Primjena tehnika procesiranja slike u svrhu prepoznavanja registarskih oznaka automobila

Šimić, Frano

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Frano Šimić

PRIMJENA TEHNIKA PROCESIRANJA SLIKE U SVRHU PREPOZNAVANJA REGISTARSKIH OZNAKA AUTOMOBILA

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Frano Šimić

Matični broj: 0016152548

Studij: Informacijski i poslovni sustavi

PRIMJENA TEHNIKA PROCESIRANJA SLIKE U SVRHU PREPOZNAVANJA REGISTARSKIH OZNAKA AUTOMOBILA

ZAVRŠNI RAD

Mentor :

Doc. dr. sc. Marko Mijač

Varaždin, rujan 2024.

Frano Šimić

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 potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U teorijskom dijelu ovog rada bit će opisane ključne tehnologije, algoritmi, alati i biblioteke koje olakšavaju i omogućuju obradu slika. U praktičnom dijelu rada bit će razvijen softver za automatizirano otvaranje parkirne rampe na temelju prepoznavanja registarskih oznaka vozila. Za pouzdaniju identifikaciju vozila, osim registarskih oznaka, bit će korištena i boja automobila.

Ključne riječi: Tesseract, EMGU.CV, registracija, prepoznavanje teksta, prepoznavanje oblika, prepoznavanje objekata, procesiranje slike

Sadržaj

1. Uvod

Ovaj završni rad bavi se istraživanjem i izradom programskog rješenja za automatizirano otvaranje parkirne rampe na temelju prepoznavanja registarskih oznaka vozila i njegove boje. Rješenje uključuje jednostavni prototip koji pomoću slike može prepoznati registarske oznake i boju automobila. Sustav također posjeduje jednostavnu bazu podataka u kojoj se mogu spremati podaci o automobilu kako bi prototip bio što vjerniji i točniji. Takav sustav, uz potrebni hardver i prilagođeno programsko rješenje, može imati praktičnu primjenu. Parkirališta privatnih poduzeća, podzemne garaže stambenih zgrada ili hotela su samo neki od primjera gdje bi sustav mogao imati široku primjenu.

Uz izradu programskog rješenja, svrha ovog rada je istraživanje tehnika procesiranja slike i potrebnih alata. Cilj je alate proučiti, upotrijebiti pri izradi programskog rješenja i na kraju ocijeniti alate na temelju učinkovitosti i jednostavnosti.

Programsko rješenje se izrađuje u C# programskom jeziku s .NET programskim okvirom. Preporučuje se korištenje Visual Studio razvojnog okruženja koje je najpogodnije za izradu složenih aplikacija programiranje u C#-u. Uz to, Visual Studio omogućuje jednostavnu instalaciju svih potrebnih alata. Za procesiranje slike koristi se OpenCV alat koji ima jako široku primjenu kada je riječ o računalnom vidu. Budući da OpenCV nije kompatibilan s C#-om, potrebno je koristiti EmguCV omotač koji omogućava korištenje OpenCV funkcija u C#-u. Za čitanje znakova se koristi Tesseract alat koji je specijaliziran za tu svrhu i ima podršku za većinu svjetskih jezika. Aplikacija ima troslojnu arhitekturu tj. ima prezentacijski sloj, sloj poslovne logike i sloj pristupa bazi podataka. Za izradu grafičkog korisničkog sučelja se koristi WPF (Windows presentation foundation) alat koji je u sklopu .NET okvira.

Rad je organiziran u četiri poglavlja. U prvom poglavlju su opisane osnovne tehnike za procesiranje slike te tehnike detekcije objekata na slici. Drugo poglavlje opisuje alate za procesiranje slike koji se koriste u programskom rješenju. Nakon toga, u trećem dijelu, opisan je postupak izrade programskog rješenja te detaljan opis korištenih algoritama. Za kraj, prikazan je način rada i izgled aplikacije te osobni osvrt na korištene alate

Ovaj rad pruža teorijsku podlogu za razumijevanje procesiranja slike i primjer programskog rješenja, pa bi njegova primjena mogla biti široka. Zainteresirani za procesiranje slike, mogu u ovom radu pronaći osnovne informacije i temeljna znanja neovisno o primjeni programskog rješenja. Programsko rješenje se može iskoristiti kao prototip, tako da se može prilagođavati prema potrebi. Uz to, planiran je daljnji razvoj aplikacije koja će raditi sa pravom kamerom u stvarnom vremenu.

2. Osnove procesiranja slike i objekata

Računalni vid je oblik umjetne inteligencije koji trenira strojeve za interpretaciju i razumijevanje vizualnog svijeta. Koristeći vizualne podatke, strojevi se mogu naučiti da točno identificiraju i klasificiraju objekte, te donesu odluku ili poduzmu određene korake temelju onoga što "vide" [1].

Obrada slike je osnovna komponenta računalnog vida, omogućavajući strojevima da tumače i analiziraju vizualne informacije. Ključni aspekti obrade slika su:

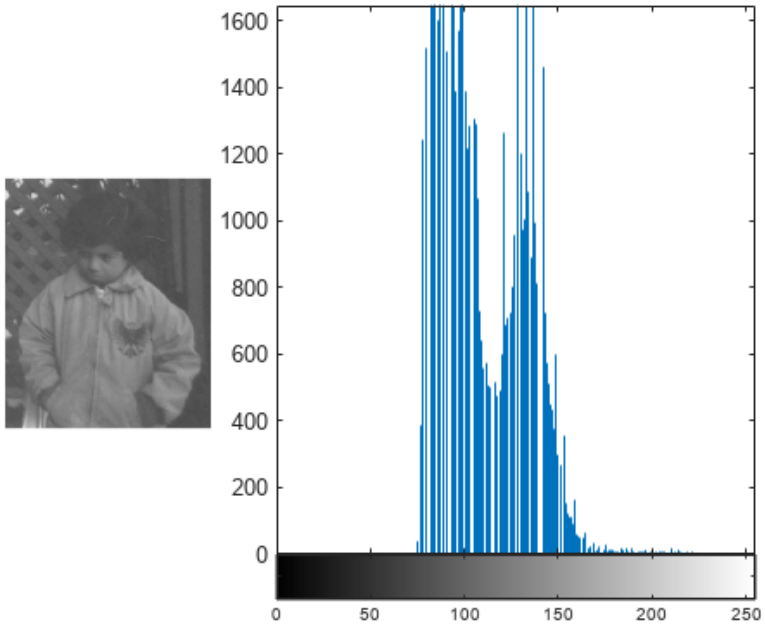
- **Unapređenje slike**: Ovo uključuje poboljšanje kvalitete slike podešavanjem atributa poput svjetline, kontrasta i oštrine. Tehnike uključuju izjednačavanje histograma i smanjenje šuma.
- **Segmentacija slike**: Ovaj korak uključuje dijeljenje slike na više segmenata kako bi se pojednostavila analiza.
- **Prepoznavanje slike**: Prepoznavanje slike uključuje identificiranje i kategoriziranje objekata ili uzoraka unutar slike. Algoritmi strojnog učenja poput konvolucijskih neuronskih mreža (CNN) često se koriste za ovaj zadatak.
- **Detekcija objekata**: Detekcija objekata ide korak dalje, ne samo da prepoznaje objekte, već ih i locira unutar slike. Ovo je presudno za zadatke poput autonomne vožnje i nadzora [2].

2.1. Tehnike za preprocesiranje slike

Postoji mnogo tehnika za procesiranje slike, ali prije procesiranja, slika se mora pripremiti. Taj proces se naziva preprocesiranje slike. Da bi se slika mogla procesirati ili da se na njoj prate objekti, mora se pripremiti tako da stroju bude najlakša za obradu.

