranje slike
28  void MainWindow::on_rotate_clicked()
29  {
30      cv::rotate(slika, slika, cv::ROTATE_90_CLOCKWISE);
31      update_image();
32  }
```

Program B.7: Funkcija za dodavanje tehnika predprocesiranja

```
1   void MainWindow::check_status()
2   {
3       if(ui->checkBox_median->isChecked())
4       {
5           savedImage_median = slika.clone();
6           GaussianBlur(slika,slika, Size(5,5), 0);
7       }
8       if (ui->checkBox_cs->isChecked())
9       {
10          savedImage_cs = slika.clone();
11          cv::normalize(slika, slika, 200, 250, cv::NORM_MINMAX);
```

```
12      }
13      if (ui->checkBox_histeq->isChecked())
14      {
15          savedImage_histeq = slika.clone();
16          cv::equalizeHist( slika, slika );
17      }
18  }
```

Program B.8: Funkcija koja izvršava predprocesiranje i pokreće prepoznavanje karaktera

```
1   void MainWindow::preprocess()
2   {
3       ui->tesseractOutput->clear();
4       cv::cvtColor(slika, slika,cv::COLOR_BGR2GRAY);
5       check_status();
6       binary();
7       crop();
8       QImage prikaznaSlika = Mat2QImage(slika);
9       if(capture.isOpened() == true){
10          prikaznaSlika =prikaznaSlika.scaledToWidth(ui->
               label_threshold->width(), Qt::SmoothTransformation);
11          prikaznaSlika = prikaznaSlika.scaledToHeight(ui->
               label_threshold->height(), Qt::SmoothTransformation);
12          prikaznaSlika= prikaznaSlika.scaled(ui->label_threshold->
               size(), Qt::KeepAspectRatio);
13          ui->label_threshold->setPixmap(QPixmap::fromImage(
               prikaznaSlika));
14          ui->label_threshold->setAlignment(Qt::AlignTop | Qt::
               AlignLeft);
15      }
16      else if(capture.isOpened() == false)
17      {
18          update_image();
19      }
20      identifyText();
21  }
```

Literatura

[1] Gabasio, A., Comparison of optical character recognition (OCR) software. Department of Computer Science, Faculty of Engineering, LTH, Lund University, 2013.

[2] Nagy, G., Nartker, T. A., Rice, S. V., “Optical character recognition: An illustrated guide to the frontier”, in Document Recognition and Retrieval VII, Vol. 3967. International Society for Optics and Photonics, 1999, str. 58–69.

[3] Eikvil, L., “Optical character recognition”, citeseer. ist. psu. edu/142042. html, 1993.

[4] Chakraborty, P., Mallik, A., “An open source tesseract based tool for extracting text from images with application in braille translation for the visually impaired”, International Journal of Computer Applications, Vol. 68, No. 16, 2013.

[5] Kaehler, A., Bradski, G., Learning OpenCV 3: computer vision in C++ with the OpenCV library. " O’Reilly Media, Inc.", 2016.

[6] Mithe, R., Indalkar, S., Divekar, N., “Optical character recognition”, International journal of recent technology and engineering (IJRTE), Vol. 2, No. 1, 2013, str. 72–75.

[7] Li, Y., Lopresti, D., Nagy, G., Tomkins, A., “Validation of image defect models for optical character recognition”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18, No. 2, 1996, str. 99–107.

[8] “Best font and size for optical character recognition (ocr)”, dostupno na: http://www.cvisiontech.com/library/pdf/pdf-ocr/best-font-and-size-for-ocr.html (juli, 2019.).

[9] Turajlić, E., “Predavanje 3 - tehnologije televizije”. Elektrotehnički fakultet Sarajevo, 2019.

[10] Shafait, F., Keysers, D., Breuel, T. M., “Efficient implementation of local adaptive thresholding techniques using integral images”, in Document recognition and retrieval XV, 2008, str. 510-681.

[11] Chaudhuri, A., Mandaviya, K., Badelia, P., Ghosh, S. K., “Optical character recognition systems”, in Optical Character Recognition Systems for Different Languages with Soft Computing. Springer, 2017, str. 9–41.

[12] Mori, S., Nishida, H., Yamada, H., Optical character recognition. John Wiley & Sons, Inc., 1999.

[13] Peng, X., Cao, H., Setlur, S., Govindaraju, V., Natarajan, P., “Multilingual ocr research and applications: an overview”, in Proceedings of the 4th International Workshop on Multilingual OCR. ACM, 2013, str. 1.

[14] Sonka, M., Hlavac, V., Boyle, R., Image processing, analysis, and machine vision. Cengage Learning, 2014.

[15] Alginahi, Y., “Preprocessing techniques in character recognition”, in Character recognition. InTechOpen, 2010.

[16] Joshi, P., Escrivá, D. M., Godoy, V., OpenCV By Example. Packt Publishing Ltd, 2016.

[17] Smith, R., “An overview of the tesseract ocr engine”, in Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), Vol. 2. IEEE, 2007, str. 629–633.

[18] Smith, R., Antonova, D., Lee, D.-S., “Adapting the tesseract open source ocr engine for multilingual ocr”, in Proceedings of the International Workshop on Multilingual OCR. ACM, 2009, str. 1.