Detekcija označenih odgovora na pisanom ispitu primjenom tehnika procesiranja slike

Nižić, Luka Krševan

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Krševan Nižić

DETEKCIJA OZNAČENIH ODGOVORA NA PISANOM ISPITU PRIMJENOM TEHNIKA PROCESIRANJA SLIKE

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Krševan Nižić

JMBAG: 0016155331

Studij: Informacijski i poslovni sustavi

Detekcija označenih odgovora na pisanom ispitu primjenom tehnika procesiranja slike

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Marko Mijač

Varaždin, ožujak 2024.

Luka Krševan Nižić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog završnog rada je "Detection of marked answers on written exam using image processing techniques". Rad se bavi razvojem softverskog rješenja koje koristi tehnike procesiranja slike za automatsko prepoznavanje i ocjenjivanje označenih odgovora na pisanom ispitu. U teorijskom dijelu rada obrađuju se temeljne tehnologije, algoritmi, alati i biblioteke koje omogućuju i olakšavaju procesiranje slike. Poseban naglasak stavljen je na metode za predprocesiranje slika te detekciju objekata na slikama te na korištenje OpenCV biblioteke za implementaciju tih metoda. Praktični dio rada uključuje specifikaciju, modeliranje i implementaciju softvera koji koristi tehniku binarizacije slike, detekciju kontura i algoritme za sortiranje kontura kako bi prepoznao i ocijenio označene odgovore. Softver je testiran na različitim slikama ispitnih listova kako bi se ocijenila njegova učinkovitost i točnost. Rezultati evaluacije pokazuju da softver može precizno prepoznati označene odgovore uz minimalne pogreške pod optimalnim uvjetima. Testiranjem programa u realnoj okolnosti te njegovom prilagodbom potvrđuje se njegova korisnost u automatizaciji procesa ocjenjivanja ispita. Zaključci rada ističu prednosti korištenja tehnika procesiranja slike u obrazovnom sustavu, kao što su ušteda vremena i resursa te povećanje točnosti ocjenjivanja. Također se identificiraju mogući smjerovi za budući rad, uključujući poboljšanje algoritama za prepoznavanje i prilagodbu softvera za rad s različitim vrstama ispitnih listova.

Ključne riječi: procesiranje slike; detekcija kontura; OpenCV; automatizacija ocjenjivanja; binarizacija; računalni vid; grayscale

Sadržaj

Sadržaj iii
1. Uvod 1
2. Osnove digitalnog procesiranja slike 2
2.1. Digitalne slike 2
2.1.1. Tipovi slika 2
2.2. Postupci u digitalnom procesiranju slike 4
2.3. Računalni vid 5
2.4. Usporedba procesiranja slika nasuprot računalnog vida 6
2.4.1. Ciljevi 6
2.4.2. Ulaz i izlaz 6
2.4.3. Opseg 7
2.4.4. Procesiranje ili računalni vid? 7
3. Detekcija oblika na slici 7
3.1. Filtriranje buke 8
3.2. Detekcija rubova 11
3.2.1. Definicija i svrha 11
3.2.2. Algoritmi za detekciju rubova 12
3.3. Segmentacija slike 13
3.3.1. Definicija i svrha 13
3.3.2. Metode segmentacije 14
4. Pregled popularnih biblioteka za procesiranje slika 16
4.1. OpenCV 16
4.1.1. Podržani algoritmi 16
4.1.2. Primjena i utjecaj 16
4.1.3. Kompatibilnost i performanse 17
4.2. Scikit-Image 17
4.2.1. Podržani algoritmi 18
4.2.2. Primjena i utjecaj 18
5. Izrada sustava za detekciju odgovora na ispitu 19
5.1. Slučajevi korištenja 20
5.2. Use Case: Ocijeni ispit 20

5.3. Klase podataka 22
5.3.1. Klasa Test 22
5.3.2. Veza s klasom Grader 23
5.3.3. Klasa Grader 23
5.3.4. Veza s klasom Test: 23
5.3.5. Veza s klasom OMRScanner: 23
5.3.6. Klasa OMRScanner 23
5.4. Slijed programa 24
5.4.1. Inicijalna komunikacija 25
5.4.2. Procesiranje Testa: Grader i OMRScanner 25
5.4.3. Ocjenjivanje Testa: Grader i OMRScanner 26
5.4.4. Prikazivanje Rezultata: Grader i Korisnik 26
5.4.5. Završetak Procesa 26
5.5. Implementacija 26
5.5.1. Detaljna analiza klase Test 27
5.5.2. Detaljna analiza klase OMRScanner 28
5.5.3. Detaljna analiza klase Grader 31
5.5.4. Pokretanje programa: Funkcija main 34
5.6. Analiza Rezultata 35
5.6.1. Poboljšanja 37
5.7. Testiranje rješenja u realnim okolnostima 40
5.7.1. Ciljevi 40
5.7.2. Priprema Testova 40
5.7.3. Problemi u Detekciji i Njihova Rješenja 42
5.7.4. Implementacija API-ja 43
5.7.5. Primjer rada 46
6. Zaključak 50
Popis literature 51
Popis slika 53

1. Uvod

Tema „Detekcija označenih odgovora na pisanom ispitu primjenom tehnika procesiranja slike“ značajna je jer automatizacija procesa ocjenjivanja ispita može značajno povećati učinkovitost i točnost u obrazovnom sustavu. Ručno ocjenjivanje ispita je vremenski zahtjevan proces sklon pogreškama, dok automatizirani sustavi mogu uštedjeti vrijeme i resurse te osigurati dosljednost i preciznost (Balla, Erika, 2023). Moja motivacija za odabir ove teme dolazi iz mog osobnog interesa za računalni vid i tehnike procesiranja slike. Tijekom studija, posebno me zaintrigirala domena procesiranja slika i njena praktična primjena na stvarne probleme. Ova tema mi omogućuje da spojim teorijsko znanje s praktičnom primjenom, čime bih mogao doprinijeti unapređenju obrazovnog sustava. Vjerujem da automatizacija, posebno u sferi obrazovanja, ima ogroman potencijal, a područje poput ocjenjivanja ispita nudi mogućnosti za implementaciju naprednih tehnologija, kao što su računalni vid i algoritmi za prepoznavanje obrazaca. Ovaj rad je prilika da iskoristim svoje znanje i vještine kako bih pridonio razvoju ovakvih sustava.

Glavni cilj mog rada je razvoj softverskog rješenja koje će precizno detektirati i ocjenjivati označene odgovore na pisanom ispitu. Ovo uključuje implementaciju algoritama za obradu slike koji mogu analizirati odgovore i identificirati točne ili netočne oznake. Sekundarni ciljevi uključuju procjenu učinkovitosti softvera, analiziranje njegovih rezultata i identifikaciju potencijalnih poboljšanja koja bi mogla unaprijediti točnost i pouzdanost sustava. Testiranje softvera bit će ključno za procjenu njegovih mogućnosti u stvarnim uvjetima.

Struktura rada podijeljena je na nekoliko dijelova. U teorijskom dijelu bit će obrađene ključne metode procesiranja slika, uključujući temeljne tehnologije i algoritme koji se koriste za prepoznavanje označenih odgovora. Ovaj dio rada obuhvatit će pregled algoritama poput prepoznavanja obrazaca, optičkog prepoznavanja znakova (OCR) i segmentacije slike. Nakon teorijskog pregleda slijedi praktični dio u kojem će biti opisani postupci modeliranja, specifikacije i implementacije softvera za automatsku detekciju označenih odgovora.

Posebna pažnja bit će posvećena evaluaciji razvijenog softvera. U ovoj fazi analizirat ćemo točnost prepoznavanja odgovora, kao i prednosti i nedostatke samog rješenja. Evaluacija će također uključivati prijedloge za buduća poboljšanja, s ciljem daljnjeg unapređenja performansi softverskog sustava. Završni dio rada donosi zaključak koji obuhvaća sažetak postignutih rezultata te preporuke za buduće radove u ovom području. Dodatno, softversko rješenje bit će testirano u realnim okolnostima pretvaranjem programa u

API. Time će se omogućiti šira primjena softvera u različitim obrazovnim ustanovama i sustavima, čime se potvrđuje njegova fleksibilnost i praktičnost za stvarnu primjenu.

2. Osnove digitalnog procesiranja slike

Oko je najvažnije ljudsko osjetilo kojim primamo 90% svih informacija iz okoline pa nije iznenađujuće da slike igraju ulogu u ljudskoj percepciji svijeta. Međutim, za razliku od ljudi, koji su ograničeni na vizualni pojas elektromagnetskog (EM) spektra, slikovni uređaji pokrivaju gotovo cijeli EM spektar, od gama zraka do radio valova (Gonzalez i ostali, 2009). Autor također opisuje uređaje za snimanje fotografija koji mogu obuhvatiti čitav elektromagnetski spektar, omogućujući analizu slika iz izvora koje ljudi nisu navikli asocirati sa slikama, kao što su ultrazvuk i elektronska mikroskopija te računalno generirane slike te se time može potkrijepiti raznolika primjena digitalnog procesiranja. Da bi se razumjele temeljne tehnologije procesiranja slike treba odgovoriti na pitanja kakve su to slike koje procesiramo te koje se sve discipline bave procesiranjem slika.

2.1. Digitalne slike

Svaka digitalna slika može se predstaviti kao dvodimenzionalna funkcija f(x,y) koja ima prostorne koordinate (x,y) i vrijednosti intenziteta, odnosno amplitude f, u svakoj točki koja prikazuje razinu sive slike iste točke. Kada su sve prostorne koordinate i njihove amplitude konačno definirane, onda se slika može nazivati digitalnom slikom. Digitalna slika sastoji se od konačnog broja elemenata zvanih pikseli, od kojih svaki ima svoju jedinstvenu lokaciju unutar slike kao i specifičnu vrijednost (Gonzalez i ostali, 2009)Ova definicija postavlja osnovu za daljnje razumijevanje kako se slike obrađuju na digitalnim računalima.

2.1.1. Tipovi slika

U računalnom vidu i obradi slika, slike se kategoriziraju prema intenzitetu piksela i prikazima boja. Slijede objašnjenja glavnih tipova slika.

2.1.1.1. RGB obojena slika (16-bitni format boja)

RGB obojene slike sastoje se od tri kanala boja: crveni (Red), zeleni (Green) i plavi (Blue). Svaki kanal u RGB slici može imati 65.536 različitih vrijednosti boja, čime ovo postaje 16-bitni format. Ova kombinacija omogućuje širok raspon boja koje se mogu prikazati na slici.

RGB format je najčešće korišteni format boja u modernom digitalnom snimanju (Kumar , N, 2018) (Kundu, 2024)

Slika 1: Primjer RGB slike: Šalica čaja (Laurel Fan, Izvor: Internet)

2.1.1.2. Binarna slika

Binarna slika sadrži samo dvije vrijednosti piksela: 0 i 1. Ovdje 0 predstavlja crnu, a 1 bijelu boju. Zbog svoje jednostavnosti, ovaj tip slike često se koristi za zadatke poput segmentacije slike, gdje je cilj istaknuti određene dijelove obojene slike (Kundu, 2024). Binarne slike također su poznate kao monokromatske slike (Kumar , N, 2018).

Slika 2: Primjer binarizacije (Izvor: Vlastita izrada)

2.1.1.3. Siva slika (8-bitni format boja)

Sive slike, također poznate kao 8-bitne slike, sastoje se od 256 nijansi sive boje, u rasponu od krajnje crne (0) do bijele boje (255). U ovom formatu:

- 0 predstavlja crnu boju.
- 255 predstavlja bijelu boju.
- Vrijednosti između 0 i 255 predstavljaju različite nijanse sive boje (Kumar , N, 2018).
- Sive slike su često korištene jer pružaju detaljan prikaz slika uz smanjenu složenost u usporedbi s obojenim slikama (Kundu, 2024)

Slika 3: Primjer greyscalea (Izvor: Vlastita izrada)

2.2. Postupci u digitalnom procesiranju slike

Digitalna obrada slika je temeljno područje unutar računalnih znanosti i digitalne tehnologije koje se bavi manipulacijom i analizom digitalnih slika. Cilj digitalne obrade slika je poboljšati kvalitetu slika, izdvojiti značajne informacije iz slika i automatizirati zadatke temeljene na slikama (Kumar , N, 2018). Obuhvaća širok raspon tehnika i algoritama usmjerenih na poboljšanje, izdvajanje informacija ili izmjenu digitalnih slika kako bi se postigli specifični ciljevi. Suštinski, obrada slika uključuje niz operacija izvedenih na osnovi piksela, gdje svaki piksel odgovara malenom elementu slike. Te operacije mogu uključivati filtriranje za uklanjanje šuma, promjenu veličine radi prilagodbe dimenzija slike, podešavanje boje za

korekciju svjetline i kontrasta, te složene zadatke poput detekcije i prepoznavanja objekata koristeći napredne algoritme strojnog učenja (Kundu, 2024). Osnovni koraci u digitalnoj obradi slika:

- Dobivanje slike: Ovo uključuje snimanje slike digitalnim fotoaparatom ili skenerom, ili uvoz postojeće slike u računalo.
- Poboljšanje slike: Ovo se odnosi na poboljšanje vizualne kvalitete slike, kao što je povećanje kontrasta, smanjenje šuma i uklanjanje artefakata.
- Obnova slike: Ovo uključuje uklanjanje degradacije sa slike, poput zamućenja, šuma i izobličenja.
- Segmentacija slike: Ovo uključuje dijeljenje slike na regije ili segmente, od kojih svaki odgovara specifičnom objektu ili značajki na slici.
- Reprezentacija i opis slike: Ovo uključuje predstavljanje slike na način koji omogućava računalu analizu i manipulaciju te opisivanje značajki slike na kompaktan i smislen način.
- Analiza slike: Ovo uključuje korištenje algoritama i matematičkih modela za izdvajanje informacija iz slike, kao što je prepoznavanje objekata, detekcija uzoraka i kvantifikacija značajki.
- Sinteza i kompresija slike: Ovo uključuje generiranje novih slika ili komprimiranje postojećih slika radi smanjenja zahtjeva za pohranu i prijenos.

Digitalna obrada slika široko se koristi u raznim područjima, uključujući medicinsko snimanje, daljinsko istraživanje, računalni vid i multimediju (Kumar , N, 2018).

2.3. Računalni vid

Računalni vid (eng. Computer vision) je područje koje se bavi razvojem matematičkih tehnika za dobivanje trodimenzionalnog oblika i izgleda objekata iz slika. Iako ljudi lako percipiraju trodimenzionalnu strukturu svijeta oko sebe, računalni vid nastoji postići isto koristeći algoritme za analizu slika. To uključuje stvaranje 3D modela iz niza fotografija, praćenje objekata u složenim pozadinama i prepoznavanje lica na fotografijama. Međutim, potpuno razumijevanje i interpretacija slike na razini dvogodišnjeg djeteta još uvijek je nedostižan cilj za računalni vid. Ova složenost proizlazi iz prirode vizije kao inverznog problema gdje je potrebno rekonstruirati nepoznate informacije iz ograničenih podataka, koristeći fizičke i probabilističke modele kako bi se razjasnile potencijalne interpretacije (Szeliski, 2011).

Računalni vid funkcionira slično kao i ljudski vid, ali ljudi imaju prednost dugogodišnjeg iskustva. Ljudski vid koristi kontekst iz cijelog života kako bi naučio razlikovati objekte, procijeniti njihovu udaljenost, prepoznati kretanje ili uočiti probleme na slici (*What Is Computer Vision?*, 2021).

Računalni vid trenira strojeve da obavljaju ove funkcije u mnogo kraćem vremenu koristeći kamere, podatke i algoritme umjesto mrežnice, optičkih živaca i vizualnog korteksa. Sustav koji je obučen za pregled proizvoda ili nadzor proizvodnih procesa može analizirati tisuće proizvoda i procesa u minuti, uočavajući neprimjetne nedostatke ili probleme, te tako brzo nadmašiti ljudske sposobnosti (*What Is Computer Vision?*, 2021).