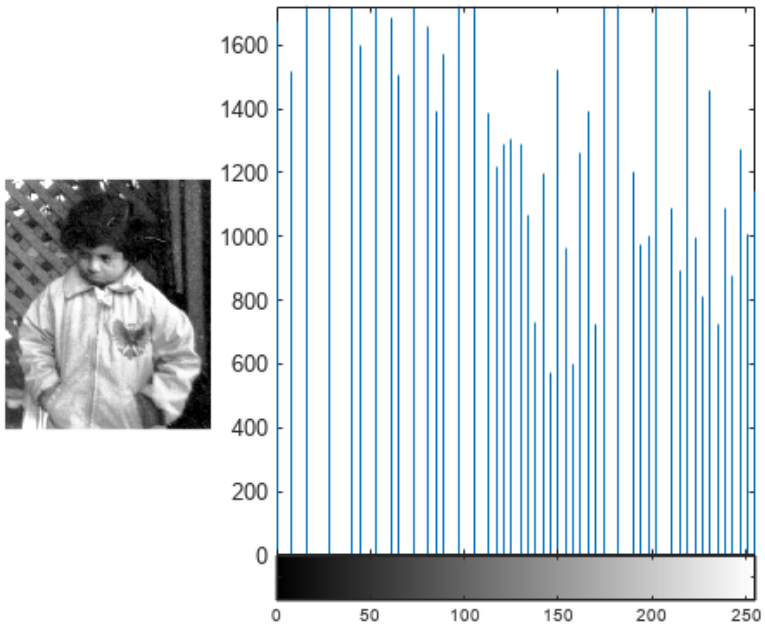
2.1.1. Izjednačavanje histograma

Izjednačavanje histograma je jedna od najčešćih tehnika za uređivanje kontrasta slike. Histogram slike prikazuje koliko piksela na slici ima isto osvjetljenje. Na slici 1 može se vidjeti fotografija djevojčice i pripadni histogram s desne strane. Na histogramu se jasno vidi da je previše jedne boje i njenih nijansi. Siva boja prevladava slikom dok crne i bijele gotovo da i nema. Ovakva slika je jako teška za procesiranje, pa je stoga potrebno popraviti kontrast. Također, izjednačavanje histograma radi i u suprotnom smjeru. Ako slika ima prevelik kontrast, također se može popraviti izjednačavanjem histograma. Za primjer na slici 1 uzeta je crno bijela boja, no izjednačavanje histograma se može primijeniti i na zelenu, crvenu i plavu boju.



Slika 1: Fotografija prije izjednačavanja histograma

Kada se na sliku 1 primjeni izjednačavanje histograma dobije se situacija kao na slici 2. Može se na prvi pogled primijetiti da je slika jasnija, a ako se pogleda histogram, može se vidjeti da je raspored boja bolji i da slika sadrži većinu nijansi od crne do bijele boje.



Slika 2: Fotografija nakon izjednačavanja histograma

2.1.2. Gaussovo smanjenje šuma

Gaussovo smanjenje šuma je tehnika koja se koristi za smanjenje šuma na slici i omekšavanje rubova. Ova metoda koristi Gaussovu funkciju, poznatu po svojoj sposobnosti da izgladi slike na način koji je nježan prema rubovima, zadržavajući važne strukture dok uklanja sitne nepravilnosti. Zbog toga je Gaussovo zamućivanje često prvi korak u složenijim algoritmima obrade slike, kao što su detekcija rubova i prepoznavanje objekata [3].

Gaussova funkcija koristi se za izračunavanje težine svakog piksela unutar kernela, matrice koja se pomiče preko slike kako bi se izračunala nova vrijednost za svaki piksel. Funkcija je definirana kao:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

gdje je:

- $G(x, y)$ vrijednost Gaussove funkcije u točki (x, y), koja predstavlja težinu piksela,
- σ standardna devijacija koja određuje širinu zamućivanja; manja vrijednost σ rezultira oštrijim slikama, dok veća vrijednost proizvodi mekše, zamućenije slike,
- x i y koordinate unutar kernela koje određuju relativnu poziciju piksela u odnosu na središnji piksel.

Kernel se primjenjuje na svaki piksel slike, a nova vrijednost piksela dobiva se zbrajanjem umnožaka originalnih vrijednosti piksela i odgovarajućih težina izračunatih Gaussovom funkcijom. Ovaj proces rezultira glatkom tranzicijom između boja, smanjujući vidljivost šuma dok se zadržavaju ključne informacije [3]. Na slici 3 prikazan je primjer filtriranja slike korištenjem Gaussovog filtra za smanjenje šuma:

Slika 3: Primjena Gaussovog filtra za smanjenje šuma na slici - desno slika bez filtera - lijevo slika s filterom

[4]

2.1.3. Cannyjev algoritam za detekciju rubova

Cannyjev algoritam je jedan od najpoznatijih algoritama za detekciju rubova na slikama. Algoritam sadrži nekoliko ključnih koraka, a to su:

- Gaussovo smanjenje šuma
- Izračunavanje gradijenta
- Određivanje praga na osnovi magnituda gradijenta
- Dvostruko pragiranje
- Praćenje rubova pomoću histereze

Gaussovo smanjenje šuma: Prvi korak u Cannyjevom algoritmu je smanjenje šuma na slici pomoću Gaussovog filtra. Ovaj filter koristi Gaussovu funkciju za smanjenje sitnog šuma. Primjenom Gaussovog filtra na svaki piksel slike i korištenjem težina koje opadaju s udaljenošću od središnjeg piksela smanjuju se nesigurni detalji i šum, čime se omogućuje preciznija detekcija rubova u sljedećim koracima [5].

Izračunavanje gradijenta: Nakon smanjenja šuma, sljedeći korak je izračunavanje gradijenta slike kako bi se identificirale promjene u intenzitetu. Gradijent se izračunava pomoću operatora poput Sobelovog operatora, koji mjeri promjenu intenziteta u horizontalnom i vertikalnom smjeru. Magnitudu gradijenta i smjer izračunavamo pomoću formula:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

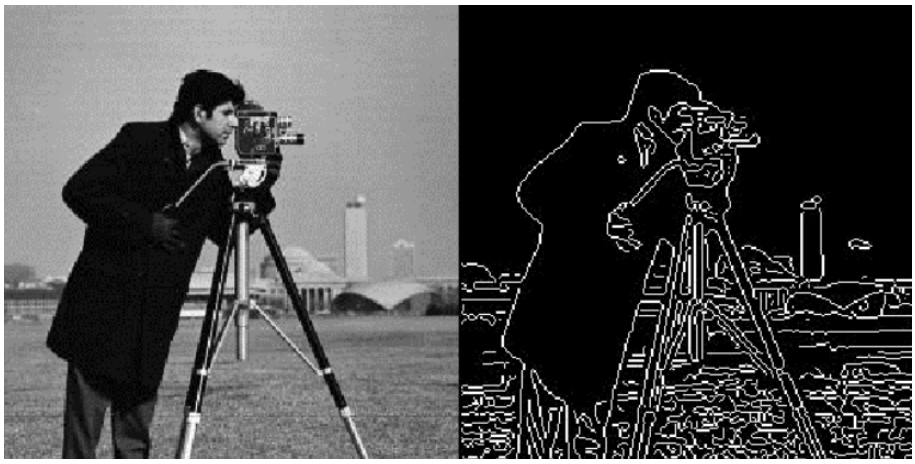
Gdje su G_x i G_y gradijenti u horizontalnom i vertikalnom smjeru. [5]

Određivanje praga na osnovi magnituda gradijenta: U ovom koraku, magnituda gradijenta koristi se za određivanje koji pikseli predstavljaju potencijalne rubove. Pragiranje magnituda gradijenta uključuje postavljanje praga kako bi se filtrirali pikseli prema njihovoj jačini rubova. Pikseli čija magnituda gradijenta prelazi određeni prag označeni su kao rubovi, dok se oni ispod praga ignoriraju [5].

Dvostruko pragiranje: Dvostruko pragiranje je metoda koja koristi dva praga: visoki i niski. Pikseli s magnitudom gradijenta iznad visokog praga smatraju se jakim rubovima, dok su oni između niskog i visokog praga označeni kao slabi rubovi. Ova metoda omogućuje razlikovanje između jasnih i potencijalnih rubova, što pomaže u preciznoj detekciji rubova [5].

Praćenje rubova pomoću histereze: Zadnji korak u Cannyjevom algoritmu je praćenje rubova pomoću histereze. Ovaj proces eliminira slabe rubove koji nisu povezani s jakim rubovima. Svi slabi rubovi povezani s jakim rubovima zadržavaju se, dok se ostali slabi rubovi odbacuju. Ovaj korak osigurava da rubovi budu kontinuitetni i da se eliminiraju nesigurni

ili nepotrebni detalji [5]. Na slici 4 može se vidjeti rezultat Cannyjevog algoritma za detekciju rubova.



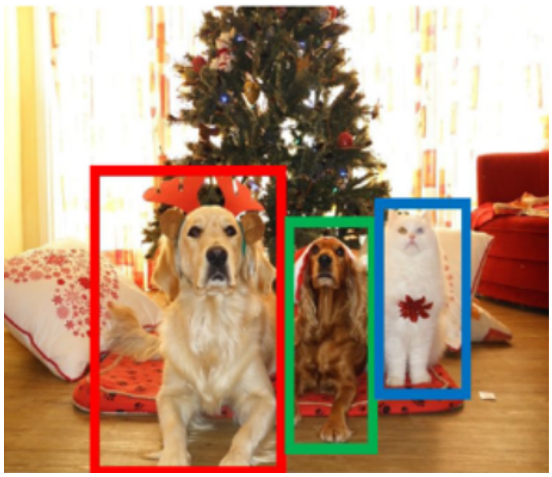
Slika 4: Primjena Cannyjevog algoritma za detekciju rubova na slici -desno ulazna slika - lijevo slika s detektiranim rubovima

Cannyjev algoritam je jedan od najčešće korištenih algoritama u obradi slike zbog svoje učinkovitosti i preciznosti. Ovaj algoritam je izuzetno važan u različitim područjima računalnog vida, gdje se koristi za detekciju rubova i segmentaciju slika. Cannyjev algoritam se koristi za analizu video materijala, omogućavajući detekciju i praćenje kretanja objekata i osoba. Ova primjena je ključna za sustave video nadzora i analizu sigurnosnih snimaka. U medicini, Cannyjevg algoritam igra važnu ulogu u analizi medicinskih slika kao što su MRI i CT skenovi. Algoritam pomaže u preciznom označavanju i prepoznavanju rubova različitih anatomskih struktura.

2.2. Procesiranje slike

Nakon završetka preprocesiranja, slika je spremna za daljnju obradu. Ovisno o specifičnom problemu, mogu se primijeniti različite tehnike procesiranja slike. U ovom poglavlju bit će pojašnjeni segmentacija slike i detekcija objekata. Iako obje tehnike imaju slične ciljeve, one se razlikuju u svojoj primjeni. Detekcija objekata služi za lociranje objekata unutar slike. Ova metoda identificira i označava specifične objekte pomoću okvira, što omogućuje prepoznavanje njihove lokacije i vrste. S druge strane, segmentacija slike razdvaja sliku na regije ili segmente. Ova tehnika detaljno analizira slike kako bi odvojila različite dijelove slike ili objekte, omogućavajući precizno izdvajanje i analizu svakog segmenta.

Najbolji prikaz razlike između detekcije objekata i segmentacije slike može se vidjeti na slikama 5 i 6. Osim načina na koji rade, postoji razlika i u njihovim primjenama.



Slika 5: Detekcija objekata

Slika 6: Segmentacija slike

2.2.1. Segmentacija slike

Segmentacija slike je tehnika koja razdvaja sliku na različite regije koje predstavljaju različite objekte ili značajne strukture unutar slike. Ova metoda omogućava detaljno razumijevanje slike i precizno izdvajanje objekata na temelju njihovih svojstava. Postoje dvije vrste segmentacije slike: semantička segmentacija i instancijska segmentacija [6].

Semantička segmentacija dodjeljuje oznake svim pikselima u slici na temelju njihovih semantičkih kategorija, kao što su "cesta", "nebo" ili "osoba". Svaki piksel u slici se klasificira u jednu od tih kategorija. Cilj je da se svaki dio slike pripiše odgovarajućoj klasi, omogućujući precizno prepoznavanje i analiziranje različitih područja slike [6]. Na slici 7 može se vidjeti primjer kako se semantičkom segmentacijom može prepoznati područje na slici koje sadrži ovce.

Slika 7: Prepoznavanje regije s ovcama koristeći semantičku segmentaciju

Primjenjuju se algoritmi poput Fully Convolutional Networks (FCNs) i U-Net koji koriste duboke konvolucijske mreže za segmentaciju slike na osnovi sadržaja [6].

Segmentacija instanci proširuje semantičku segmentaciju tako što razlikuje instance iste kategorije objekta [6]. Na primjer, na slici 8 se prepoznaju ovce te svaka ovca posebno.

Slika 8: Prepoznavanje regije s ovcama koristeći segmentaciju instanci

Algoritmi poput Mask R-CNN omogućuju precizno označavanje i razlikovanje različitih instanci objekata u slici, pružajući maske koje označavaju granice svake instance objekta [6].

Segmentacija slike se koristi u različitim scenarijima gdje je potrebno precizno razdvajanje i analiza različitih regija unutar slike. U medicinskoj dijagnostici, segmentacija omogućuje izdvajanje specifičnih dijelova tijela ili abnormalnosti, kao što su tumori ili lezije, iz medicinskih slika poput MRI ili CT skenova. U području autonomnih vozila, segmentacija slike omogućava prepoznavanje i razdvajanje različitih objekata na cesti, kao što su vozila, pješaci i prometni znakovi. Ovo omogućava vozilima da razumiju i reagiraju na svoje okruženje u stvarnom vremenu, poboljšavajući sigurnost i navigaciju.

2.2.2. Detekcija objekata

Detekcija objekata je metoda koja identificira i locira specifične objekte unutar slike. Ova tehnika koristi okvire za označavanje područja u slici gdje se nalaze objekti. Svaki okvir predstavlja područje s objektom i može biti označen vrstom objekta. Ovi okviri omogućuju brz i učinkovit lociranje i klasifikaciju objekata unutar slike [6].

Algoritmi za detekciju objekata, poput YOLO (You Only Look Once) koji je poznat po svojoj brzini i preciznosti te SSD (Single Shot MultiBox Detector) koji nudi dobru ravnotežu između točnosti i brzine, koriste okvire za brzo prepoznavanje i klasifikaciju objekata u stvarnom vremenu [6].

Slika 9: Detekcija automobila na slici

Na slici 9 prikazan je primjer detekcije vozila. Bijeli okviri označavaju prepoznate automobile. Sustavi za detekciju objekata mogu prepoznavati različite klase objekata i označiti ih različitim bojama. Na primjer, sustav može detektirati ovce označene ljubičastim okvirom i pse označene žutim okvirom.

Detekcija objekata je posebno korisna kada je potrebno prepoznati i locirati jasno definirane i vidljive oblike, kao što su automobili ili životinje. Također, koristi se kada je cilj brzo pronaći lokaciju objekta bez potrebe za detaljnom analizom svake regije slike. Na primjer, sustav za brojanje ovaca može koristiti detekciju objekata za jednostavno brojanje prolaznih životinja, bez potrebe za kompleksnim segmentacijskim procesima.

3. Alati za prepoznavanje objekata

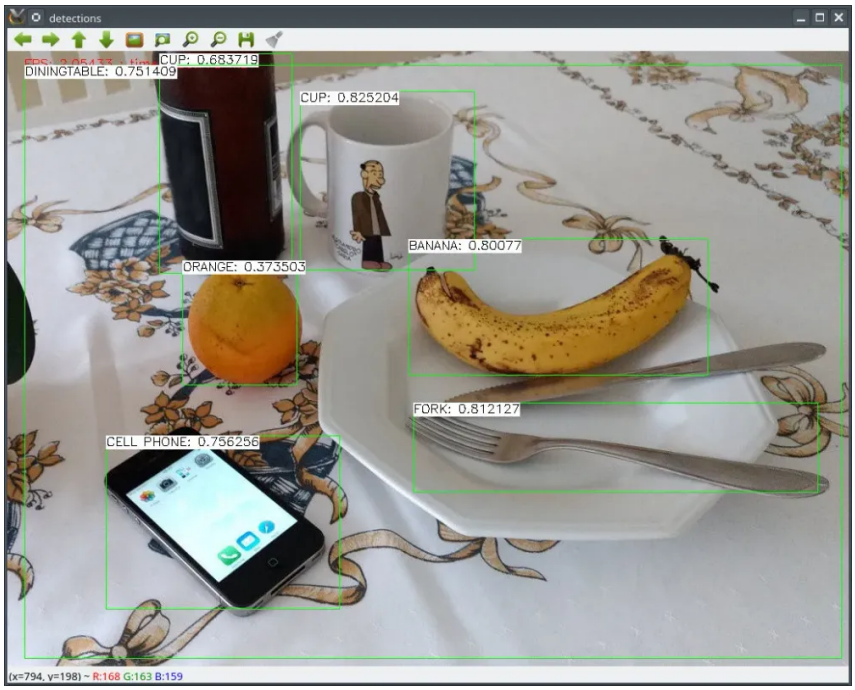
U ovom poglavlju će biti opisani neki od najčešće korištenih alata za prepoznavanje objekata na slici. Prvo će biti opisan OpenCV biblioteka, zatim EmguCV omotač te na kraju Tesseract alat za prepoznavanje znakova.