2.4. Usporedba procesiranja slika nasuprot računalnog vida

Unatoč njihovoj bliskoj povezanosti, područja računalnog vida i obrade slike nisu ista u smislu ciljeva, ulaza/izlaza, opsega, tehnika i uobičajenih primjena.

2.4.1. Ciljevi

Računalni vid možete zamisliti kao „mozak” koji stoji iza vizualne percepcije. Njegov glavni cilj je dati strojevima isto razumijevanje i interpretaciju vizualnog svijeta kao što to imaju ljudi. Upravljaju složenim zadacima poput razumijevanja scena, prepoznavanja objekata i izvlačenja zaključaka iz slika i videozapisa.

S druge strane, obrada slike je „zanatlija” digitalnih slika. Njezin primarni cilj je poboljšati i raditi s vizualnim elementima slike. To može uključivati poboljšanje kvalitete slika, uklanjanje određenih značajki ili pripremu slika za daljnju analizu. Tipični zadaci obrade slike uključuju izoštravanje, smanjenje šuma i promjenu kontrasta boja (Quan, 2023).

2.4.2. Ulaz i izlaz

U aplikacijama računalnog vida koriste se slike, sekvence slika ili video datoteke kao ulaz. Rezultat je često razumijevanje ili interpretacija scene, koja možda ne uključuje vizualne elemente.

S druge strane, u obradi slike, slika se koristi i kao ulaz i kao izlaz. Obično je rezultat dotjerana ili izmijenjena kopija originalne slike (Quan, 2023).

2.4.3.Opseg

Računalni vid pristupa sceni na sveobuhvatan način, pokušavajući razumjeti cijelu scenu. Njegovi glavni ciljevi su izvući relevantne informacije, razumjeti kontekst i donositi opravdane odluke koristeći vizualne podatke.

Obrada slike fokusira se na lokalizirane, niskorazinske operacije koje utječu na specifične piksele ili male dijelove unutar slike. Bez nužnog razumijevanja sadržaja slike, njezin fokus je na poboljšanju ili promjeni vizualnih izgleda (Quan, 2023).

2.4.4. Procesiranje ili računalni vid?

Nema općeg suglasja među autorima o tome gdje procesiranje slike prestaje, a druge srodne oblasti, kao što su analiza slike i računalni vid, počinju. Ponekad se pravi razlika definiranjem procesiranja slike kao discipline u kojoj su i ulaz i izlaz procesa slike (Gonzalez i ostali, 2009). Autor smatra da nema jasne granice između ovih područja te ako se pridržava definicije procesiranja slike, prema kojoj je to disciplina gdje su ulaz i izlaz slike, onda i najtrivijalni zadatak izračuna intenziteta slike bi ne bi spadao pod procesiranje. Zato se autor zalaže za korisnu paradigmu razlikovanja niskorazinskih, srednjerazinskih i visokorazinskih procesa.

Niskorazinski procesi uključuju osnovne operacije kao što su **smanjenje šuma i poboljšanje kontrasta**. Srednjerazinski procesi uključuju **segmentaciju i prepoznavanje objekata**, dok visokorazinski procesi uključuju **razumijevanje** skupa prepoznatih objekata, što može uključivati kognitivne funkcije povezane s vidom.

Autor na kraju naglašava da digitalno procesiranje slike obuhvaća širok raspon tehnika i aplikacija, uključujući prepoznavanje teksta i analizu njegovog sadržaja. Ovo područje ima iznimnu društvenu i ekonomsku vrijednost, što ga čini ključnim za mnoge suvremene tehnologije.

3. Detekcija oblika na slici

U prethodnom odjeljku razmotrene su razlike između različitih grana procesiranja digitalnih slika te kompleksnost svakog pristupa. U ovom odjeljku bit će obrađeni ključni koraci i poznati algoritmi koji omogućuju učinkovitu obradu i analizu slika. Detekcija elemenata na slici u kontekstu računalnog vida i analize slika sastoji se od nekoliko ključnih koraka. Ovi koraci osiguravaju da se relevantni uzorci mogu pouzdano prepoznati, čak i kada su prisutni šum, varijacije u osvjetljenju ili deformacije slike. Tipičan proces obuhvaća:

- **Filtriranje šuma (engl. Noise Filtering)**: Uklanjanje neželjenih artefakata i šuma sa slike radi poboljšanja kvalitete signala i omogućavanja preciznije analize.
- **Detekcija rubova (engl. Edge Detection)**: Identifikacija rubova i granica objekata na slici, što omogućava izdvajanje struktura od interesa.
- **Segmentacija slike (engl. Image Segmentation)**: Podjela slike na različite regije koje predstavljaju logičke dijelove slike, kao što su pojedinačni odgovori na ispitima.

Ovi koraci čine osnovu svakog sustava za analizu slike, a detaljnije su obrađeni u nastavku poglavlja.

3.1. Filtriranje buke

Buka u digitalnim slikama odnosi se na neželjene signale koji izobličuju stvarne informacije unutar slike. Ova buka može biti uzrokovana različitim čimbenicima, uključujući osjetilne greške senzora kamere, prijenosne greške tijekom digitalizacije signala ili nepovoljni uvjeti osvjetljenja prilikom snimanja. Nepravilno rukovanje bukom može značajno utjecati na kvalitetu rezultata u kasnijim fazama obrade slike, uključujući detekciju rubova, segmentaciju i prepoznavanje objekata.

Filtriranje buke (engl. Noise filtering) stoga predstavlja ključni korak u procesiranju slike jer omogućuje smanjenje ili eliminaciju ovih neželjenih elemenata prije nego što se provede bilo koja daljnja analiza ili obrada. Kao takvo, filtriranje buke smatra se osnovnim preprocesom, čija se implementacija smatra neophodnom za postizanje pouzdanih i preciznih rezultata u daljnjim fazama obrade slike, bez obzira na specifičnu svrhu obrade. Neki od glavnih algoritama za filtriranje buke su:

- **Median Filter:** Median Filter je nelinearni digitalni filtar koji se često koristi za uklanjanje impulzivne buke, poznate i kao „buka soli i papra" (engl. salt and pepper noise). Za razliku od drugih filtara, Median Filter uspješno uklanja buku dok istovremeno zadržava oštrinu rubova, što ga čini vrlo pogodnim za slike koje sadrže izražene rubove i fine detalje (*OpenCV: Smoothing Images*, bez dat.). Ovaj algoritam funkcionira tako da za svaki piksel u slici određuje vrijednost medijana u njegovoj blizini te zamjenjuje izvornu vrijednost piksela tom vrijednošću medijana (Szeliski, 2011). U primjeru napravljenom uz OpenCV, prikazuje se slika kojoj je dodana buka prije filtriranja kroz median blur.

Slika 4: Salted grayscale šalica (Izvor: Vlastita izrada)

Slika 5: Primjer Median Blura (Izvor: Vlastita izrada)

- **Gaussov Filter:** Gaussov Filter je linearni filtar koji smanjuje šum zamagljujući sliku primjenom Gaussianove funkcije na okolinu svakog piksela. Gaussova funkcija daje težine obližnjim pikselima na temelju njihove udaljenosti od centralnog piksela, uz to da bliži pikseli imaju veću težinu. Ovo rezultira blago zamagljenom slikom s

reduciranim šumom, pri čemu Gaussian Filter zadržava glatke prijelaze u intenzitetu unutar slike, ali može zamućivati fine detalje i rubove (Szeliski, 2011).

Slika 6: Gaussian filter (Izvor: Vlastita izrada)

- **Bilateralni Filter:** Bilateralni Filter predstavlja sofisticiraniji pristup filtriranju buke, kombinirajući filtriranje prema prostornoj blizini piksela i sličnosti u intenzitetu (Szeliski, 2011). To znači da Bilateralni Filter ne samo da uzima u obzir prostornu udaljenost piksela u okolini, već i koliko su ti pikseli slični po intenzitetu. Rezultat je očuvanje oštrih rubova dok se šum smanjuje, što ga čini izuzetno korisnim za slike gdje je važno očuvati oštrinu rubova, primjerice u medicinskoj ili satelitskoj obradi slika (*OpenCV: Smoothing Images*, bez dat.).

Slika 7: Bilateral filter (Izvor: Vlastita izrada)

Ovi algoritmi predstavljaju osnovu za uklanjanje buke u slikama i ključni su za pripremu slika za daljnju analizu i obradu. Izbor odgovarajućeg algoritma često ovisi o specifičnim karakteristikama slike i vrsti buke koja je prisutna. U mnogim slučajevima, uspješna obrada slike zahtijeva kombinaciju različitih metoda filtriranja kako bi se postigli optimalni rezultati.

3.2. Detekcija rubova

3.2.1. Definicija i svrha

Rubovi unutar digitalne slike predstavljaju nagle promjene u intenzitetu piksela, često označavajući granice između različitih objekata ili područja unutar slike. Ove promjene u intenzitetu mogu odražavati prijelaz između različitih boja, tekstura i svjetlosnih uvjeta, što rubove čini izuzetno važnima za razne zadatke u analizi slike, uključujući prepoznavanje objekata, segmentaciju i analizu scena.

Detekcija rubova (eng. Edge detection) je jedan od temeljnih koraka u procesiranju slike, jer omogućuje identifikaciju ključnih značajki slike koje su od kritične važnosti za razumijevanje njenog sadržaja. Rubovi često sadrže važne informacije o geometriji i strukturi objekata unutar slike, pa njihova pravilna detekcija omogućuje razgraničenje i identifikaciju tih objekata. Bez precizne detekcije rubova, kasniji koraci poput segmentacije ili prepoznavanja

mogu biti značajno otežani ili čak neuspješni jer bi osnovni podaci bili netočni ili nepotpuni (Szeliski, 2011).

3.2.2. Algoritmi za detekciju rubova

Postoji nekoliko algoritama koji su široko korišteni za detekciju rubova, a svaki od njih pristupa problemu na različit način, ovisno o specifičnim zahtjevima i karakteristikama slike. U nastavku su opisani najpoznatiji algoritmi za detekciju rubova:

- **Sobelov Operator:** Sobelov Operator je jedan od najosnovnijih, ali i najčešće korištenih alata za detekciju rubova. Ovaj operator koristi jednostavne konvolucijske maske za izračunavanje gradijenata slike u horizontalnom (x) i vertikalnom (y) smjeru. Korištenjem ovih gradijenata, Sobel Operator identificira rubove gdje dolazi do naglih promjena u intenzitetu piksela. Jedna od glavnih prednosti Sobel Operatora je njegova robusnost u prisutnosti šuma, što ga čini pogodnim za osnovne zadatke u obradi slike (*OpenCV: Sobel Derivatives*, bez dat.).

Slika 8: Primjer detekcije ruba Sobelovim Operatorom (Izvor: Vlastita izrada)

- **Cannyjev Detektor Rubova:** Cannyjev Detektor Rubova (engl. Canny Edge Detector) je složeniji algoritam koji je postao standard u industriji zbog svoje sposobnosti da precizno i pouzdano detektira rubove. Razvijen od strane Johna

Cannyja 1986. godine (Szeliski, 2011), ovaj algoritam koristi višestepeni pristup koji uključuje filtriranje buke pomoću Gaussian filtra, izračunavanje gradijenata slike, praćenje rubova pomoću algoritma „non-maximum suppression" kako bi se eliminirali lažni rubovi, te primjenu histereze za povezivanje rubova. Ova metoda osigurava da detektirani rubovi budu kontinuirani i glatki, čime se značajno poboljšava kvaliteta detekcije u složenim scenama. (*OpenCV: Canny Edge Detection*, bez dat.).

Slika 9: Primjer Cannyjev Detektor Rubova(Izvor: Vlastita izrada)

3.3. Segmentacija slike

3.3.1. Definicija i svrha

Segmentacija slike (engl. Image Segmentation) je proces koji podrazumijeva podjelu slike na više segmenata ili regija koje dijele slične karakteristike, poput boje, intenziteta ili teksture. Svrha segmentacije je pojednostaviti ili promijeniti prikaz slike kako bi postala značajnija i lakša za analizu. Na primjer, u medicinskoj dijagnostici, segmentacija se koristi za identifikaciju i izolaciju određenih struktura unutar slika tijela, dok se u računalnom vidu koristi za prepoznavanje i praćenje objekata u scenama.

Segmentacija je ključni korak u mnogim aplikacijama obrade slike jer omogućuje izolaciju interesnih regija iz pozadine, čime se olakšava daljnja analiza i interpretacija slike. U praksi, segmentacija može biti izuzetno složen proces, pogotovo kada su granice između objekata nejasne ili kada slika sadrži veliku količinu šuma.

3.3.2. Metode segmentacije

Postoji nekoliko metoda za segmentaciju slike, od kojih svaka ima specifične prednosti i mane ovisno o karakteristikama slike i ciljevima analize:

- **Određivanje pragova:**
    - **Globalni prag (engl. Global Thresholding):** Ova metoda koristi jedinstvenu vrijednost praga za cijelu sliku, pri čemu se svi pikseli koji prelaze taj prag klasificiraju kao „prednji plan" (engl. foreground), dok se ostali klasificiraju kao „pozadina". Global Thresholding je jednostavna i brza metoda, ali može biti neučinkovita za slike s neujednačenim osvjetljenjem (*OpenCV: Image Thresholding*, bez dat.).

Slika 10: Prilagodljivi prag (Izvor: Vlastita izrada)

    - **Prilagodljivi prag (engl. Adaptive Thresholding):** Za razliku od globalnog praga, Adaptive Thresholding prilagođava vrijednost praga za male dijelove slike na temelju lokalnih svojstava. Ova metoda je korisna za slike sa značajnim

varijacijama u osvjetljenju, jer omogućuje precizniju segmentaciju (*OpenCV: Image Thresholding*, bez dat.).

Slika 11:Adaptive Thresholding (Izvor: Vlastita izrada)

- **K-srednje grupiranje:** K-srednje grupiranje (engl. K-means Clustering) je algoritam grupiranja koji organizira piksele u k klasa na temelju njihovog intenziteta boja. Ova metoda se koristi za stvaranje homogenih segmenata unutar slike, gdje svaki segment predstavlja grupu piksela sa sličnim svojstvima. K-srednje grupiranje je učinkovit za segmentaciju slika sa jasno definiranim klasterima, ali može biti izazovan za slike s nejasnim granicama između klasa (Szeliski, 2011)

Slika 12: K-means Clustering(Izvor: Vlastita izrada)

4. Pregled popularnih biblioteka za procesiranje slika

Za problem detekcije jednostavnih odgovora na ispitu koristeći tehnike obrade slike, rješenja mogu ponuditi popularne biblioteke kao što su OpenCV i scikit-image. S obzirom na to da opseg ovog završnog rada ne obuhvaća konvolucione i neuronske mreže, neće se razmatrati okvir Keras. Budući da je fokus na grafičkim odgovorima, a ne na tekstualnim, alat Tesseract se također neće razmatrati.

4.1. OpenCV

OpenCV, odnosno otvorena biblioteka za računalni vid, postala je temelj u području računalnog vida i strojnog učenja, nudeći robusni okvir za razvoj sofisticiranih aplikacija za vid. Dizajnirana da pruži zajedničku infrastrukturu za takve aplikacije, OpenCV značajno ubrzava integraciju sposobnosti strojne percepcije u komercijalne proizvode. Njena otvorena priroda, pod Apache 2 licencom, bila je ključna u omogućavanju pristupa naprednoj tehnologiji vida i njenoj modifikaciji za poslovne svrhe, potičući inovacije u raznim industrijama. Jedan od primarnih ciljeva OpenCV-a je pružiti jednostavnu infrastrukturu koja omogućuje programerima da brzo i učinkovito izgrade sofisticirane aplikacije za računalni vid. Kako bi to postigao, OpenCV uključuje više od 500 funkcija koje pokrivaju širok raspon zadataka vezanih uz vid. Ovi zadaci obuhvaćaju više domena, uključujući inspekciju proizvoda u tvornicama, medicinsko snimanje, sigurnosne sustave, dizajn korisničkog sučelja, kalibraciju kamera, stereo vid i robotiku (Kaehler & Bradski, 2017).

4.1.1. Podržani algoritmi

Biblioteka se može pohvaliti opsežnom kolekcijom od preko 2.500 optimiziranih algoritama, koji obuhvaćaju širok spektar klasičnih i najnovijih tehnika. Ovi algoritmi omogućuju širok raspon funkcionalnosti, od prepoznavanja i detekcije lica do praćenja pokretnih objekata, ekstrakcije 3D modela, pa čak i poboljšanja kvalitete slike uklanjanjem artefakata poput crvenih očiju. Štoviše, mogućnosti OpenCV-a proširuju se na područja proširene stvarnosti, gdje može prepoznati pejzaže i postaviti markere za prekrivanje virtualnih elemenata, te na video analizu, gdje može klasificirati ljudske akcije i pratiti kretanje kamere.

4.1.2. Primjena i utjecaj

S korisničkom zajednicom koja prelazi 47.000 pojedinaca i više od 18 milijuna preuzimanja, utjecaj OpenCV-a je širok i dubok. Njegova primjena nije ograničena na određeni

sektor, već se proteže kroz razne domene, uključujući tehnološke divove poput Googlea, Yahooa, Microsofta i Intela, kao i rastuće startupove poput Applied Mindsa i VideoSurfa. Ove organizacije koriste OpenCV za različite svrhe; od spajanja slika za Google Street View do osiguravanja sigurnosti u sustavima nadzora i poboljšanja sposobnosti robotike u istraživačkim okruženjima poput Willow Garage. Razumijevajući da se računalni vid i strojno učenje često nadopunjuju, OpenCV također sadrži sveobuhvatan modul za strojno učenje (ML). Ova podbiblioteka je dizajnirana za statističko prepoznavanje uzoraka i klasterizaciju, što je čini izuzetno vrijednom za zadatke vezane uz vid koji su središnji za misiju OpenCV-a. Štoviše, ML modul je dovoljno svestran da se može primijeniti na širok spektar problema strojnog učenja izvan samog vida, dodatno povećavajući korisnost i fleksibilnost biblioteke za programere u raznim disciplinama (Kaehler & Bradski, 2017)

OpenCV omogućuje izvođenje raznih operacija na slici poput (Kulhary, 2019):

- Čitanje slike: OpenCV pomaže čitati sliku iz datoteke ili izravno s kamere kako bi bila dostupna za daljnju obradu.
- Poboljšanje slike: Može poboljšati sliku omogućujući korisniku podešavanje svjetline, oštrine ili kontrasta slike. Ovo je korisno za vizualizaciju kvalitete slike.
- Detekcija objekata: Objekti, poput narukvica, satova, uzoraka, lica i sličnog, se također mogu detektirati pomoću OpenCV-a.
- Filtriranje slike: Može mijenjati sliku primjenom raznih filtara kao što su zamućenje ili izoštravanje.
- Crtanje na slici: OpenCV omogućuje crtanje teksta, linija i bilo kojih oblika na slikama.
- Spremanje promijenjenih slika: Nakon obrade, može spremiti slike koje su modificirane za buduću analizu.

4.1.3. Kompatibilnost i performanse

Svestranost OpenCV-a dodatno je naglašena njegovom kompatibilnošću s više programskih jezika, uključujući C++, Python, Javu i MATLAB, te podrškom za različite operativne sustave poput Windowsa, Linuxa, Androida i macOS-a. Njegove mogućnosti obrade u stvarnom vremenu poboljšane su korištenjem MMX i SSE instrukcija, a tekući razvoj CUDA i OpenCL sučelja obećava još veće performanse u budućnosti (*About*, bez dat.)

4.2. Scikit-Image

Scikit-image je napredna Python biblioteka prilagođena za obradu slika, besprijekorno integrirana s širim ekosustavom znanstvenog računalstva. Slobodno je dostupna i bez

ograničenja, distribuirana pod BSD licencom, te predstavlja vrijedan alat za razne primjene, od akademskih istraživanja do industrijske upotrebe (van der Walt i ostali, 2014).

4.2.1. Podržani algoritmi

Scikit-image je poznat po svom raznolikom rasponu algoritama dizajniranih za rješavanje različitih zadataka obrade slika. U to spadaju:

- **Segmentacija slika**: Ova značajka omogućuje podjelu slike na više segmenata ili regija. Posebno je korisna za identificiranje različitih objekata ili granica unutar slike, olakšavajući detaljnu analizu i obradu.
- **Geometrijske transformacije**: Scikit-image pruža funkcije za operacije poput skaliranja, rotacije i deformacije slika. Ove transformacije su ključne za prilagodbu prostorskih karakteristika slika kako bi odgovarale specifičnim analitičkim potrebama.
- **Manipulacija prostorom boja**: Biblioteka uključuje alate za pretvaranje slika između različitih prostora boja, kao što je iz RGB u sivu skalu. Ova sposobnost je ključna za razne primjene gdje su potrebne različite reprezentacije boja za analizu ili vizualizaciju.
- **Filtriranje slika**: Metode za filtriranje slika su integralne za poboljšanje ili suzbijanje određenih značajki. Tehnike poput smanjenja šuma i detekcije rubova spadaju u ovu kategoriju, pomažući u poboljšanju kvalitete slike i isticanju važnih detalja.
- **Morfologija**: Morfološke operacije obrađuju slike na temelju njihovih oblika, što je korisno za zadatke poput detekcije objekata i poboljšanja slike. Ove operacije manipuliraju strukturom elemenata slike kako bi postigle željene rezultate.
- **Detekcija značajki**: Scikit-image uključuje algoritme za detekciju i opisivanje značajki unutar slika, kao što su kutovi, rubovi i mrlje. Ove značajke su temeljne za zadatke koji uključuju prepoznavanje i analizu slika (van der Walt i ostali, 2014), (Singh, 2019).

4.2.2. Primjena i utjecaj

Svestranost scikit-imagea čini ga primjenjivim u raznim područjima (van der Walt i ostali, 2014):

- **Medicinska slika**: U medicinskom području, scikit-image koristi se za analizu medicinskih skeniranja, uključujući MRI i CT slike, olakšavajući dijagnozu i istraživanje.
- **Astronomija**: Biblioteka pomaže astronomima u obradi slika nebeskih objekata, pomažući u detekciji i analizi astronomskih fenomena.
- **Strojno učenje**: Scikit-image igra ključnu ulogu u pripremi podataka o slikama za modele strojnog učenja. Podržava zadatke poput ekstrakcije značajki i povećanja podataka, što je ključno za treniranje točnih i robusnih modela.

- **Industrijska inspekcija**: U proizvodnom sektoru, scikit-image se koristi za kontrolu kvalitete analizom slika proizvoda kako bi se otkrili nedostaci, osiguravajući visoke standarde u proizvodnji.

5. Izrada sustava za detekciju odgovora na ispitu

Projekt se bavi razvojem sustava za automatsko ocjenjivanje ispita koristeći tehnologiju optičkog prepoznavanja oznaka (OMR - Optical Mark Recognition). Cilj je stvoriti alat koji omogućava preciznu, brzu i automatiziranu analizu ispita temeljenih na označenim odgovorima. Projektnim rješenjem cilj je eliminirati manualno ocjenjivanje, smanjiti mogućnost ljudske pogreške i ubrzati proces obrade ispita, čime se značajno povećava efikasnost i preciznost u obrazovnim procesima.

Osnovni funkcionalni zahtjevi:

• **FR1: Učitavanje slike ispita** Sustav mora omogućiti učitavanje slike ispita u različitim formatima (npr. JPEG, PNG) za daljnju obradu.
• **FR2: Unos odgovora** Sustav mora omogućiti unos točnog ključa odgovora koji će se uspoređivati sa dotektiranim odgovorima
• **FR3: Detekcija i prepoznavanje označenih odgovora** Sustav mora pronaći odgovore na testu i prepoznati one koji su zabilježeni..
• **FR4: Izračun ocjene** Na temelju broja točnih odgovora, sustav mora izračunati konačnu ocjenu.
• **FR5: Prikaz rezultata** Sustav mora prikazati ocjenu i postotak točnih odgovora, kao i vizualno označiti ispravne i pogrešne odgovore na slici ispita.

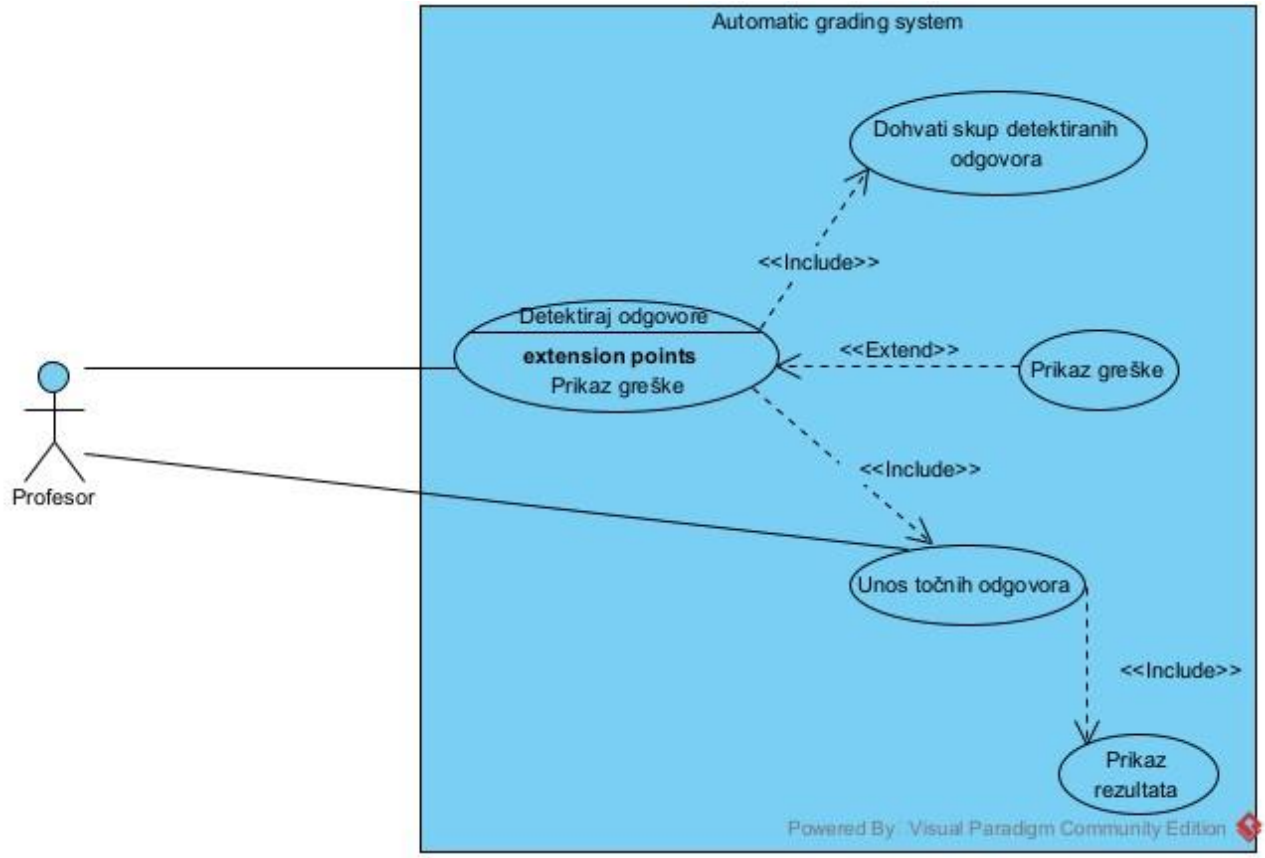
Osnovni nefunkcionalni zahtjevi:

• **NFR1: Točnost** Sustav mora osigurati na 10 testova da 8 bude točno u prepoznavanju i ocjenjivanju odgovora, čak i kada su odgovori blago izvan zadanih granica.
• **NFR2: Efikasnost** Sustav mora biti sposoban obraditi i ocijeniti ispit unutar 3 sekunde.

- **NFR3: Jednostavnost korištenja**
  Sustav mora imati intuitivan proces rada koji ne zahtijeva napredna tehnička znanja od korisnika.

5.1. Slučajevi korištenja

Dijagram slučajeva korištenja pruža pregled na visokoj razini o interakciji različitih aktera sa sustavom. Pomaže u razjašnjavanju funkcionalnosti koje sustav nudi i uloga različitih korisnika. U našem slučaju, primarni akter je Profesor koji inicira i nadzire cijeli proces ocjenjivanja ispita.



Slika 13: Use-case diagram (Izvor: Vlastita izrada)

5.2. Use Case: Ocijeni ispit

1. Primarni Aktor: Profesor

Profesor je vanjski aktor koji inicira proces ocjenjivanja ispita.

2. Glavni Scenarij: Detekcija odgovora

Profesor pokreće aktivnost „Detektiraj odgovore" unutar sistema. Ova aktivnost ima nekoliko povezanih slučajeva korištenja koji osiguravaju ispravan tijek rada.

3. Include Use Case: Unos točnih odgovora

Prije nego što se ispit može ocijeniti, profesor mora unijeti točne odgovore putem aktivnosti „Unos točnih odgovora". Ovo je ključni korak za validno ocjenjivanje jer sistem koristi ove unose kao referentne tačke za ocjenjivanje odgovora učenika.

4. Include Use Case: Dohvati skup detektiranih odgovora

Sistem automatski pokreće aktivnost dohvaćanja „". Ova aktivnost prikazuje profesoru detaljne rezultate, uključujući postotak točnih odgovora i konačnu ocjenu.

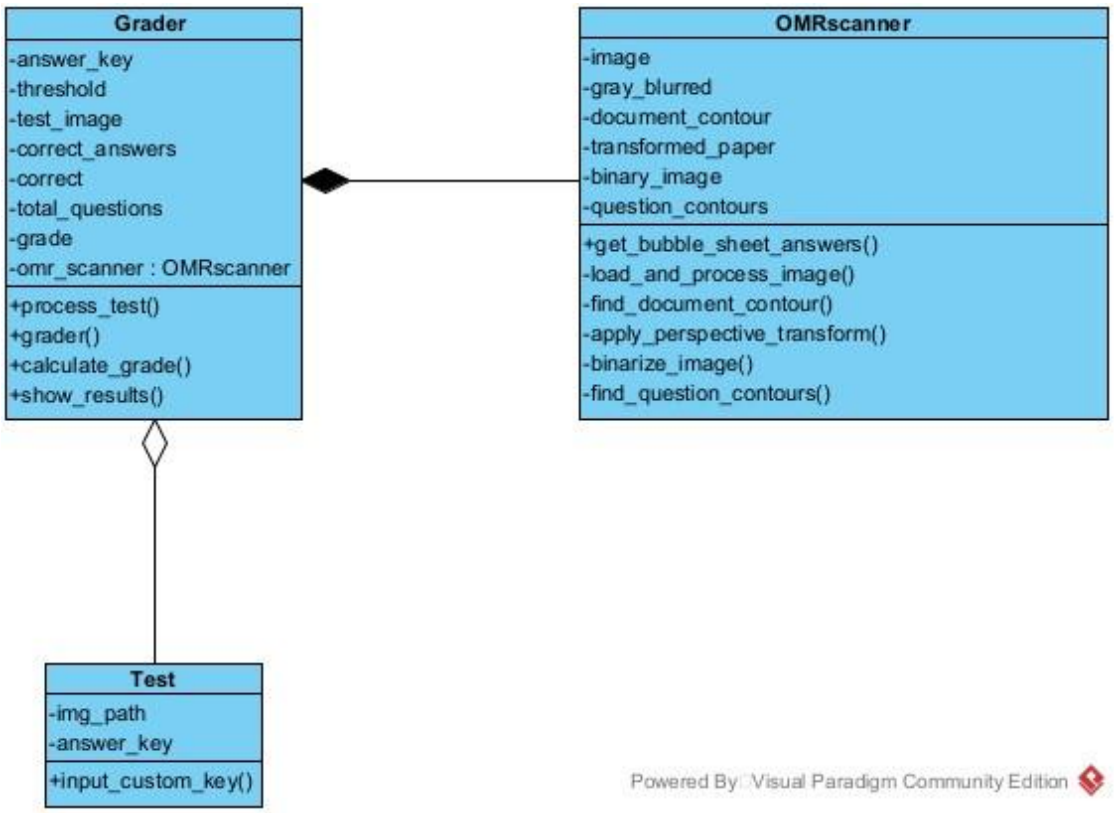
5. Extend Use Case: Prikaz greške

U slučaju da se tokom ocjenjivanja pojavi problem (npr. neispravan unos točnih odgovora ili problem sa skeniranjem slike), aktivnost „Detektiraj odgovore" proširen je slučejm korištenja „Prikaz greške". Ovaj slučaj korištenja pokreće se samo u izuzetnim slučajevima i obavještava profesora o problemu.