3.1. OpenCV

OpenCV (Open Source Computer Vision Library) je biblioteka otvorenog koda za računalni vid i strojno učenje. Napravljena je kako bi osigurala zajedničku infrastrukturu za aplikacije računalnog vida i ubrzala korištenje strojne percepcije u komercijalnim proizvodima. Budući da je proizvod s licencom Apache 2, OpenCV tvrtkama olakšava korištenje i izmjenu koda. OpenCV ima zajednicu koja broji više od 47 tisuća korisnika i procijenjeni broj preuzimanja je veći od 18 milijuna. Biblioteka se intenzivno koristi u tvrtkama, istraživačkim grupama i vladinim tijelima [7].

3.1.1. Povijest razvoja

Razvoj OpenCV biblioteke započeo je 1999. godine u Intel Research Labs, pod vodstvom Garyja Bradskog. Prvotno razvijena na C i C++ jezicima, biblioteka je kasnije proširena podrškom za moderne jezike poput Pythona, s ciljem pružanja zajedničke infrastrukture za računalni vid i ubrzanja istraživanja u tom području. Prva verzija OpenCV-a objavljena je 2000. godine, nudeći osnovne funkcionalnosti za obradu slika. Godine 2006. uvedeno je novo C++



Slika 10: Primjer detekcije objekata pomoću OpenCV-a [8]

sučelje, a 2008. dodana je podrška za GPU ubrzanje. Verzija 2.0 iz 2010. godine donijela je značajna poboljšanja performansi i kompatibilnost s različitim platformama. OpenCV 3.0, objavljen 2015. godine, uveo je novo C++11 sučelje i napredne algoritme strojnog učenja, dok je verzija 4.0 iz 2018. godine integrirala mogućnosti dubokog učenja. Danas, OpenCV nastavlja evoluirati. Najnovija verzija, v4.9, uključuje poboljšanja modula za detekciju objekata, novi API za praćenje objekata temeljen na vidnim transformatorima, te podršku za najnovije platforme poput AppleVisionOS-a i Androida [8].

Predviđa se veliki napredak u razvoju OpenCV-a te računalnog vida općenito. S porastom edge computinga i Interneta stvari (IoT), potražnja za laganim i učinkovitim rješenjima za računalni vid će rasti. OpenCV konstantno radi na napretku svojih algoritama, pa se u budućnosti mogu očekivati još brži i kvalitetniji sustavi[8].

Paralelno s tim, rastuća zabrinutost za privatnost dovest će do usmjerenja na tehnike očuvanja privatnosti unutar sustava računalnog vida. Budućnost umjetne inteligencije bit će usmjerena na osiguravanje odgovorne upotrebe ove moćne tehnologije [8].

3.1.2. Funkcionalnosti

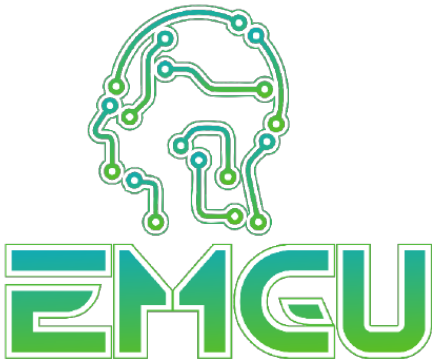
Biblioteka ima više od 2500 optimiziranih algoritama, što uključuje veliki broj klasičnih i modernih algoritama računalnog vida i strojnog učenja. Ovi se algoritmi mogu koristiti za otkrivanje i prepoznavanje lica, identifikaciju objekata, klasificiranje ljudskih radnji u videozapisima, praćenje pokreta kamere, praćenje pokretnih objekata, izdvajanje 3D modela objekata, proizvodnju 3D oblaka točaka iz stereo kamera, spajanje slika kako bi se proizvela slika visoke rezolucije cijelog prizora, pronalaženje sličnih slika iz baze podataka, uklanjanje očiju sa slika snimljenih bljeskalicom, praćenje pokreta očiju, prepoznavanje krajolika i postavljanje markera za prekrivanje s proširenom stvarnošću [7].

Uz stabilne tvrtke kao što su Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota koje financiraju razvoj biblioteke, postoji mnogo startupova kao što su Applied Minds, VideoSurf i Zeitera, koji u velikoj mjeri koriste OpenCV. OpenCV ima široku upotrebu od spajanja slika s prikazom ulice, otkrivanja upada u video nadzor u Izraelu, praćenja rudničke opreme u Kini, pomaganja robotima u navigaciji i podizanju predmeta u Willow Garageu, otkrivanja nesreća utapanja u bazenu u Europi, pokretanja interaktivne umjetnosti u Španjolskoj i New Yorku, provjera avionskih pista za krhotine u Turskoj, pregled naljepnica na proizvodima u tvornicama diljem svijeta do brzog prepoznavanja lica u Japanu[7].

Ima C++, Python, Java i MATLAB sučelja i podržava Windows, Linux, Android i Mac OS. OpenCV se uglavnom oslanja na aplikacije vida u stvarnom vremenu i koristi MMX i SSE upute kada su dostupne. CUDA i OpenCL sučelja s punim značajkama trenutno se aktivno razvijaju. OpenCV je izvorno napisan u C++ i ima predložak sučelja koji besprijekorno radi sa STL spremnicima[7].

3.2. EmguCV

EmguCV je višeplatformski .NET omotač biblioteke za obradu slika OpenCV, koji omogućuje pozivanje OpenCV funkcija iz .NET kompatibilnih jezika. EmguCV se može kompajlirati pomoću Visual Studio i Unity alata, i može raditi na raznim operativnim sustavima, uključujući Windows, Linux, Mac OS, iOS i Android. EmguCV je u potpunosti napisan u C#, što omogućuje njegovo pokretanje na bilo kojoj platformi koju .NET podržava. Uloženo je puno truda kako bi se postigla čista C# implementacija, jer se zaglavlja moraju prenijeti, za razliku od upravljane C++ implementacije gdje se datoteke zaglavlja mogu jednostavno uključiti [9].



Slika 11: EmguCV logo
[10]

3.2.1. Povijest razvoja

Nema konkretnih podataka o povijesti razvoja EmguCV ,ali s [11] može se vidjeti da je Emgu korporacija osnovana 2007. godine u Ontariu u Kanadi. Prva verzija EmguCV-a izlazi 11. ožujka 2008. godine, a zadnja stabilna verzija 4.7.0 objavljena je 11. rujna 2023. godine [12].

3.2.2. Funkcionalnosti

EmguCV nudi širok raspon funkcionalnosti za obradu slika i računalni vid. Jedna od ključnih funkcionalnosti je filtriranje slike, koje omogućuje primjenu različitih filtera za poboljšanje kvalitete slike među kojima je i Gaussovo zamućivanje koje je često ključni korak za preprocesiranje slike. Druga važna funkcionalnost je detekcija rubova. Koristeći algoritme poput Cannyjeve detekcije rubova, moguće je izdvojiti rubove objekata unutar slike.

Osim toga, EmguCV pruža alate za transformacije slike, kao što su rotacija, skaliranje i translacija. Ove transformacije su ključne u mnogim aplikacijama za računalni vid, uključujući prepoznavanje objekata i analizu slike.

Još jedna značajna funkcionalnost je prepoznavanje objekata, koja omogućuje detekciju

i prepoznavanje različitih objekata koristeći tehnike poput Haar kaskadnih klasifikatora i HOG (Histogram of Oriented Gradients) deskriptora. Ove tehnike su široko korištene u aplikacijama za prepoznavanje lica, registarskih oznaka, vozila i drugih objekata.

3.3. Tesseract

Tesseract je open-source OCR (Optical Character Recognition) alat koji može prepoznati tekst iz različitih formata slika, uključujući TIFF, PNG, JPEG, i PDF. Podržava više izlaznih formata, kao što su plain text, HOCR, i PDF [13]. Zbog visoke točnosti prepoznavanja, osobito za čiste i jasne tekstove, Tesseract je idealan izbor za aplikacije koje koriste slike visokih rezolucija kao ulaz. Danas ima podršku za 116 jezika među kojima je i podrška za hrvatski jezik.

3.3.1. Povijest razvoja

Tesseract je izvorno razvijen u Hewlett-Packard Laboratories u Bristolu, UK, i u Hewlett-Packard Co. u Greeleyu, Colorado, SAD, između 1985. i 1994. godine. Dodatne izmjene su napravljene 1996. godine kako bi se Tesseract mogao koristiti na Widnows-ima i djelomično je kod prebačen u C++, a 1998. godine čitav kod je prebačen u C++. Godine 2005., HP je otvorio izvorni kod Tesseracta, čime je projekt postao dostupan zajednici za daljnji razvoj, a od 2006. godine, razvoj Tesseracta sponzoriran je od strane Googla. Trenutna stabilna verzija, Tesseract 5, je objavljena 2021. nakon više od dvije godine razvoja i testiranja [14]. Novije manje verzije i ispravci grešaka redovito se objavljuju na GitHubu [15].

3.3.2. Funkcionalnosti

Tesseract pruža širok raspon funkcionalnosti, uključujući prepoznavanje više jezika, automatsko otkrivanje orijentacije teksta, podršku za različite formate slika i integraciju s drugim softverskim alatima. Također, može se prilagoditi specifičnim potrebama kroz treniranje na korisničkim skupovima podataka. Tesseract također ima i podršku za pisma koja se pišu s desna na lijevo, kao što je arapski ili hebrejski.

4. Izrada aplikacije za automatsko otvaranje parkirne rampe

Parkirne rampe imaju jedan nepoželjan trošak, a to je plaćanje osobe koja će samo sjediti i podizati rampu, što nekad zahtjeva tri ili čak četiri radnika. S umjetnom inteligencijom za računalni vid, lako se može izraditi aplikacija koja će taj posao obavljati dvadeset četiri sada dnevno bez potrebe za ljudskom intervencijom.

Ono što je potrebno izraditi u aplikaciji jest sustav za računalni vid koji je spojen s bazom automobila kojima se rampa treba otvoriti. Sustav je idealan za korporacijska ili privatna parkirališta kojima je cilj da propuštaju samo vozila registrirana u bazi. Također, sustav se može prenamijeniti da radi za parkirališta na kojima se plaća parking ili bilo koju drugu svrhu gdje je potrebna rampa, ali trenutna domena problema je samo da se propuštaju određena vozila.

Izrada aplikacije sama po sebi je jednostavna. Program ima jedan ulazni podatak i jedan izlazni podatak. Ti podaci su slika ili kadar videozapisa koji se treba obraditi te boolean podatak koji javlja je li registracija prepoznata u bazi ili nije.

Uz sve navedeno, u sustavu mora postojati mogućnost manipulacije spremljenih zapisa u bazi kako bi ovlaštena osoba mogla dodavati nove automobile ili brisati postojeće.

4.1. Specifikacija funkcionalnih zahtjeva

Identifikator	FZ-01
Zahtjev	Prepoznavanje registarskih oznaka
Obrazloženje	Sustav će prepoznavati registarske oznake automobila. Korisnik prvo mora u sustav dodati fotografiju automobila na kojem su jasno vidljive registarske oznake. Jednom kada je slika dodana, sustav prepoznaje registarske oznake, čita tekst sa njih i za output vraća korisniku što je pročitao
Način provjere	Potrebno je dodati fotografiju automobila s registarskom oznakom i provjeriti je li pravilno pročitao registarsku oznaku
Prioritet [1-5]	1
Izvor/Porijeklo	Frano Šimić

Tablica 1: FZ-01 - Prepoznavanje registarskih oznaka

Identifikator	FZ-02
Zahtjev	Prepoznavanje boje automobila
Obrazloženje	Sustav će prepoznavati boju automobila. Korisnik prvo mora u sustav dodati fotografiju automobila. Jednom kada je slika dodana, sustav prepoznaje boju i korisniku ispisuje heksadekadski zapis boje.
Način provjere	Potrebno je dodati fotografiju automobila i provjeriti je li sustav pravilno prepoznao boju.
Prioritet [1-5]	1
Izvor/Porijeklo	Frano Šimić

Tablica 2: FZ-02 - Prepoznavanje boje automobila

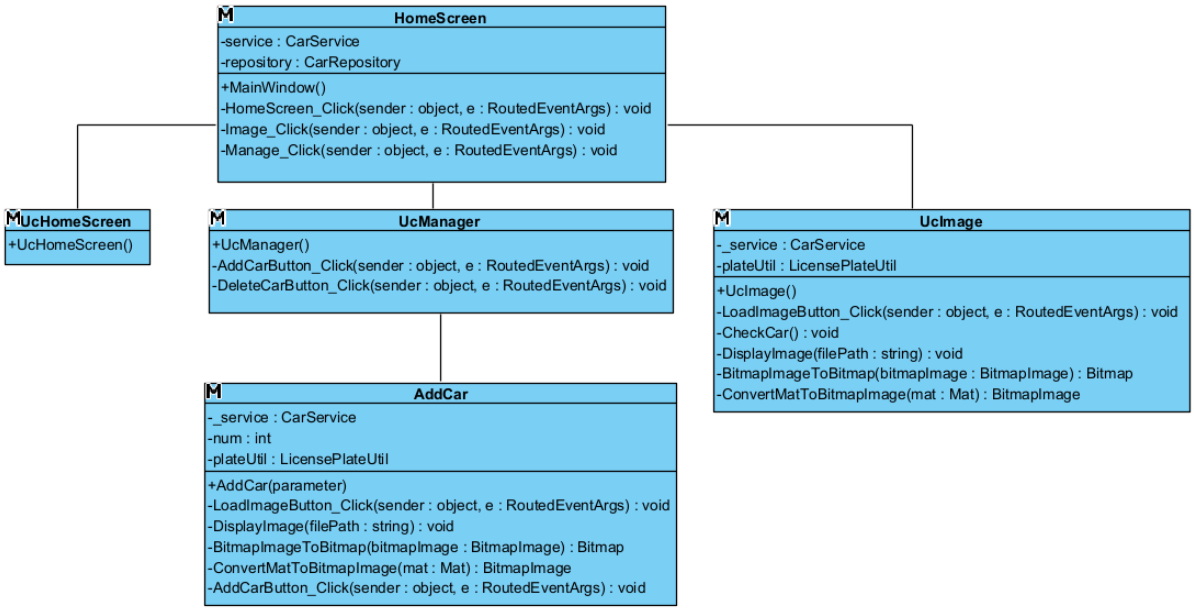
Identifikator	FZ-03
Zahtjev	Provjera zapisa automobila u sustavu
Obrazloženje	Sustav će provjeriti ako se automobil nalazi u bazi podataka te na osnovu toga dati povratnu informaciju. Jednom kada se pročitaju registarske oznake i boja, provjerava se u sustavu ako postoji zapis sa istim podacima. Ukoliko postoji, korisniku se prikazuje zeleno označeni "Passed", ukoliko ne, onda se ispisuje "Reject". Ova funkcionalnost simulira rampu koja se diže ili spušta.
Način provjere	Potrebno je dodati fotografiju automobila koji jest u sustav i fotografiju automobila koji nije u sustavu i provjeriti je li dobar output.
Prioritet [1-5]	1
Izvor/Porijeklo	Frano Šimić

Tablica 3: FZ-03 - Provjera zapisa automobila u sustavu

Identifikator	FZ-04
Zahtjev	Upravljanje automobilima u bazi podataka
Obrazloženje	Sustav će omogućiti korisniku da dodaje novi automobil u sustav ili da ga obriše. Automobil se dodaje tako da se u sustav ubaci fotografija automobila koji prepoznaje registracije i boju, te se na pritisak gumba zapisuju podaci o automobilu u bazu podataka. Na sučelju gdje je prikazana lista svih automobila, potrebno je odabrati jedan automobil koji se može obrisati na pritisak gumba.
Način provjere	Potrebno je dodati podatke o jednom automobilu u sustav i spremiti ih nakon čega će se prikazati na listi svih podataka i obrisati taj zapis nakon čega više ne bi trebao biti vidljiv.
Prioritet [1-5]	1
Izvor/Porijeklo	Frano Šimić

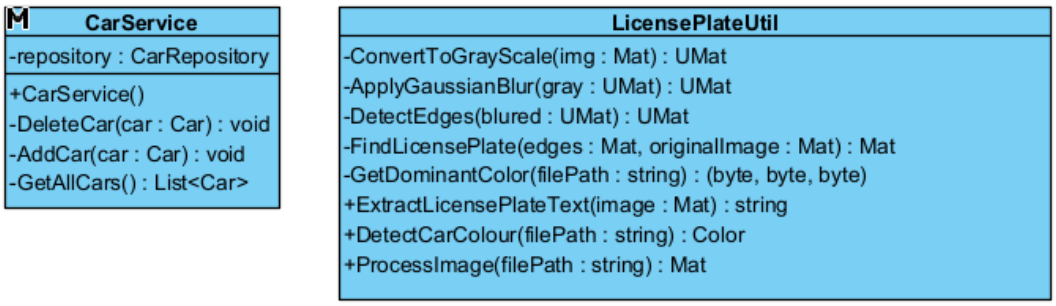
Tablica 4: FZ-04 - Upravljanje automobilima u bazi podataka