Tok aktivnosti:

1. **Pokretanje procesa:** Profesor inicira proces detektiranja odgovora ispita putem slučaja korištenja „Detektiraj odgovore".
2. **Unos točnih odgovora:** Ako točni odgovori nisu prethodno uneseni, sistem zahtijeva da profesor unese ili potvrdi točne odgovore.
3. **Dohvati skup detektiranih odgovora:** Sistem na slici detektira odgovore te ih vraća.
4. **Prikaz rezultata:** Nakon ocjenjivanja sistem prikazuje rezultate profesoru na temelju detektiranih odgovora.
5. **Prikaz greške:** Ako se dogodi problem tokom ocjenjivanja, sistem pokreće prikaz greške.

5.3. Klase podataka



Slika 14: Dijagram klasa (Izvor: Vlastita izrada)

U sustavu za prepoznavanje odgovora na ispitu identificirane su tri ključne klase: Test, Grader i OMRScanner. Ove klase zajedno omogućuju procesiranje i ocjenjivanje testova na temelju skeniranih ispitnih obrazaca. Ovdje ćemo detaljno opisati strukturu i odnose između ovih klasa kako bi se jasno razumjele njihove uloge i interakcije.

5.3.1. Klasa Test

Test je osnovna klasa koja predstavlja ispitni obrazac i odgovore na testu. Ona ima dva glavna atributa:

- image_path: String koji označava putanju do slike ispitnog obrasca.
- answer_key: Rječnik (dict) koji sadrži ključ točnih odgovora za pitanja na testu.

Konstruktor ove klase prima putanju do slike (image_path) i opcionalno ključ odgovora (answer_key). Ako ključ odgovora nije proslijeđen, Test omogućava korisniku unos prilagođenog ključa odgovora putem metode input_custom_answer_key().

5.3.2. Veza s klasom Grader

Test je povezan s klasom Grader putem agregacije. Grader koristi Test objekt za dobivanje slike ispitnog obrasca i ključa odgovora. Agregacija ukazuje na to da Test objekt može postojati neovisno o Grader objektu.

5.3.3. Klasa Grader

Grader je odgovorna klasa za proces ocjenjivanja testa. Ona koristi podatke iz Test objekta za pokretanje procesa prepoznavanja i analize odgovora. Ključni atributi ove klase uključuju:

- test_image: Slika ispitnog obrasca, koja se učitava iz Test objekta.
- answer_key: Ključ točnih odgovora preuzet iz Test objekta.
- omr_scanner: Objekt tipa OMRScanner koji se koristi za procesiranje slike.

Metode unutar Grader klase uključuju procesiranje testa (process_test()), ocjenjivanje odgovora (grader()), izračunavanje ocjene (calculate_grade()), te prikaz rezultata (show_results()).

5.3.4. Veza s klasom Test:

Grader koristi Test objekt za pristup podacima potrebnim za ocjenjivanje. Ovaj odnos je modeliran agregacijom, jer Grader koristi Test, ali ne upravlja njegovim životnim ciklusom.

5.3.5. Veza s klasom OMRScanner:

Grader stvara i upravlja objektom OMRScanner, što predstavlja kompoziciju. Kada se Grader objekt stvori, on inicijalizira OMRScanner kako bi procesirao sliku testa. Ako se Grader objekt uništi, OMRScanner objekt također prestaje postojati jer je životni ciklus OMRScanner objekta čvrsto vezan uz životni ciklus Grader objekta.

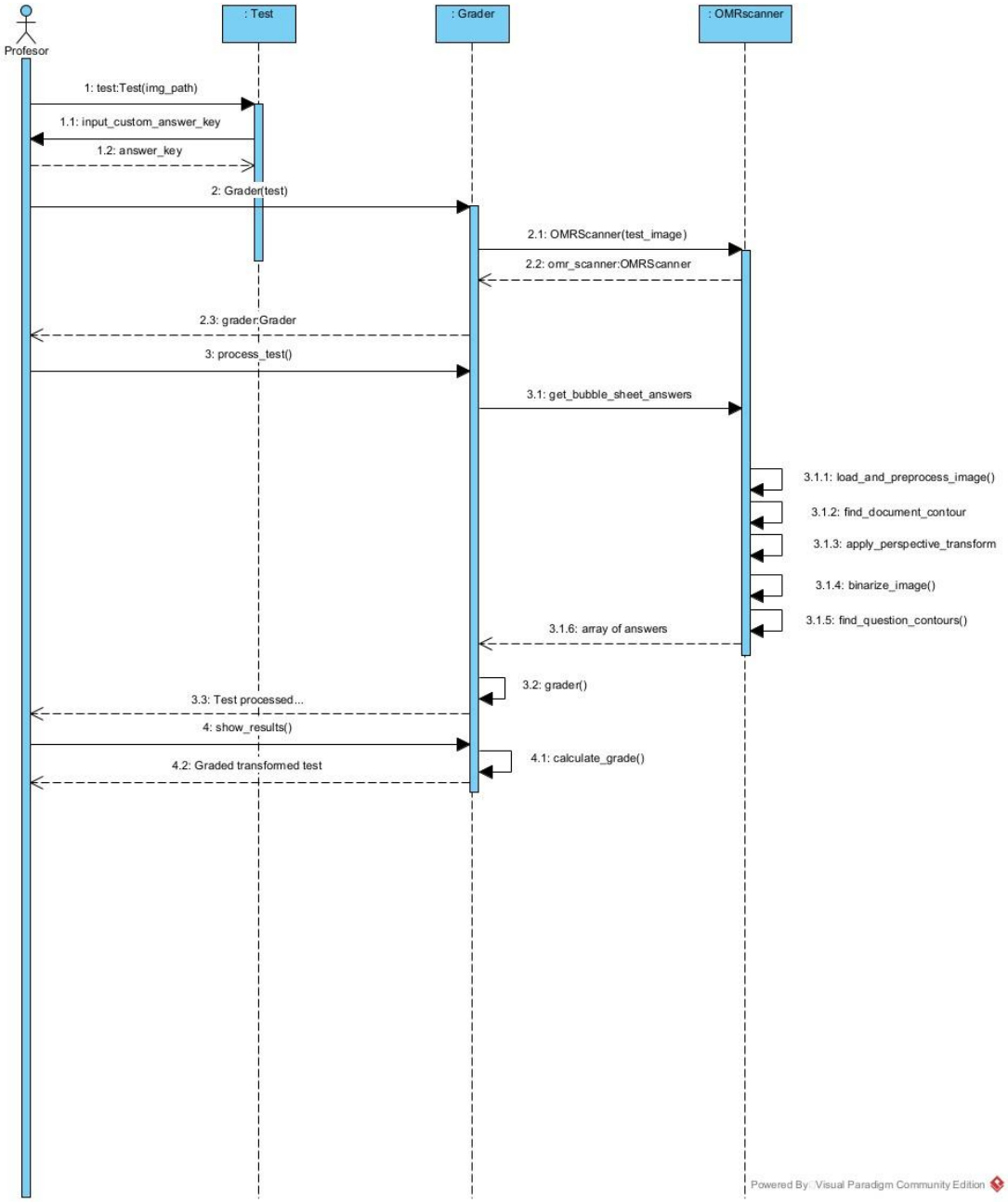
5.3.6. Klasa OMRScanner

OMRScanner je klasa koja pruža funkcionalnosti za procesiranje slike, uključujući pretvorbu slike u sivu skalu, pronalaženje rubova, primjenu perspektivne transformacije te binarizaciju slike. Osim toga, OMRScanner pronalazi konture odgovora na skeniranom testu. Glavni atributi uključuju:

- image: Ulazna slika ispitnog obrasca.
- gray_blurred: Slika nakon konverzije u sivu skalu i primjene zamućenja.
- document_contour: Kontura koja odgovara rubovima dokumenta.

- transformed_paper: Transformirana slika ispitnog obrasca.
- binary_image: Binarizirana slika korištena za prepoznavanje odgovora.
- question_contours: Popis kontura za pojedinačna pitanja.
- Veza s klasom Grader: OMRScanner je unutar klase Grader u obliku kompozicije. Grader stvara OMRScanner objekt i kontrolira njegov životni ciklus. OMRScanner ne postoji neovisno jer je potpuno upravljan od strane Grader objekta.

5.4. Slijed programa



Slika 15: Dijagram slijeda (Izvor: Vlastita izrada)

5.4.1. Inicijalna komunikacija

Na početku dijagrama slijeda, korisnik, koji predstavlja profesora ili osobu koja koristi sistem, pokreće program i inicijalizira proces ocjenjivanja testa. Ova akcija rezultira stvaranjem objekata Test i Grader, koji su ključni za cijeli proces.

- Instanciranje Test objekta: Korisnik stvara instancu klase Test s putanjom do slike testa. Ako nije naveden ključ odgovora, korisnik ima opciju unosa točnih odgovora za svako pitanje. Ova fleksibilnost omogućava sistemu da se prilagodi različitim testovima i situacijama.
- Instanciranje Grader objekta: Nakon stvaranja Test objekta, korisnik instancira Grader objekt, koji se veže za Test objekt. Ovaj korak označava početak stvarnog procesa ocjenjivanja testa. Grader također stvara instancu OMRScanner objekta, koji je odgovoran za tehničke zadatke obrade slike.

5.4.2. Procesiranje Testa: Grader i OMRScanner

Nakon što su Test i Grader objekti kreirani, korisnik pokreće proces ocjenjivanja pozivom metode process_test() na Grader objektu. Ovaj korak uključuje niz aktivnosti koje omogućuju sistemu da analizira test.

- Instanciranje OMRScanner objekta: Unutar metode process_test(), Grader instancira OMRScanner objekt, prosljeđujući mu sliku testa. Ovaj objekt je ključan za obradu slike i prepoznavanje pitanja i odgovora.
- Obrada slike: Grader poziva metodu get_bubble_sheet_answers OMRScanner objekta koja sekvencijalno poziva sljedeće metode:
    1. load_and_preprocess_image(): Učitava sliku i pretvara je u sivu skalu, primjenjujući Gaussian Blur za smanjenje šuma.
    2. find_document_contour(): Pronalaženje rubova dokumenta koristeći Canny algoritam, identificirajući konturu koja predstavlja rub testa.
    3. apply_perspective_transform(): Primjena perspektivne transformacije na dokument kako bi se slika „ispravila" za daljnju obradu.
    4. binarize_image(): Konverzija slike u binarnu (crno-bijelo) koristeći Otsuovu metodu, što omogućava lakše prepoznavanje ispunjenih odgovora.
    5. find_question_contours(): Pronalaženje kontura pitanja na binariziranoj slici kako bi se mogli detektirati označeni odgovori.

Ove metode zajedno omogućavaju OMRScanner objektu da pripremi sve potrebne podatke za ocjenjivanje testa ili ti ga da u sebe spremi pronađenu listu odgovora na temelju koje grader poslje može provrijeti koji su odgovori zabilježeni te dati ocijenu.

5.4.3. Ocjenjivanje Testa: Grader i OMRScanner

Nakon što je procesiranje slike završeno, Grader objekt započinje ocjenjivanje testa. Ključni dio ovog procesa je usporedba otkrivenih odgovora s ključem odgovora koji je korisnik prethodno unio.

- Ocjenjivanje odgovora (grader()): Grader koristi podatke prikupljene od strane OMRScanner objekta kako bi analizirao koji su odgovori označeni i usporedio ih s točnim odgovorima. Proces prolazi kroz svako pitanje, identificira označene odgovore te ih uspoređuje s ključem odgovora.
- Bilježenje rezultata: Za svako pitanje, ako je odgovor točan, Grader bilježi to kao ispravan odgovor. Na kraju Grader izračunava ukupan broj točnih odgovora.

5.4.4. Prikazivanje Rezultata: Grader i Korisnik

Nakon što je test ocijenjen, korisnik može vidjeti rezultate. Ovaj korak uključuje izračunavanje konačne ocjene i prikaz rezultata na ekranu.

- Izračunavanje Ocjene (calculate_grade()): Grader poziva metodu calculate_grade() kako bi izračunao postotak točnih odgovora i odredio ocjenu na temelju tog postotka.
- Prikaz rezultata (show_results()): Nakon izračuna ocjene, Grader prikazuje postotak točnih odgovora i konačnu ocjenu korisniku. Rezultati se također vizualiziraju na originalnoj slici testa, gdje se točni i netočni odgovori označavaju različitim bojama.

5.4.5. Završetak Procesa

Nakon što su rezultati prikazani, dijagram slijeda završava. Korisnik dobiva sve potrebne informacije o testu, uključujući vizualni prikaz odgovora, te može zatvoriti program.

5.5. Implementacija

Za razvoj softverskog rješenja za automatsku detekciju označenih odgovora na pisanom ispitu korištene su sljedeće metode i tehnike:

- **Računalni vid**: Primijenjene su tehnike računalnog vida za analizu slika ispitnih listova i prepoznavanje označenih odgovora.
- **Procesiranje slike**: Korištene su tehnike binarizacije, detekcije kontura i prepoznavanja uzoraka za identifikaciju i analizu označenih odgovora na ispitnim listovima.
- **Canny**: Detekcija rubova za oktrivanje rubova objekata.

- **Gaussovo zamućenje**: filtar u obradi slika koji se koristi za smanjenje šuma i detalja.
- **Binarizacija**: Primijenjena je Otsuova metoda binarizacije kako bi se siva slika pretvorila u binarnu sliku s ciljem jasnog razdvajanja pozadine od označenih odgovora.
- **Detekcija kontura**: Korištena je funkcija cv2.findContours iz OpenCV biblioteke za pronalaženje kontura u binariziranoj slici.
- **Sortiranje kontura**: Konture su sortirane metodom „top-to-bottom" i „left-to-right" kako bi se ispravno identificirale grupe odgovora na ispitnim pitanjima.

Za implementaciju softverskog rješenja korišteni su sljedeći programski alati i aplikacije:

- **PyCharm IDE**: Integrirano razvojno okruženje (IDE) koje je korišteno za pisanje, testiranje i debugiranje Python koda.
- **Python**: Programski jezik koji je korišten za razvoj algoritama i logike aplikacije.
- **OpenCV:** Biblioteka za računalni vid koja pruža funkcionalnosti za procesiranje slike, detekciju kontura i analizu uzoraka.
- **imutils**: Pomoćna biblioteka koja olakšava rad s OpenCV funkcijama.
- **Scipy**: Fundamentalni algoritmi za znanstveno računalstvo u Pythonu.
- **NumPy**: Biblioteka za numeričke operacije koja je korištena za rad s nizovima i matricama podataka.

5.5.1. Detaljna analiza klase Test

Klasa Test je temeljna komponenta programa koja predstavlja skenirani test i pohranjuje ključ odgovora za ocjenjivanje. Ova klasa omogućuje učitavanje slike testa i pohranu ključnih informacija potrebnih za ocjenjivanje, uključujući ispravan niz odgovora na pitanja. Klasa također omogućuje korisniku unos prilagođenog ključa odgovora u slučaju da nije unaprijed definiran.

5.5.1.1. Inicijalizacija klase (__init__ metoda)

```
1. def __init__(self, image_path, answer_key=None):
2.     self.image_path = image_path
3.     if answer_key is None:
4.         self.answer_key = self.input_custom_answer_key()
5.     else:
6.         self.answer_key = answer_key
7.
```

Opis: Konstruktor klase Test prima dva argumenta: image_path, koji predstavlja putanju do slike testa, i answer_key, koji predstavlja ključ odgovora. Ako ključ odgovora nije dostavljen prilikom stvaranja objekta, konstruktor poziva metodu input_custom_answer_key() kako bi korisniku omogućio unos vlastitog ključa. U suprotnom se koristi dostavljeni ključ odgovora.

Važnost: Ova metoda postavlja osnovne atribute objekta Test, uključujući putanju do slike testa (image_path) i ključ odgovora (answer_key). Ovi atributi su ključni za daljnje procese obrade i ocjenjivanja testa jer omogućuju programu da zna koja slika predstavlja test i koji točni odgovori su očekivani za ocjenjivanje.

5.5.1.2. Unos prilagođenog ključa odgovora (input_custom_answer_key metoda)

```
1. def input_custom_answer_key(self):
2.     print("Unesite broj pitanja:")
3.     num_questions = int(input())
4.     answer_key = {}
5.     for i in range(num_questions):
6.         print(f"Unesite točne odgovore za pitanje {i + 1} (npr. 0, 1, 2 za više odgovora):")
7.         correct_answers = list(map(int, input().split(',')))
8.         answer_key[i] = correct_answers
9.     return answer_key
10.
```

Opis: Metoda input_custom_answer_key() omogućuje korisniku unos ključa odgovora za test. Metoda započinje unosom broja pitanja na testu, a zatim iterira kroz svako pitanje, omogućujući korisniku da unese točne odgovore za svako pitanje. Točni odgovori pohranjuju se u rječniku gdje je ključ broj pitanja, a vrijednost je lista točnih odgovora.

Važnost: Ova metoda omogućuje programu fleksibilnost pri radu s različitim testovima jer korisnik može unijeti prilagođeni ključ odgovora. To je korisno u slučajevima kada unaprijed definirani ključ odgovora nije dostupan, omogućujući ocjenjivanje testova u stvarnom vremenu s dinamičkim unosom podataka.

5.5.2. Detaljna analiza klase OMRScanner

Klasa OMRScanner je ključna komponenta u procesu automatskog ocjenjivanja testova. Ova klasa omogućuje obradu i analizu skenirane slike testa, uključujući detekciju rubova dokumenta, binarizaciju slike te identifikaciju kontura pitanja. Svaka metoda unutar ove klase igra specifičnu ulogu u procesu pripreme slike za ocjenjivanje.

5.5.2.1. Inicijalizacija klase (__init__ metoda)

```
1. def __init__(self, test_image):
```

```
2.      self.image = test_image
3.      self.gray_blurred = None
4.      self.document_contour = None
5.      self.transformed_paper = None
6.      self.binary_image = None
7.      self.question_contours = None
```

Opis: Konstruktor klase OMRScanner prima jedan argument, test_image, koji predstavlja sliku skeniranog testa. Unutar metode se inicijaliziraju atributi instance, uključujući izvorni test (image), zatim privremene podatke poput zamagljene sive slike (gray_blurred), konture dokumenta (document_contour), perspektivno transformiranu sliku (transformed_paper), binariziranu sliku (binary_image), i konture pitanja (question_contours). Svi ovi atributi su inicijalno postavljeni na None.

Važnost: Ova metoda postavlja osnovne atribute objekta OMRScanner koji će se koristiti u drugim metodama za pohranu i manipulaciju podacima tijekom obrade slike. Ovi atributi omogućuju pohranjivanje i rad s različitim fazama obrade slike, čime se osigurava dosljedan tok obrade i pripreme podataka za ocjenjivanje.

5.5.2.2. Pokretanje spremanja liste pitanja (get_bubble_sheet_anwers)

```
1.   def get_bubble_sheet_answers(self):
2.         self.load_and_preprocess_image()
3.         self.find_document_contour()
4.         self.apply_perspective_transform()
5.         self.binarize_image()
6.         self.find_question_contours()
```

Opis: Metoda get_bubble_sheet_answers je ključna funkcija u procesu obrade skeniranog testa pomoću klase OMRScanner. Ona orkestrira cijeli postupak analiziranja slike kako bi se dobili odgovori sa skeniranog ispita. Pozivom ove metode, sustav prolazi kroz nekoliko koraka obrade slike

Važnost: Metoda get_bubble_sheet_answers kombinira niz koraka u jedno integrirano rješenje za prepoznavanje i analizu odgovora na skeniranom testu. Time se osigurava da se sve potrebne faze obrade slike izvrše u pravilnom redoslijedu, što omogućuje preciznu i efikasnu ekstrakciju odgovora iz slike ispita. Ova metoda je osnova za daljnje ocjenjivanje ispita, čime se osigurava da sustav može pouzdano prepoznati i analizirati odgovore na temelju skenirane slike.

5.5.2.3. Učitavanje i pretvorba slike (load_and_preprocess_image metoda)

```
1. def load_and_preprocess_image(self):
2.     """Učitavanje slike i pretvorba u blure"""
```

```
3.      gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
4.      self.gray_blurred = cv2.GaussianBlur(gray, (5, 5), 0)
```

Opis: Ova metoda prvo pretvara sliku iz kolor prostora u sivi prostor koristeći cv2.cvtColor. Zatim se slika zamućuje (blur) korištenjem GaussianBlur filtra kako bi se smanjila razina šuma i olakšala detekcija rubova u kasnijim koracima.

Važnost: Učitavanje i pretvorba slike u sivi prostor te zamućivanje su ključni koraci za smanjenje nepotrebnih detalja i pripremu slike za kasniju obradu, kao što je detekcija rubova i kontura.

5.5.2.4. Pronalaženje rubova dokumenta (find_document_contour metoda)

```
 1. def find_document_contour(self):
 2.     """Pronalaženje rubova dokumenta"""
 3.     edges = cv2.Canny(self.gray_blurred, 75, 200)
 4.     contours_list = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
 5.     contours_list = imutils.grab_contours(contours_list)
 6.
 7.     if contours_list:
 8.         contours_list = sorted(contours_list, key=cv2.contourArea, reverse=True)
 9.         for contour in contours_list:
10.             peri = cv2.arcLength(contour, True)
11.             approx = cv2.approxPolyDP(contour, 0.02 * peri, True)
12.             if len(approx) == 4:
13.                 self.document_contour = approx
14.                 break
15.
```

Opis: Ova metoda koristi zamagljenu sliku kako bi detektirala rubove dokumenta i pronašla konturu koja odgovara pravokutnom obliku (dokument).

- cv2.Canny: Algoritam za detekciju rubova na slici.
- cv2.findContours: Pronalaženje kontura na slici.
- cv2.approxPolyDP: Pojednostavljuje rubove do točke da ima četiri vrha, što odgovara pravokutniku.

Važnost: Precizna detekcija kontura dokumenta ključna je za ispravno poravnavanje slike tijekom transformacije koja slijedi.

5.5.2.5. Binarizacija slike (binarize_image metoda)

```
1. def binarize_image(self):
2.     """Binarizacija slike pomoću Otsuove metode"""
3.     if self.transformed_paper is not None:
```

```
4.          gray = cv2.cvtColor(self.transformed_paper, cv2.COLOR_BGR2GRAY)
5.          self.binary_image = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV |
cv2.THRESH_OTSU)[1]
6.
```

Opis: Ova metoda pretvara izravnatu sliku u binarnu sliku (crno-bijelo) pomoću Otsuove metode(adaptive thresholding), što omogućava lakše prepoznavanje kontura.

- cv2.threshold: Binarizacija slike gdje su sve nijanse ispod određene vrijednosti pretvorene u crnu, a iznad u bijelu.

Važnost: Binarizacija slike pomaže u jasnoj diferencijaciji između pozadine i ispunjenih odgovora, što je bitno za točnu detekciju odgovora.

5.5.2.6. Pronalaženje kontura pitanja (find_question_contours metoda)

```
1. def find_question_contours(self):
2.     """Pronalaženje kontura pitanja na binariziranoj slici"""
3.     if self.binary_image is not None:
4.         contours_list = cv2.findContours(self.binary_image.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
5.         contours_list = imutils.grab_contours(contours_list)
6.         self.question_contours = []
7.
8.         for contour in contours_list:
9.             (x, y, w, h) = cv2.boundingRect(contour)
10.            aspect_ratio = w / float(h)
11.            if w >= 20 and h >= 20 and 0.9 <= aspect_ratio <= 1.1:
12.                self.question_contours.append(contour)
13.
```

Opis: Ova metoda pronalazi konture potencijalnih odgovora na binariziranoj slici.

- cv2.boundingRect: Izračunava pravokutnik koji potpuno okružuje konturu.
- aspect_ratio: Omjer širine i visine koji pomaže u identifikaciji kontura koje su približno kvadratnog oblika (što odgovara tipičnom obliku zaokruženih odgovora).

Važnost: Identifikacija kontura pitanja omogućava daljnje ocjenjivanje jer ove konture predstavljaju odgovore koje je potrebno usporediti s ključem.

5.5.3. Detaljna analiza klase Grader

Klasa Grader je odgovorna za ocjenjivanje skeniranog testa na temelju prethodno definiranog ključa odgovora. Ova klasa koristi instancu klase OMRScanner za obradu slike testa, detekciju odgovora te konačno ocjenjivanje i prikaz rezultata.

5.5.3.1. Inicijalizacija klase (__init__ metoda)

```
 1. def __init__(self, test, threshold=650):
 2.     self.answer_key = test.answer_key
 3.     self.test_image = cv2.imread(test.image_path)
 4.     self.threshold = threshold
 5.     self.omr_scanner = OMRScanner(self.test_image)
 6.     self.correct_answers = None
 7.     self.correct = 0
 8.     self.total_questions = len(test.answer_key)
 9.     self.grade = None
10.
```

Opis: Konstruktor klase Grader prima objekt klase Test kao argument, zajedno s opcionalnim pragom (threshold) za broj piksela koji definiraju označeni odgovor. Konstruktor inicijalizira ključ odgovora (answer_key), učitava sliku testa (test_image) te kreira instancu klase OMRScanner. Također, postavlja se broj točnih odgovora (correct) na nulu, kao i broj ukupnih pitanja (total_questions). Varijabla grade inicijalno je postavljena na None.

Važnost: Ova metoda postavlja sve ključne atribute potrebne za ocjenjivanje testa. Kreira se instanca klase OMRScanner koja će omogućiti obradu slike testa, dok prag piksela omogućuje prilagodbu algoritma za detekciju odgovora.

5.5.3.2. Procesiranje testa (process_test metoda)

```
1. def process_test(self):
2.     """Procesiranje testa"""
3.     self.omr_scanner.get_bubble_sheet_answers()
8.     self.grader()
9.
```

Opis: Ova metoda poziva niz metoda iz instance klase OMRScanner kako bi obradila sliku testa. Nakon što se slika učita, obrade i pronađu konture pitanja, metoda poziva internu metodu grader() kako bi se test ocijenio.

Važnost: Ova je metoda centralni dio procesa ocjenjivanja jer povezuje sve korake obrade slike u jedan tok, od učitavanja slike do detekcije i ocjenjivanja odgovora.

5.5.3.3. Ocjenjivanje testa (grader metoda)

```
 1. def grader(self):
 2.     if self.omr_scanner.question_contours:
 3.         question_contours = contours.sort_contours(self.omr_scanner.question_contours,
method="top-to-bottom")[0]
 4.     else:
 5.         raise ValueError("Nisu pronađene konture pitanja.")
```

```
 6.
 7.     for (q, i) in enumerate(np.arange(0, len(question_contours), 5)):
 8.         contour_questions = contours.sort_contours(question_contours[i:i + 5])[0]
 9.         selected_answers = []
10.
11.         for (j, contour) in enumerate(contour_questions):
12.             mask = np.zeros(self.omr_scanner.binary_image.shape, dtype="uint8")
13.             cv2.drawContours(mask, [contour], -1, 255, -1)
14.             mask = cv2.bitwise_and(self.omr_scanner.binary_image,
self.omr_scanner.binary_image, mask=mask)
15.             total_pixels = cv2.countNonZero(mask)
16.             if total_pixels > self.threshold:
17.                 selected_answers.append(j)
18.
19.         correct_answers = self.answer_key[q]
20.         color = (0, 0, 255)
21.
22.         if set(selected_answers) == set(correct_answers):
23.             color = (0, 255, 0)
24.             self.correct += 1
25.
26.         for idx in selected_answers:
27.             cv2.drawContours(self.omr_scanner.transformed_paper, [contour_questions[idx]], -
1, color, 3)
28.
```

Opis: Ova metoda predstavlja srž funkcionalnosti Grader klase. Na temelju kontura pitanja prepoznatih na slici pomoću OMRScanner klase, grade metoda uspoređuje označene odgovore s točnim odgovorima iz ključa.

- contours.sort_contours(): Sortira konture pitanja kako bi se osiguralo da su pitanja poredana od vrha prema dnu, što odgovara rasporedu pitanja na testu.
- np.arange(0, len(question_contours), 5): Petlja prolazi kroz skupove od pet kontura koje predstavljaju moguće odgovore na jedno pitanje.
- cv2.drawContours(): Ova funkcija crta konture na maski kako bi izolirala područje odgovora.
- cv2.bitwise_and(): Kombinira binarnu sliku s maskom kako bi se identificiralo područje označeno na testu.
- cv2.countNonZero(mask): Broji bijele piksele unutar maske kako bi se odredilo je li odgovor označen.
- if total_pixels > self.threshold: Ako broj bijelih piksela prelazi prag, taj odgovor se smatra označenim.

- set(selected_answers) == set(correct_answers): Uspoređuje označene odgovore s točnim odgovorima. Ako se poklapaju, povećava se broj točnih odgovora (correct).

Važnost: Ova metoda omogućuje precizno ocjenjivanje skeniranih testova. Koristeći definirani prag i ključ odgovora, grade metoda određuje točnost odgovora na testu, što je ključno za izračunavanje konačne ocjene. Također, vizualno označava točne i netočne odgovore na skeniranoj slici, što može biti korisno za daljnju analizu.