4.2. Specifikacija dizajna



Slika 12: Dijagram klasa PresentationLayer

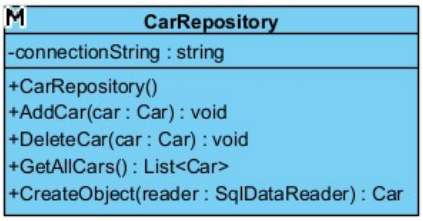
Na dijagramu klasa [12] može se proučiti prezentacijski sloj aplikacije. Glavni prozor na kojem se odvija aplikacija je izrađena u *Home Screen* klasi. *Home Screen* klasa sadrži nekoliko funkcija za upravljanje korisničkim kontrolama koje omogućava WPF (Windows Presentation Foundation) kao i varijable u kojima su *CarService* i *CarRepository* koji se koriste za umetanje ovisnosti (eng. Dependancy injection). Korisnička kontrola *UcHomeScreen* je klasa koja je otvorena za nadogradnju i trenutno nema nikakve funkcije i atribute. *UcImage* sadrži funkcije za prikaz ulazne slike, obrađene slike te funkcije za prikaz podataka dobivenih obradom slike. Ima atribut *service* i *plateUtil* koja sadrži funkcije za obradu slike. Klasa *UcManager* sadrži samo konstruktor i funkciju za dodavanje novog automobila u bazu i brisanje postojećeg. *AddCar* klasa sadrži sve potrebne funkcije za dodavanje novog zapisa u bazu. Iako je slična *UcImage* klasi, zbog nekih različitosti i daljnjeg razvoja ostavljena je kao zasebna klasa.



Slika 13: Dijagram klasa ServiceLayer

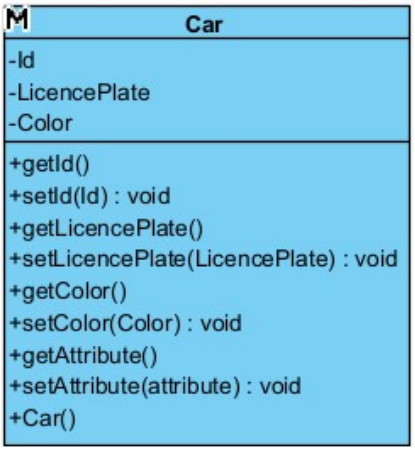
Na slici 13 može se proučiti dijagram klasa *ServiceLayer* i sadrži dvije klase. *CarSer-*

vice je klasa koja komunicira s repozitorijem. Sadrži kontruktor, funkciju *AddCar* za dodavanje, *GetAllCars* za dohvaćanje svih i *DeleteCar* za brisanje automobila. Druga klasa *LicensePlateUtil* je klasa koja sadrži funkcije za obradu slike. Sadrži funkcije za pretvorbu slike u sivu skalu, primjenu Gaussovog zamućivanja, detekciju rubova te dohvat dominantne boje. Osim toga, uključuje javne funkcije za dohvat teksta s registarske oznake, vraćanje dominantne boje i vraćanje obrađene slike.



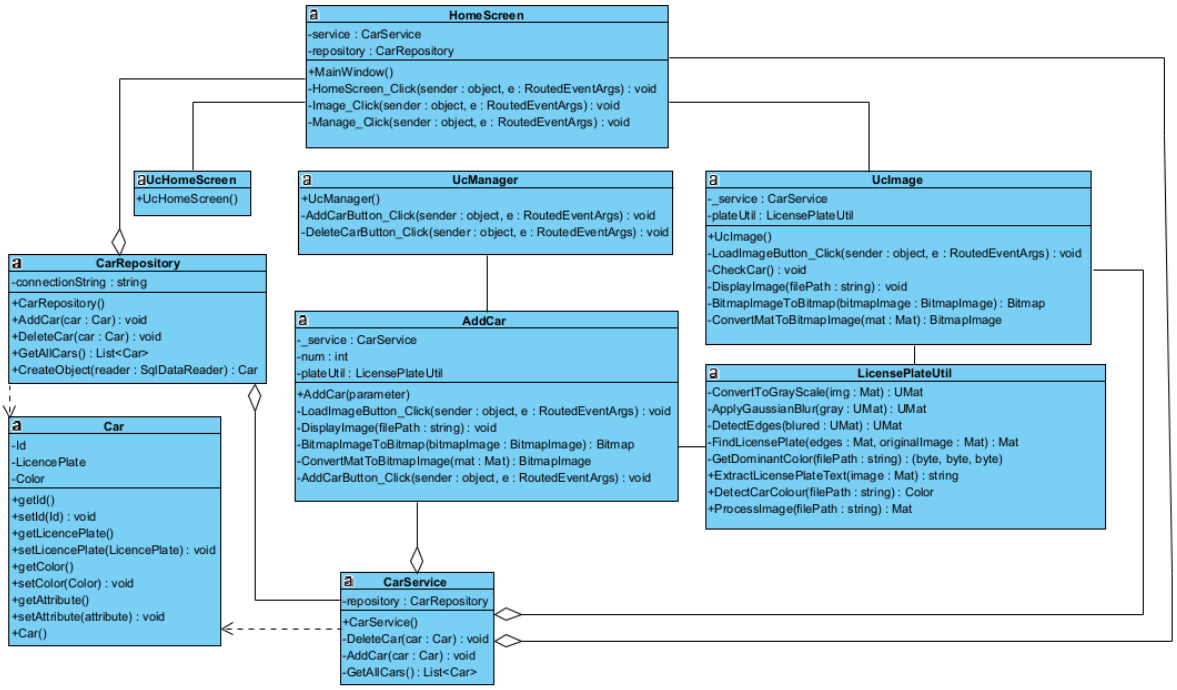
Slika 14: Dijagram klasa DataAccessLayer

DataAccessLayer je sloj za pristup bazi podataka. Ima samo jednu klasu *CarRepository*. Sadrži atribut *connectionString* u kojem su zapisani podaci za pristup bazi podataka. Sadrži konstruktor i funkcije *AddCar*, *DeleteCar*, *GetAllCars* te funkciju *CreateObject* koja služi za kreiranje objekta tipa *Car*.



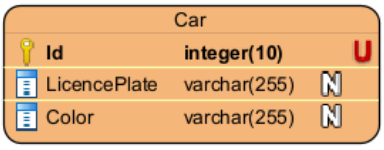
Slika 15: Dijagram klasa EntitiesLayer

EntitiesLayer sadržava sve entitete projekta. U trenutnoj verziji projekta postoji samo entitet *Car* s kojim se trenutno radi. Entitet ima sve atribute kakvi su i u bazi podataka te za svaki od atributa odgovarajući *Setter* i *Getter* te konstruktor.



Slika 16: Kompletan dijagram klasa

Na slici [16] može se vidjeti cijeli dijagram klasa za aplikaciju te međusobna povezanost klasa.