5.5.4. Pokretanje programa: Funkcija main

Funkcija main služi kao ulazna točka za pokretanje programa za automatsko ocjenjivanje testa. U nastavku je kratki opis.

```
1. def main():
2.     ap = argparse.ArgumentParser()
3.     ap.add_argument("-i", "--image", required=False, default="rotirana.png", help="path to
the input image")
4.     args = vars(ap.parse_args())
5.
6.     test = Test(args["image"])
7.     grader = Grader(test)
8.
9.     grader.process_test()
10.    grader.show_results()
11.
```

Opis:

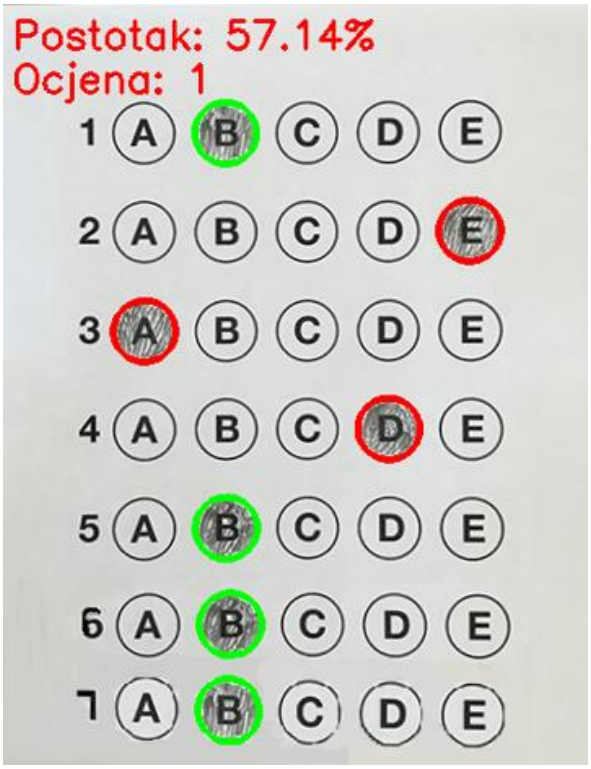
- Argumenti komandne linije: Program koristi biblioteku argparse za prihvaćanje putanje do slike testa kao argumenta komandne linije. Ako putanja nije navedena, koristi se podrazumevana slika „rotirana.png".
- Instanciranje objekata: Na osnovu unesenih argumenata, kreiraju se objekti klase Test i Grader.
- Procesiranje testa: Poziva se metoda process_test iz klase Grader koja obrađuje sliku testa i ocjenjuje odgovore.
- Prikaz rezultata: Na kraju, metoda show_results prikazuje postotak točnih odgovora i konačnu ocjenu.

Ova funkcija povezuje sve komponente sistema, pokreće proces obrade slike i ocjenjivanja testa te na kraju prikazuje rezultate korisniku.

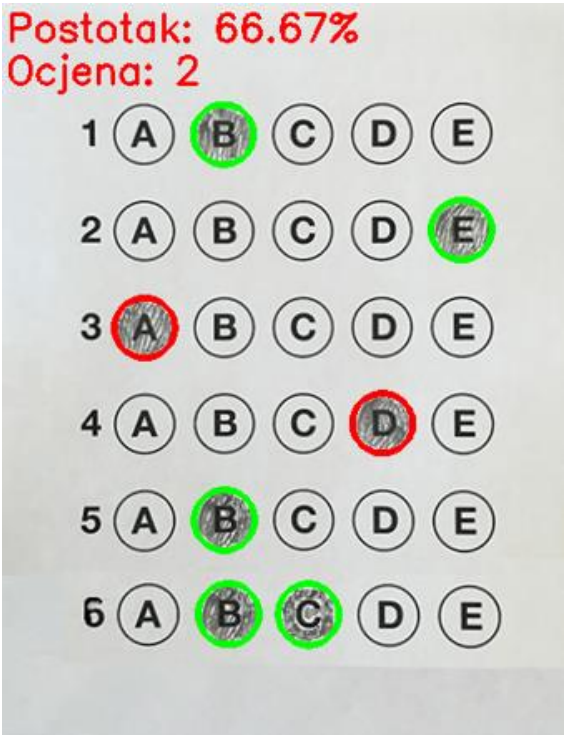
5.6. Analiza Rezultata

Nakon što se program izvrši s određenom putanjom slike i proizvoljno definiranim ključem točnih odgovora, rezultat je slika na kojoj su prikazani točno i netočno označeni odgovori. Točno označeni odgovori obično su prikazani zelenom bojom, dok su netočno označeni odgovori prikazani crvenom bojom. Ova vizualna povratna informacija omogućava korisniku da brzo uoči pogreške i usporedi ih s ključem točnih odgovora.

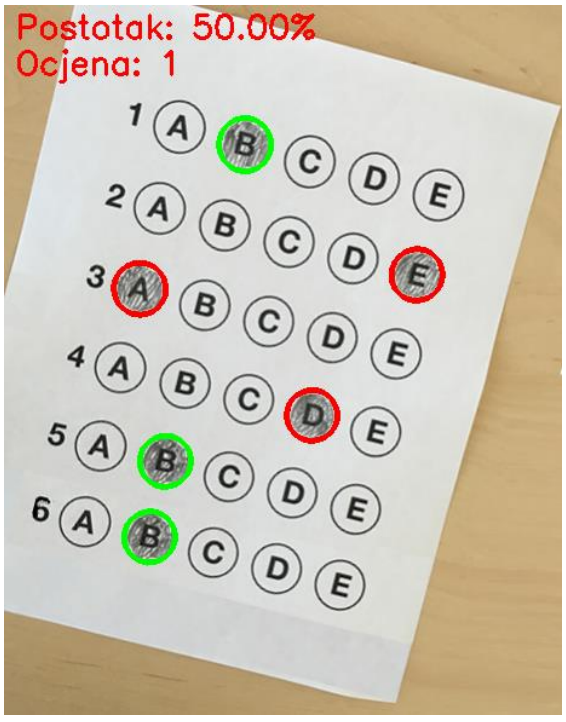
Na kraju analize, program prikazuje postotak točno riješenih pitanja te odgovarajuću ocjenu na temelju tog postotka. Ova informacija se prikazuje direktno na testu, omogućujući korisniku da odmah vidi svoj uspjeh. Na slici rezultata jasno je prikazan postotak točnih odgovora, kao i konačna ocjena koja je izračunata prema standardiziranim kriterijima (npr. 90% ili više za ocjenu 5).



Slika 17: Primjer uspješno obrađenog ispita (Izvor: Vlastita izrada)



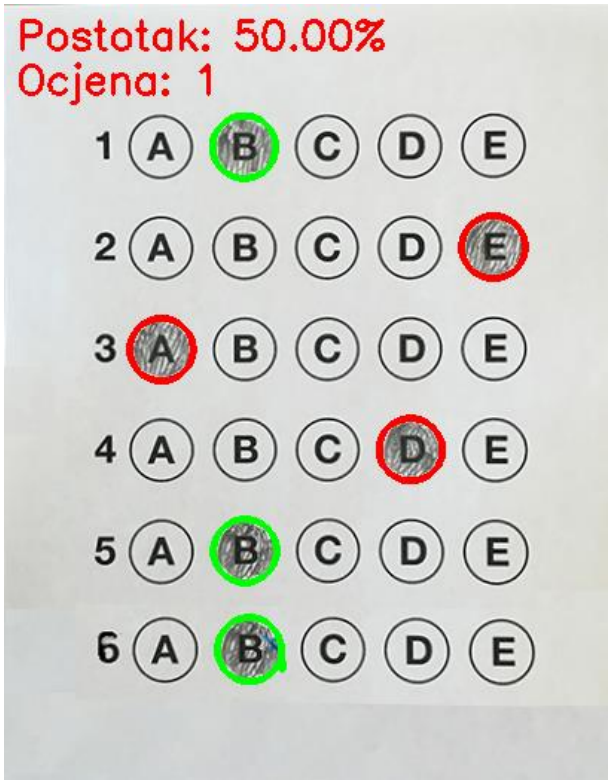
Slika 16: Primjer dvostrukog odgovora (Izvor: Vlastita izrada)



Slika 18: Primjer rotirano slikano ispita (Izvor: Vlastita izrada)

U primjeru slike 18 može se uočiti da na slici nema svih rubova ispita te da transformacija nije uspjela. No program je dalje uspješno ocijenio ispit.

U primjeru slike 19 u 6. pitanju se nalazi mala devijacija gdje zabilježeni odgovor izlazi izvan okvira kruga. No program i dalje uspješno obrađuje program.



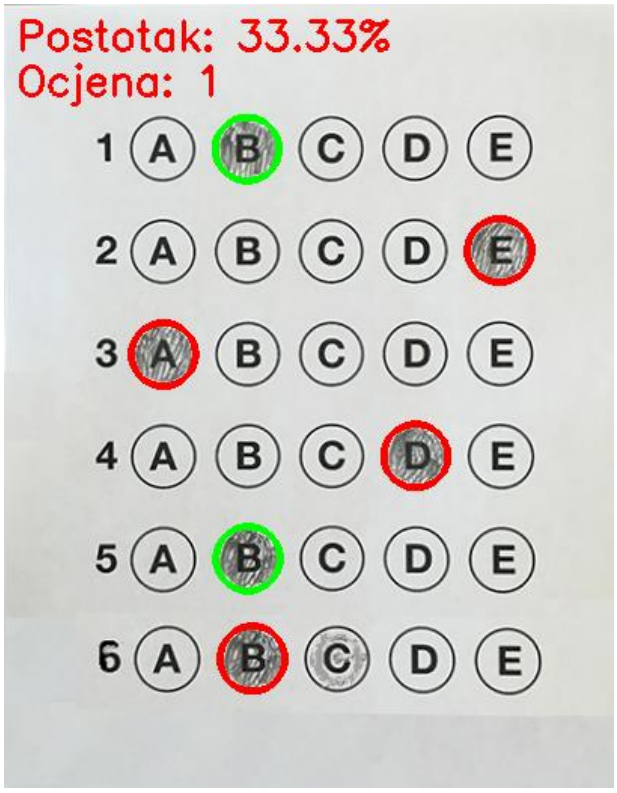
Slika 19: Primjer izlaženja van okvira kruga (Izvor : Vlastita izrada)

5.6.1. Poboljšanja

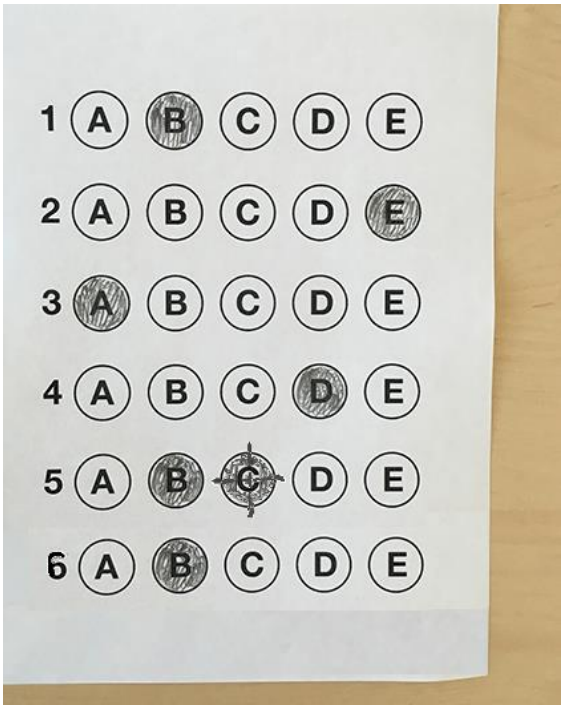
Trenutno program postiže otprilike 90% točnosti pri analizi testova s točno određenim dimenzijama krugova za bilježenje odgovora. Točnost je, međutim, osjetljiva na promjene u dimenzijama tih krugova. Kada se dimenzije testa smanje, a parametri programa nisu prilagođeni tim novim dimenzijama, točnost se može smanjiti na oko 60%.

Na primjeru slike rezultata koja slijedi, korišten je ključ odgovora [1,1,1,1,1,1,1], ali jedan od odgovora nije bio pravilno prepoznat. To ukazuje na potrebu za daljnjom optimizacijom parametara poput dimenzija (if w >= 12 and h >= 12 and 0.8 <= aspect_ratio <= 1.2) i praga za detekciju piksela (threshold=250). Ova nepreciznost naglašava potrebu za dodatnom kalibracijom sustava, posebno kad se koristi na testovima s različitim karakteristikama.

Slika 20: Primjer pogrešno obrađenog testa , točan odgovor za svako pitanje je B, nisu pronađene sve konture (Izvor: Vlastita izrada)



Slika 21: Primjer pogrešno obrađenog ispita - slabo obilježen odgovor (Izvor: Vlastita izrada)



Slika 22: Primjer neuspješno obrađenog ispita - nepronalaženje kontura pitanja (Izvor: Vlastita izrada)

Kako bi se poboljšala točnost i fleksibilnost programa, moguća su sljedeća poboljšanja:

- **Dodavanje GUI-ja** (grafičkog korisničkog sučelja): Ova nadogradnja omogućila bi korisnicima lakšu interakciju s programom, uključujući unos ključa odgovora i odabir slike, bez potrebe za korištenjem komandne linije.
- **Integracija kamere**: Korištenjem kamere, korisnici bi mogli direktno snimiti sliku testa, čime bi se smanjila potreba za prethodnim spremanjem slike na računalo.
- **Primjena strojnog učenja:** Implementacijom algoritama strojnog učenja, program bi mogao postati otporniji na različite dimenzije i formate testova, automatski se prilagođavajući promjenama u veličini krugova ili kontrastu slike. Ovo bi značajno povećalo točnost prepoznavanja i proširilo mogućnosti primjene programa.

Ova poboljšanja mogla bi dodatno unaprijediti funkcionalnost i točnost programa čineći ga korisnijim i prilagodljivijim za različite scenarije primjene te je predmet za razmatranje u daljnoj budućnosti.

5.7. Testiranje rješenja u realnim okolnostima

Prethodna implementacija automatskog prepoznavanja odgovora na OMR (Optical Mark Recognition) testovima bila je dokaz koncepta (proof of concept). Pokazala je potencijal ovakve aplikacije no i svoje nedostatke poput nefleksibilnog tipa ispita i potrebnog ručnog dodatnog poboljšavanja parametara Kako bi se ova tehnologija mogla koristiti u stvarnom svijetu, potrebno je izgraditi stabilan i robustan sustav. Efikasno rješenje uključuje razvoj API-ja koji će omogućiti jednostavnu integraciju i pouzdano prepoznavanje odgovora u stvarnim uvjetima.

5.7.1. Ciljevi

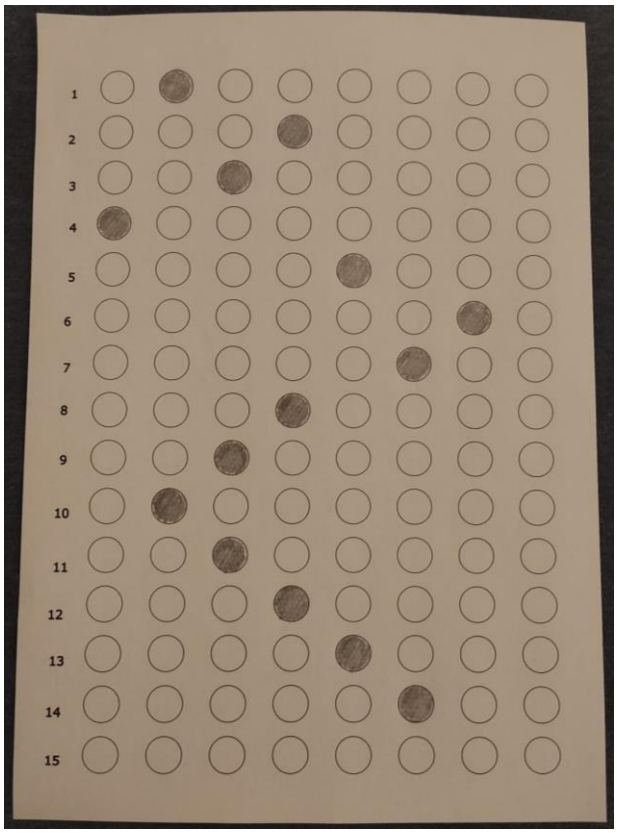
Cilj je razviti API koristeći Flask koji će koristiti postojeće klase OMRScanner i Grader. Ove klase već obavljaju osnovne zadatke skeniranja i „prepoznavanja“ obilježenih odgovora, no potrebna je daljnja optimizacija i proširenje funkcionalnosti kako bi se sustav mogao koristiti u stvarnim uvjetima.

5.7.2. Priprema Testova

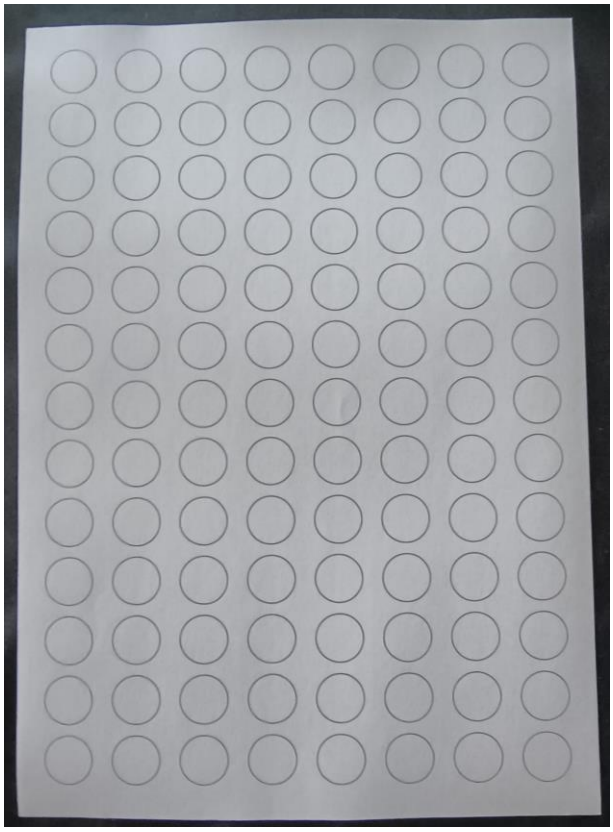
Testove ćemo izraditi koristeći Inkscape, alat za vektorsko crtanje. Pri izradi testova, ključni parametri uključuju:

- **Visina krugova**: Najmanje 18 px.
- Širina prostora između krugova: najmanje 15 px.
- **Radijus polukruga** u testu dimenzija 15x8: Oko 13 px.

Ovi parametri osiguravaju točnu obradu pitanja. Za manje dimenzije je potrebno eksperimentirati



Slika 23 : Primjer označenog ispita 15x8 (Izvor: Vlastita izrada)

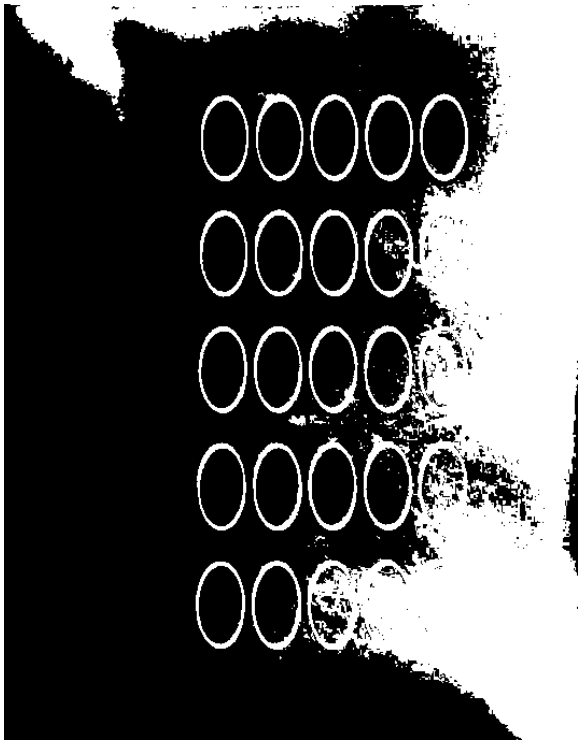


Slika 24: Primjer ispita 13x8 (Izvor: Vlastita izrada)

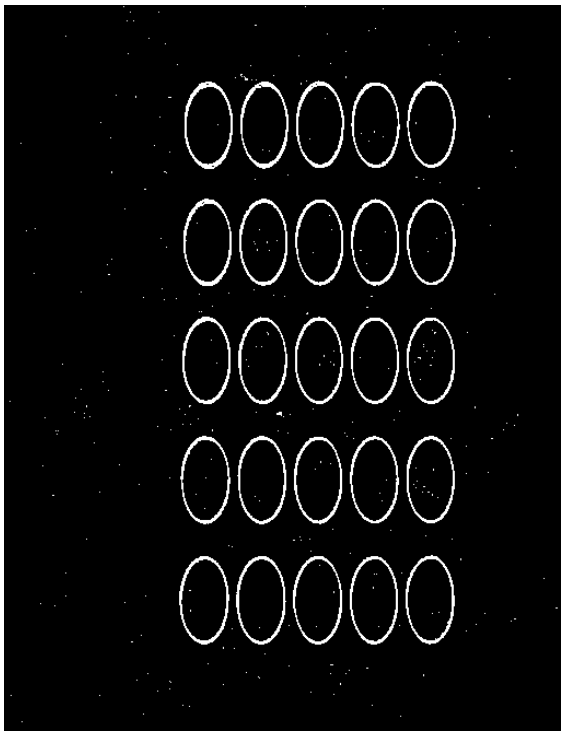
5.7.3. Problemi u Detekciji i Njihova Rješenja

Pri razvoju sustava za stvarnu upotrebu, pojavljivali su se problemi:

1. **Veličina slike:** Slike preuzete sa mobitela su bile različitih dimenzija što bi dovodilo do netočnog funkcioniranja programa
    - **Rješenje:** Korištenje cv2.resize funkcije na konstantnu veličinu za sve slike
2. **Ne prepoznavanje rubova ispita**: U slučajevima gdje je pozadina na kojoj leži test nijansa bijele algoritam ne bi prepoznao rubove ispita
    - **Rješenje**: Slikanje testova na pozadinama jasne razlike među bojama poput školske klupe.
3. **White noise**: Pozadinski šum ili nejasni otisci mogu ometati pravilno prepoznavanje odgovora kod prijenosa slika iz više izvora.
    - **Rješenje**: Korištenje **adaptive thresholdinga** u obradi slike. Ova tehnika omogućuje prilagodbu praga ovisno o lokalnim karakteristikama slike, što rezultira boljom binarizacijom i manje osjetljivošću na varijacije u svjetlosti ili kvaliteti ispisa. Također, preporuča se koristi pozadina koja je jasno različite boje od bijelog ispita, čime se smanjuje mogućnost pojave šuma.



Slika 25: Primjer šuma (Izvor: Vlastita izrada)



Slika 26: Primjer korištenja adaptivnog praga (Izvor: Vlastita izrada)

5.7.4.Implementacija API-ja

Implementacija API-ja u Flasku obuhvaća sljedeće korake:

1. **Postavljanje Flask aplikacije**: Kreiranje osnovnog API-ja koji može primati slike putem POST zahtjeva.

```
from flask import Flask, jsonify, request
from omr_module.omr import Grader
import cv2
import numpy as np

app = Flask(__name__)


@app.route('/getMarkedBubbles, methods=['POST'])
def get_marked_bubbles():
    # Get the image from the POST request
    image_file = request.files.get('image')

    # Convert the image file to a format that OpenCV can use
    image_bytes = np.fromstring(image_file.read(), np.uint8)
    test_image = cv2.imdecode(image_bytes, cv2.IMREAD_COLOR)


    num_rows = int(request.form.get('num_rows'))  # cca. 15 questions
```

```
20.     num_choices = int(request.form.get('num_choices'))  # cca. 8 possible answers per
question
21.
22.
23.     grader = Grader(test_image, num_rows, num_choices)
24.
25.
26.     answers = grader.process_test()
27.
28.
29.     return jsonify({
30.         "answers": answers
31.     })
32.
33.
34. if __name__ == '__main__':
35.     app.run(debug=True)
36.
```

Slanjem POST zahtjeva na rutu „/grade“ koja očekuje sliku pod imenom „image“ te u tijelu polja forme pod ključem „num_rows“ očekuje vrijednost broja pitanja i pod „num_choices“ broj ponuđenih odgovora te time se pokreće funkcija get_marked_bubbles. Najprije se dohvaćena slika pretvara u oblik koji openCV može prepoznati. Nakon toga se kreira grader object kojem se prosljeđuje slika te uneseni broj redaka pitanja i broj mogućih odgovora. Odgovor zahjteva je json odgovora koje se dobiju od grader.process_test() funkcije.

2. **Modul OMR** je ključni dio sustava koji se importira u main.py, gdje se pokreće Flask aplikacija. Ovaj modul je proširen i optimiziran kako bi se osiguralo precizno prepoznavanje odgovora na testovima.

Jedna od glavnih modifikacija u modulu je uvođenje prilagodljive primjene granica (engl. adaptive thresholding) tehnike. Ova metoda omogućuje dinamičku prilagodbu praga za binarizaciju slike, ovisno o lokalnim karakteristikama slike. To osigurava bolju obradu slika s neujednačenim osvjetljenjem ili šumom.

Dodatno, prag za prepoznavanje označenih krugova (engl. bubbla) više nije statičan. Umjesto toga, dinamički se izračunava na temelju srednje vrijednosti svih krugova na testu. Ovo omogućuje precizniju detekciju, čak i kada postoje varijacije u debljini ili intenzitetu oznaka. Klasa OMR ostaje identična osim djela binarizacije slike.

```
1. def binarize_image(self):
         if self.transformed_paper is not None:
           self.binary_image = cv2.adaptiveThreshold(
```

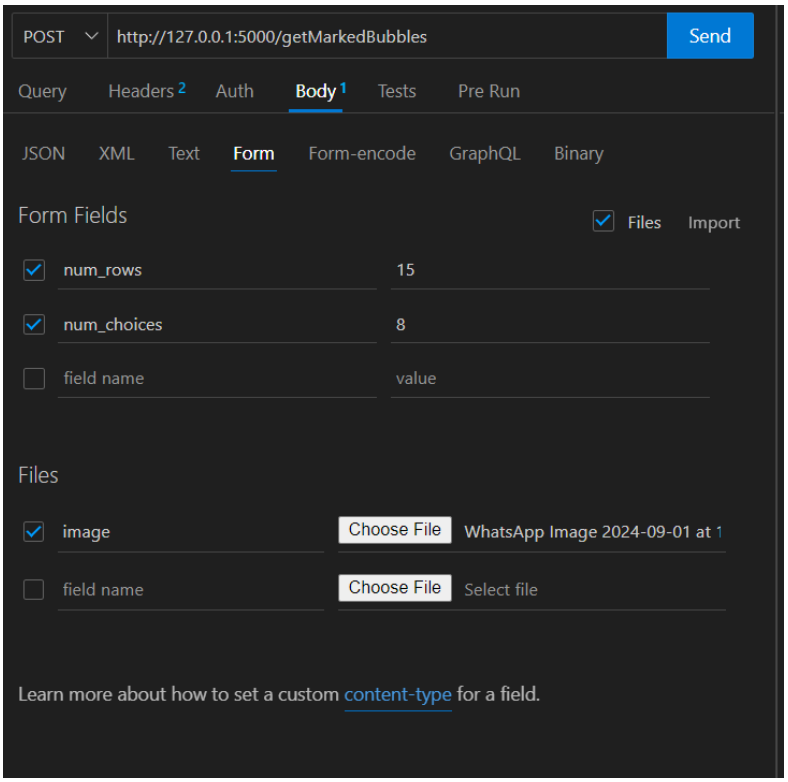
```
            self.transformed_paper,
            255,
            cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
            cv2.THRESH_BINARY_INV,
            11,
            3 )
```

```
def grader(self):
answers = []
        expected_contours = self.num_questions * self.num_choices
        actual_contours = 0

        if self.omr_scanner.question_contours:
            question_contours = contours.sort_contours(self.omr_scanner.question_contours,
method="top-to-bottom")[0]

            for i in range(0, len(question_contours), self.num_choices):
                contour_questions = contours.sort_contours(question_contours[i:i +
self.num_choices])[0]
                selected_answers = []
                pixel_values = []  # Array to store the total pixel values for each contour

                # First, calculate the total pixels for each contour and store them
                for contour in contour_questions:
                    mask = np.zeros(self.omr_scanner.binary_image.shape, dtype="uint8")
                    cv2.drawContours(mask, [contour], -1, 255, -1)
                    mask = cv2.bitwise_and(self.omr_scanner.binary_image,
self.omr_scanner.binary_image, mask=mask)
                    total_pixels = cv2.countNonZero(mask)
                    pixel_values.append(total_pixels)

                if pixel_values:
                    average_pixels = sum(pixel_values) / len(pixel_values)
                    dynamic_threshold = average_pixels

                for j, total_pixels in enumerate(pixel_values):
                    if total_pixels > dynamic_threshold + 50:
                        selected_answers.append(j)
                actual_contours += len(contour_questions)

                question_number = (i // self.num_choices) + 1
                answers.append({
                    "question": question_number,
                    "selected_answers": selected_answers
                })
```

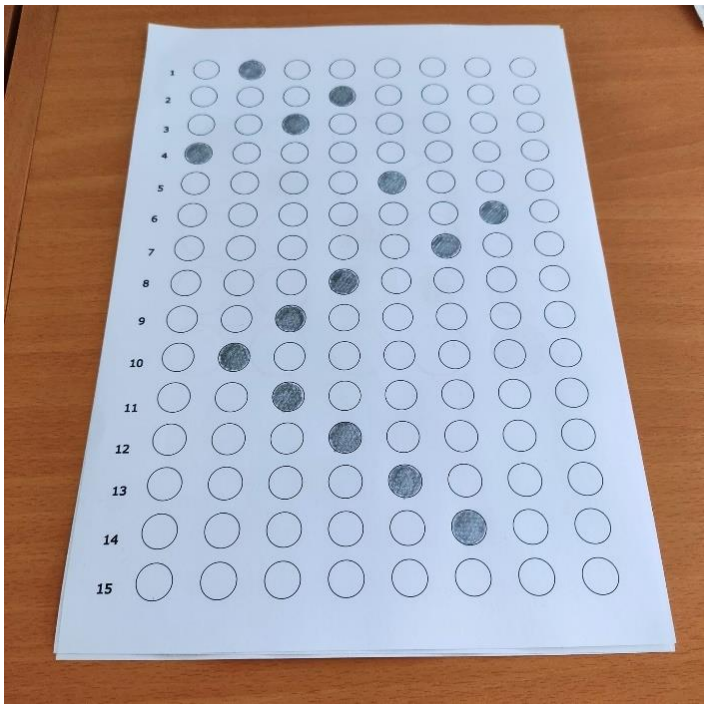
```
36.
37.
38.         if actual_contours == expected_contours:
39.             message = "All questions found."
40.         else:
41.             message = f"Warning: Expected {expected_contours} contours, but found
{actual_contours}."
42.
43.         return {
44.             "answers": answers,
45.             "message": message,
46.             "detected_contours": actual_contours
47.         }
48.
```

5.7.5.Primjer rada

Na početku je potrebno pokrenuti program main.py koji sadrži Flask aplikaciju. Zatim je pomoću odgovarajućeg alata (u ovom slučaju korišten je Thunder Client) potrebno unijeti tražene podatke kao što su prikazani na slici 27. Nakon toga je potrebno odabrati fotografiju ispita, a u ovom je slučaju priložena fotografija sa slike 28.