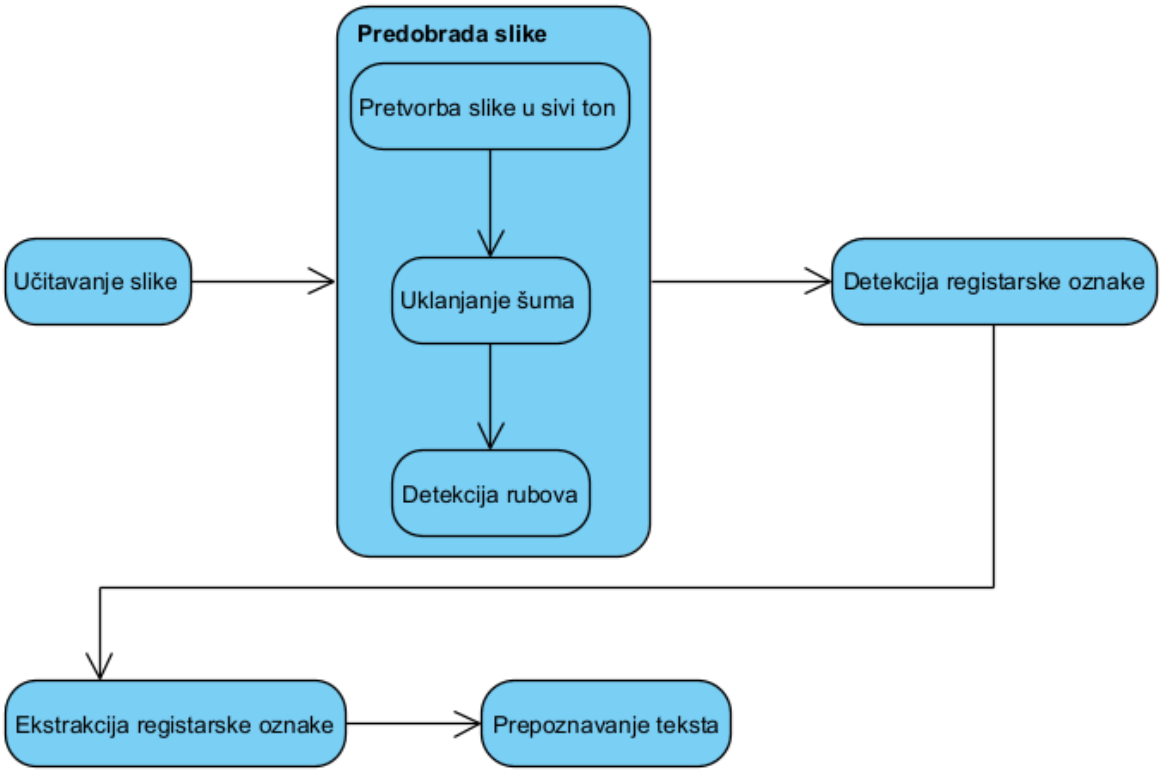
Slika 17: ERA dijagram

Model podataka ima jednu relaciju *Car* u koju se zapisuju podaci o automobilima.

4.3. Programsko rješenje

Glavni cilj ovog rada će biti opisan u ovom poglavlju, a to je pisanje programskog rješenja. Dijagram na slici 18 na najjednostavniji način opisuje osnovne korake za prepoznavanje registarskih oznaka automobila. Princip je jednostavan i primjenjiv za bilo koje druge slične sustave.

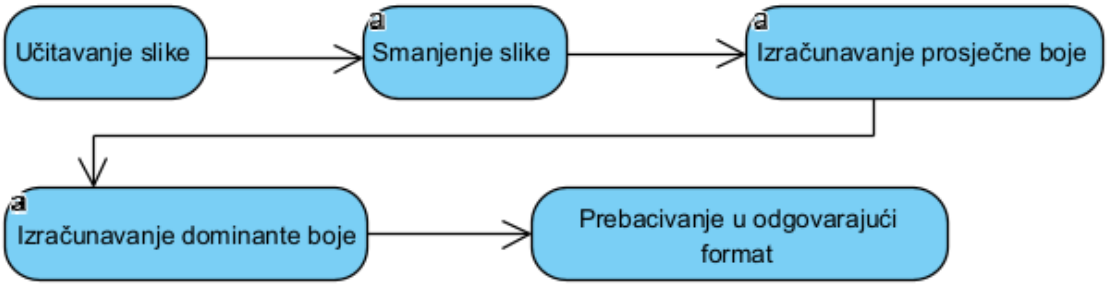
Za početak, potrebno je da korisnik prvo unese sliku koja se treba obraditi. Kad je slika unesena, počinje preprocesiranje slike koja ima tri osnovna koraka: Pretvorba slike u sivi ton, uklanjanje šuma i detekcija rubova. Pretvaranje slike iz boje u sivi ton (grayscale) smanjuje kompleksnost podataka. U sivim tonovima, svaki piksel sadrži samo informaciju o svjetlini, a ne o boji, što pojednostavljuje daljnju analizu. Uklanjanje šuma poboljšava kvalitetu slike i po-



Slika 18: Koraci za prepoznavanje registarskih oznaka

maže u boljoj detekciji značajki. Povećanje kontrasta pojačava razliku između tamnih i svijetlih dijelova slike. Povećanje kontrasta može poboljšati prepoznavanje rubova i značajki. Treći korak uključuje detekciju registarske oznake na slici. Ovo je ključni korak koji identificira i izdvaja područje slike koje sadrži registarsku oznaku. Koristi se prilično kompleksan algoritam koji će kasnije biti i opisan. Nakon što su identificirane konture koje predstavljaju registarske oznake, ta područja se izrezuju iz originalne slike. Ovo omogućava fokusiranje samo na relevantni dio slike. Za peti korak koristi se OCR alat, poput Tesseracta, za prepoznavanje teksta unutar izrezane registarske oznake. OCR alat analizira sliku i prepoznaje karaktere na temelju unaprijed naučenih modela.

Na slici 18 prikazan je osnovni model. Iako sam može pružiti zadovoljavajuće rezultate, također se može dodatno modificirati. Na primjer, programer može dodati rotiranje iz različitih kutova ili da za prepoznavanje teksta uklanja neželjene znakove.



Slika 19: Koraci za prepoznavanje boje

Slika 19 prikazuje osnovne korake za prepoznavanje dominantne boje automobila. Za početak slika se mora učitati u sustav. Kad je slika učitana, ukoliko je visoke rezolucije, proces bi bio vrlo spor, stoga je potrebno sliku smanjiti. Za smanjenje slike se koristi linearna interpolacija, ali nešto detaljnije o tome će biti kasnije rečeno. Kad je slika smanjena, slijedi izračun prosječnih boja za plavu, crvenu i zelenu u slici. Jednom kad su izračunate prosjeci tih boja, izračunava se još dominanta boja. Za kraj ih je još potrebno prebaciti u odgovarajući format.

U ovom dijelu će biti opisani ključni algoritmi koji su korišteni u radu. Čitavo programsko rješenje se može naći na [16].

4.3.1. Algoritam za detekciju registarskih oznaka

Funkcija 'ProcessImage' (vidi Algoritam 1) započinje učitavanjem slike automobila s diskovnog pogona u boji koristeći 'CvInvoke.Imread'. Učitavanje slike u boji važno je jer se kasnije u procesu koristi za detekciju boje automobila. Ovdje se koristi klasa 'Mat' iz Emgu.CV biblioteke koja predstavlja matricu slike. Nakon učitavanja slike, slika se pretvara u sivu skalu kako bi se olakšala daljnja obrada. Ovaj korak uklanja informacije o boji i zadržava samo intenzitet svjetlosti za svaki piksel, što pomaže u naglašavanju rubova i drugih važnih značajki.

Nakon pretvorbe u sivu skalu, na sliku se primjenjuje Gaussian blur, što je tehnika koja smanjuje šum na slici i izjednačava prijelaze između svjetlijih i tamnijih područja. Gaussian blur se koristi kako bi se smanjila vjerojatnost pogrešne detekcije rubova uzrokovanih šumom ili manjim nepravilnostima na slici.

Sljedeći korak uključuje detekciju rubova na zamućenoj slici. Detekcija rubova koristi Cannyjev algoritam koji identificira prijelaze između svjetlijih i tamnijih područja na slici. Ovi prijelazi često predstavljaju granice objekata unutar slike, poput rubova registarske tablice. Nakon što su rubovi detektirani, slika se analizira kako bi se pronašlo područje koje najvjerojatnije predstavlja registarsku tablicu. To se postiže pretragom kontura na slici i identifikacijom pravokutnih oblika koji odgovaraju veličini i obliku registarske tablice. Ako se pronađe odgovarajuće područje, izdvaja se kao zasebna slika koja se dalje koristi za prepoznavanje teksta.

Na kraju, izdvojeno područje koje sadrži registarsku tablicu vraća se kao povratna vrijednost funkcije.

Algoritam 1: Algoritam za detekciju registarskih oznaka

```
public Mat ProcessImage(string filePath) {
    using (Mat img = CvInvoke.Imread(filePath, ImreadModes.Color))
    using (UMat gray = ConvertToGrayScale(img))
    using (UMat blurred = ApplyGaussianBlur(gray))
    using (UMat edges = DetectEdges(blurred)) {
        Mat edgesMat = edges.ToImage().Mat;
        Mat licensePlateRegion = FindLicensePlate(edgesMat, img);
        return licensePlateRegion;
        }
}
```

4.3.2. Pretvorba slike u sivu skalu

Funkcija 'ConvertToGrayScale' (vidi Algoritam 2) ima za cilj pretvoriti sliku u sivu skalu, što je korak koji se često koristi u obradi slike kako bi se pojednostavila analiza. Funkcija uzima sliku u boji ('Mat img') kao ulazni parametar i vraća sliku u sivoj skali ('UMat gray').