Slika 27: Primjer podataka zahtjeva (Izvor: Vlastita izrada)



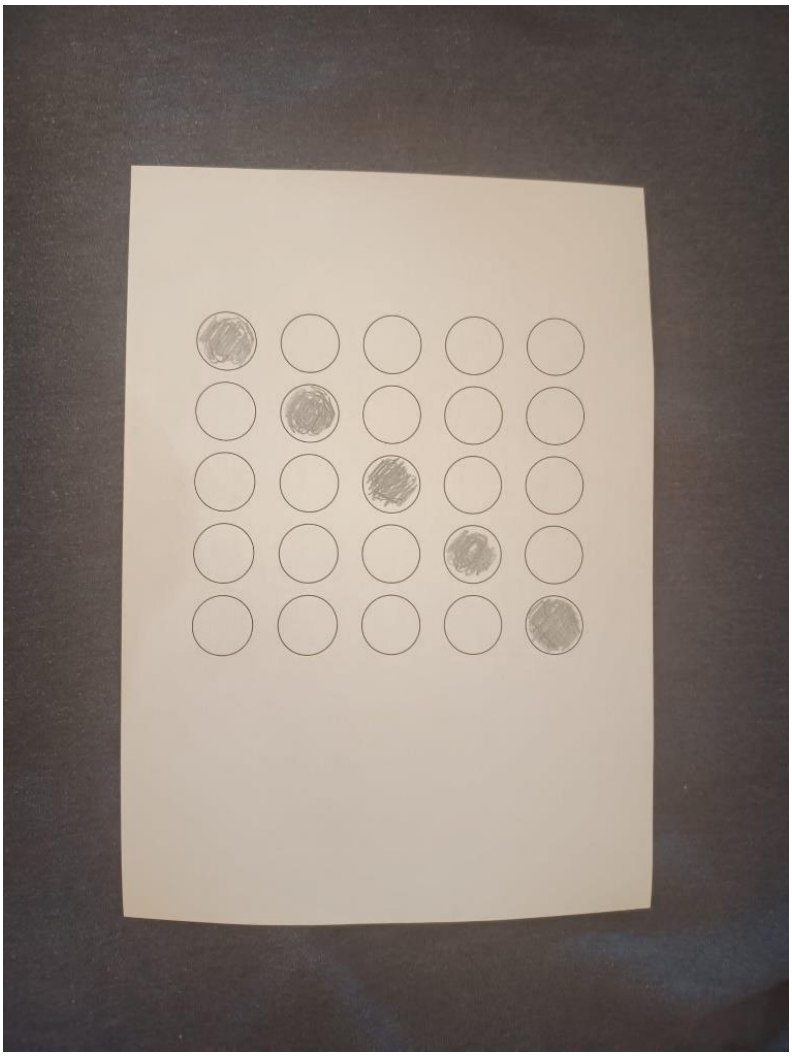
Slika 28: Primjer ispita za API 15x8 (Izvor:Vlastita izrada)

Rezultat koji program vraća kao odgovor prikazan je u nastavku.

```
{"answers":{"answers":
      [{"question":1,"selected_answers":[1]},
      {"question":2,"selected_answers":[3]},
      {"question":3,"selected_answers":[2]},
      {"question":4,"selected_answers":[0]},
      {"question":5,"selected_answers":[4]},
      {"question":6,"selected_answers":[6]},
      {"question":7,"selected_answers":[5]},
      {"question":8,"selected_answers":[3]},
      {"question":9,"selected_answers":[2]},
      {"question":10,"selected_answers":[1]},
      {"question":11,"selected_answers":[2]},
      {"question":12,"selected_answers":[3]},
      {"question":13,"selected_answers":[4]},
      {"question":14,"selected_answers":[5]},
      {"question":15,"selected_answers":[]}],
      "detected_contours":120,"message":"All questions found."}}
```

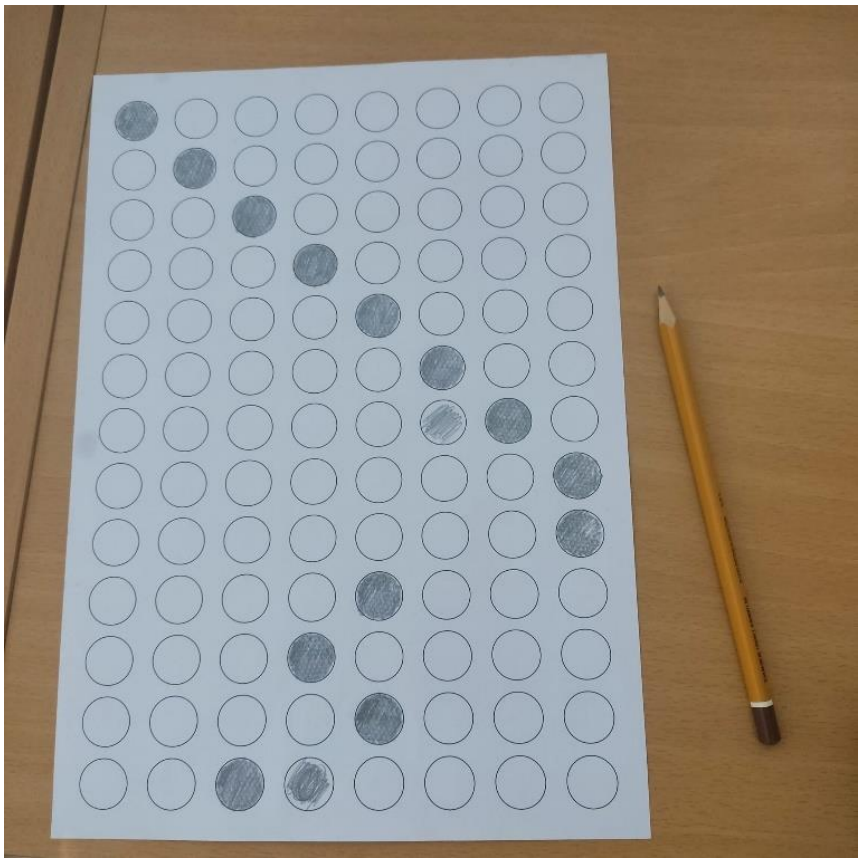
Analizom rezultata koji je zahtjev vratio u json obliku može se primjetiti da je program pronašao obrube svih pitanja te točno zabilježio svako odabrano pitanje. Na ovaj način kreirali smo API koji efikasno prepoznaje slike testova i precizno bilježi odgovore. U nastavku se

nalaze slike nekoliko primjera testova, zajedno s njihovim odgovarajućim rezultatima. Ovi primjeri demonstriraju točnost i pouzdanost API-ja u detekciji i analizi obrazaca odgovora na različitim vrstama testova.



Slika 29: Primjer API test 5x5


```
{"answers":{"answers":[{"question":1,"selected_answers":[0]},
{"question":2,"selected_answers":[1]},
{"question":3,"selected_answers":[2]},
{"question":4,"selected_answers":[3]},
{"question":5,"selected_answers":[4]}]
,"detected_contours":25,"message":"All questions found."}}
```



Slika 30: Primjer Api test 13x8 (Izvor: Vlastita izrada)


```
{"answers":{"answers":
[{"question":1,"selected_answers":[0]},
{"question":2,"selected_answers":[1]},
{"question":3,"selected_answers":[2]},
{"question":4,"selected_answers":[3]},
{"question":5,"selected_answers":[4]},
{"question":6,"selected_answers":[5]},
{"question":7,"selected_answers":[5,6]},
{"question":8,"selected_answers":[7]},
{"question":9,"selected_answers":[7]},
{"question":10,"selected_answers":[4]},
{"question":11,"selected_answers":[3]},
{"question":12,"selected_answers":[4]},
{"question":13,"selected_answers":[2,3]}],
"detected_contours":104,"message":"All questions found."}}
```

6. Zaključak

Ovaj rad rješava problem kritične potrebe u obrazovnom sustavu; potrebe za automatizacijom procesa ocjenjivanja ispita kako bi se povećala učinkovitost i točnost. Ručno ocjenjivanje nije samo dug proces, već je i sklono ljudskim pogreškama, dok automatizirani sustavi mogu uštedjeti vrijeme, smanjiti troškove i osigurati dosljednost i preciznost.

Primarni cilj ovog rada bio je razviti softver sposoban za točno otkrivanje i ocjenjivanje označenih odgovora na pisanim ispitima. Sekundarni ciljevi uključivali su evaluaciju učinkovitosti softvera i analizu rezultata kako bi se identificirala područja za buduća poboljšanja. Struktura rada bila je organizirana tako da prvo predstavi temeljne tehnologije, algoritme i metode korištene u obradi slike za detekciju odgovora. Nakon toga slijedio je praktični dio, koji je uključivao specifikaciju, modeliranje i implementaciju softvera za automatsku detekciju odgovora. Na kraju, sekcija evaluacije analizirala je točnost softvera, raspravljajući o njegovim snagama, slabostima i potencijalnim područjima za poboljšanje.

Razvijeni softver uspješno je demonstrirao sposobnost detekcije i ocjenjivanja označenih odgovora na ispitima, postigavši visoki stupanj točnosti u optimalnim uvjetima. Međutim, analiza je također istaknula ograničenja pri radu s varijacijama u formatima ispita i kvaliteti slike. Budući rad mogao bi se usredotočiti na daljnje usavršavanje softvera kroz poboljšano podešavanje parametara, integraciju tehnika strojnog učenja za povećanu prilagodljivost i razvoj korisnički prijateljskog sučelja koje bi olakšalo širu primjenu kao na primjeru testiranja programa u realnoj okolini gdje se implementirao robusniji sustav. Ovaj projekt omogućio mi je istraživanje i primjenu relevantnih tehnologija za razvoj softverskog rješenja koje može potencijalno revolucionirati proces ocjenjivanja.

Ova poboljšanja pomogla bi učvrstiti ulogu softvera kao pouzdanog alata za automatizirano ocjenjivanje ispita, što bi u konačnici doprinijelo učinkovitijim i točnijim obrazovnim procjenama.

Popis literature

About. (bez dat.). OpenCV. Preuzeto 14. kolovoz 2024., od https://opencv.org/about/

Balla, Erika. (2023, svibanj 31). *Automated Grading Systems: How AI is Revolutionizing Exam Evaluation - DataScienceCentral.com*. Data Science Central. https://www.datasciencecentral.com/automated-grading-systems-how-ai-is-revolutionizing-exam-evaluation/

Gonzalez, R. C., Woods, R. E., & Masters, B. R. (2009). Digital Image Processing, Third Edition. *Journal of Biomedical Optics*, *14*(2), 029901. https://doi.org/10.1117/1.3115362

Kaehler, A., & Bradski, G. (2017). *Learning OpenCV 3: Computer Vision in C++ With the OpenCV Library*.

Kulhary, R. (2019, rujan 23). *What is OpenCV Library?* GeeksforGeeks. https://www.geeksforgeeks.org/opencv-overview/

Kumar , N. (2018, siječanj 26). *Digital Image Processing Basics*. GeeksforGeeks. https://www.geeksforgeeks.org/digital-image-processing-basics/

Kundu, R. (2024). *Image Processing: Techniques, Types, & Applications [2024]*. https://www.v7labs.com/blog/image-processing-guide, https://www.v7labs.com/blog/image-processing-guide

OpenCV: Canny Edge Detection. (bez dat.). Preuzeto 14. kolovoz 2024., od https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html

OpenCV: Image Thresholding. (bez dat.). Preuzeto 14. kolovoz 2024., od https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html

OpenCV: Smoothing Images. (bez dat.). Preuzeto 14. kolovoz 2024., od https://docs.opencv.org/3.4/d4/d13/tutorial_py_filtering.html

OpenCV: Sobel Derivatives. (bez dat.). Preuzeto 14. kolovoz 2024., od https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

Quan, N. (2023, studeni 3). Difference Between Computer Vision and Image Processing. *Eastgate Software*. https://eastgate-software.com/difference-between-computer-vision-and-image-processing/

Singh, H. (2019). *Practical Machine Learning and Image Processing: For Facial Recognition, Object Detection, and Pattern Recognition Using Python*. Apress. https://doi.org/10.1007/978-1-4842-4149-3

Szeliski, R. (2011). *Computer Vision: Algorithms and Applications*. Springer London. https://doi.org/10.1007/978-1-84882-935-0

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., & contributors, the scikit-image. (2014). scikit-image: Image processing in Python. *PeerJ*, *2*, e453. https://doi.org/10.7717/peerj.453

What is Computer Vision? | *IBM*. (2021, srpanj 27). https://www.ibm.com/topics/computer-vision

Rosebrock, A. (2016, listopad 3). Bubble sheet multiple choice scanner and test grader using OMR, Python, and OpenCV. PyImageSearch. https://pyimagesearch.com/2016/10/03/bubble-sheet-multiple-choice-scanner-and-test-grader-using-omr-python-and-opencv/

Popis slika

Šalica čaja,Laurel F from Seattle, WA, CC BY-SA 2.0
<https://creativecommons.org/licenses/by-sa/2.0>, via Wikimedia Commons
Accessed: Aug. 05, 2024. [Online]. Available:
https://upload.wikimedia.org/wikipedia/commons/8/87/Cup_of_tea%2C_Scotland.jpg

Slika 1: Primjer RGB slike: Šalica čaja (Laurel Fan, Izvor: Internet) 3
Slika 2: Primjer binarizacije (Izvor: Vlastita izrada) 3
Slika 3: Primjer greyscalea (Izvor: Vlastita izrada) 4
Slika 4: Salted grayscale šalica (Izvor: Vlastita izrada) 9
Slika 5: Primjer Median Blura (Izvor: Vlastita izrada) 9
Slika 6: Gaussian filter (Izvor: Vlastita izrada) 10
Slika 7:Bilateral filter (Izvor: Vlastita izrada) 11
Slika 8: Primjer detekcije ruba Sobel Operatorom (Izvor: Vlastita izrada) 12
Slika 9: Primjer Cannyjev Detektor Rubova(Izvor: Vlastita izrada) 13
Slika 10: Prilagodljivi prag (Izvor: Vlastita izrada) 14
Slika 11:Adaptive Thresholding (Izvor: Vlastita izrada) 15
Slika 12: K-means Clustering(Izvor: Vlastita izrada) 15
Slika 13: Use-case diagram (Izvor: Vlastita izrada) 20
Slika 14: Dijagram klasa (Izvor: Vlastita izrada) 22
Slika 15: Dijagram slijeda (Izvor: Vlastita izrada) 24
Slika 17: Primjer uspješno obrađenog ispita (Izvor: Vlastita izrada) 35
Slika 16: Primjer dvostrukog odgovora (Izvor: Vlastita izrada) 36
Slika 18: Primjer rotirano slikano ispita (Izvor: Vlastita izrada) 36
Slika 19: Primjer izlaženja van okvira kruga (Izvor : Vlastita izrada) 37
Slika 20: Primjer pogrešno obrađenog testa ,točan odgovor za svako pitanje je B, nisu pronađene sve konture (Izvor: Vlastita izrada) 38
Slika 21: Primjer pogrešno obrađenog ispita - slabo obilježen odgovor (Izvor: Vlastita izrada) 38
Slika 22: Primjer neuspješno obrađenog ispita - nepronalaženje kontura pitanja (Izvor: Vlastita izrada) 39

Slika 23 : Primjer označenog ispita 15x8 (Izvor: Vlastita izrada) 41
Slika 24: Primjer ispita 13x8 (Izvor: Vlastita izrada) 41
Slika 25: Primjer šuma (Izvor: Vlastita izrada)..... 42
Slika 26: Primjer korištenja adaptivnog praga (Izvor: Vlastita izrada)..... 43
Slika 27: Primjer podataka zahtjeva (Izvor: Vlastita izrada) 46
Slika 28: Primjer ispita za API 15x8 (Izvor:Vlastita izrada)..... 47
Slika 29: Primjer API test 5x5 48
Slika 30: Primjer Api test 13x8 (Izvor: Vlastita izrada)..... 49