Kada se slika učita u boji, svaki piksel sadrži tri vrijednosti koje predstavljaju crvenu, zelenu i plavu komponentu boje (BGR format). Pretvaranjem slike u sivu skalu, ove tri vrijednosti se kombiniraju u jednu vrijednost intenziteta svjetlosti, što pojednostavljuje obradu slike jer se smanjuje količina podataka za analizu.

U funkciji, kreira se nova instanca klase 'UMat', koja predstavlja matricu slike u sivoj skali. Zatim se koristi funkcija 'CvInvoke.CvtColor', koja omogućava pretvorbu slike iz jednog prostora boja u drugi. U ovom slučaju, koristi se pretvorba iz BGR (boja) u sivu skalu ('ColorConversion.Bgr2Gray').

Ovaj proces je ključan jer mnogi algoritmi za obradu slike, poput detekcije rubova, bolje funkcioniraju na slikama u sivoj skali gdje su prijelazi između svjetlijih i tamnijih područja jasnije izraženi. Nakon pretvorbe, funkcija vraća sliku u sivoj skali, koja se može dalje koristiti u narednim koracima obrade slike.

Algoritam 2: Algoritam za pretvorbu slike u sivu skalu

```
private UMat ConvertToGrayScale(Mat img) {
    var gray = new UMat();
    CvInvoke.CvtColor(img, gray, ColorConversion.Bgr2Gray);
    return gray;
}
```

4.3.3. Upotreba Gaussovog zamućivanja

Funkcija 'ApplyGaussianBlur' (vidi Algoritam 3) uzima sliku u sivoj skali kao ulazni parametar i primjenjuje Gaussovo zamućivanje, tehniku zamućivanja koja smanjuje šum na slici i izjednačava prijelaze između svjetlijih i tamnijih područja. Ulazni parametar je 'gray', slika u sivoj skali koja je već prošla kroz prethodni korak pretvorbe iz boje u sivu skalu. Zamućivanje se obavlja kreiranjem nove instance 'UMat' klase pod nazivom 'blurred', koja će sadržavati rezultat primjene Gaussian blur-a.

Gaussovo zamućivanje se temelji na matematičkoj funkciji poznatoj kao "Gaussova funkcija" koja određuje kako će pikseli unutar slike biti ponderirani prilikom zamućivanja. U ovoj funkciji, koristi se 'CvInvoke.GaussianBlur', metoda koja uzima tri glavna parametra: ulaznu sliku ('gray'), izlaznu sliku ('blurred'), te veličinu kernela ('new Size(3, 3)') i standardnu devijaciju ('1'). Veličina kernela od 3x3 znači da će svaki piksel biti zamijenjen ponderiranom prosječnom vrijednošću svojih susjeda u mreži veličine 3x3. Standardna devijacija određuje koliko će zamućenje biti intenzivno – veća vrijednost rezultira jačim zamućivanjem.

Nakon primjene Gaussovog zamućivanja, funkcija vraća rezultat u obliku slike 'blurred', koja je sada zamućena verzija ulazne slike. Ova zamućena slika se dalje koristi u narednim koracima obrade slike, kao što su detekcija rubova, jer smanjuje vjerojatnost pogrešne detekcije uzrokovane šumom ili manjim nepravilnostima na slici.

Algoritam 3: Algoritam za Gaussovo zamućivanje

```
private UMat ApplyGaussianBlur(UMat gray) {
    var blurred = new UMat();
    CvInvoke.GaussianBlur(gray, blurred, new Size(3, 3), 1);
    return blurred;
}
```

4.3.4. Detekcija rubova

Funkcija 'DetectEdges' (vidi Algoritam 4) detektira rubove na slici koja je prethodno zamućena Gaussovim zamućivanjem. Ova funkcija koristi Cannyjev algoritam, koji je jedan od najpoznatijih i najefikasnijih algoritama za detekciju rubova. Ulazni parametar funkcije je slika u sivoj skali, koja je već prošla kroz zamućivanje kako bi se smanjio šum i izjednačili prijelazi između različitih dijelova slike.

Unutar funkcije, kreira se nova instanca klase 'UMat', koja će sadržavati rezultat detekcije rubova. Za detekciju rubova koristi se metoda 'CvInvoke.Canny'. Ova metoda primjenjuje Cannyjev algoritam koji radi u nekoliko koraka: prvo se izračunava gradijent slike, koji identificira promjene u intenzitetu svjetlosti na slici, a zatim se primjenjuju pragovi koji odlučuju koji će pikseli biti klasificirani kao rubovi.

Dvije ključne vrijednosti u metodi 'CvInvoke.Canny' su pragovi: gornji prag postavljen

na 180.0 i donji prag postavljen na 120.0. Pikseli s gradijentima iznad gornjeg praga odmah se klasificiraju kao rubovi, dok se pikseli između donjeg i gornjeg praga klasificiraju kao rubovi samo ako su povezani s pikselima koji su već identificirani kao rubovi. Ova dvostruka pragova metoda omogućava algoritmu da bude osjetljiv na rubove dok ignorira šum.

Kao rezultat, funkcija vraća novu sliku koja sadrži samo detektirane rubove, prikazane kao svijetle linije na tamnoj pozadini. Ovi rubovi su ključni za daljnju obradu slike, kao što je pronalaženje kontura ili prepoznavanje objekata.

Algoritam 4: Algoritam za detekciju rubova

```
private UMat DetectEdges(UMat blurred) {
    var edges = new UMat();
    CvInvoke.Canny(blurred, edges, 180.0, 120.0);
    return edges;
}
```

4.3.5. Pronalaženje registarskih oznaka

Funkcija 'FindLicensePlate' (vidi Algoritam 5) je dizajnirana da na slici detektira registarsku tablicu, koristeći informacije o rubovima koji su već detektirani u prethodnom koraku. Ova funkcija je ključna u procesu prepoznavanja registarskih tablica jer pokušava pronaći pravokutni oblik koji odgovara karakterističnim dimenzijama i obliku registarske tablice.

Na početku funkcije, kreira se prazan popis 'boxList' koji će sadržavati pravokutnike ('RotatedRect'), svaki od njih predstavlja potencijalno područje na slici koje bi moglo biti registarska tablica.

Zatim se koristi metoda 'CvInvoke.FindContours' koja analizira sliku s detektiranim rubovima ('edges') kako bi identificirala konture, odnosno oblike na slici. Konture su sekvence točaka koje opisuju granice objekata na slici. Ovdje se konture pohranjuju u 'VectorOfVectorOfPoint' strukturu.

Nakon što su konture identificirane, funkcija prolazi kroz svaku konturu i koristi metodu 'CvInvoke.ApproxPolyDP' kako bi aproksimirala konturu poligonom sa što manje vrhova, s ciljem da se kontura pojednostavi na osnovne oblike. Ako aproksimirana kontura ima četiri vrha i površinu veću od određene vrijednosti (250 u ovom slučaju), smatra se da bi mogla predstavljati pravokutnik.

Nadalje, za svaki takav četverokut provjerava se je li zaista pravokutnik. To se radi uspoređivanjem unutarnjih kutova između susjednih vrhova. Ako su svi unutarnji kutovi približno pravokutni (između 80 i 100 stupnjeva), kontura se smatra pravokutnikom i dodaje se u 'boxList' kao potencijalni kandidat za registarsku tablicu. Na kraju funkcije, odabire se najveći pravokutnik te funkcija vraća izrezani dio originalne slike ('originalImage') koji odgovara pronađenom pravokutniku.

Algoritam 5: Algoritam za pronalaženje registarskih oznaka

```
private Mat FindLicensePlate(Mat edges, Mat originalImage) {
    var boxList = new List();
    using (var contours = new VectorOfVectorOfPoint()) {
        CvInvoke.FindContours(edges, contours, null, RetrType.List,
           ChainApproxMethod.ChainApproxSimple);
        for (int i = 0; i < contours.Size; i++) {
            using (var contour = contours[i])
            using (var approxContour = new VectorOfPoint()) {
                CvInvoke.ApproxPolyDP(contour, approxContour,
                   CvInvoke.ArcLength(contour, true) * 0.05, true);
                if (CvInvoke.ContourArea(approxContour, false) > 250
                   && approxContour.Size == 4) {
                    bool isRectangle = approxContour.ToArray().Select
                       ((p, j) => Math.Abs(
                        Emgu.CV.PointCollection.PolyLine(
                           approxContour.ToArray(), true)[(j + 1) %
                           4].GetExteriorAngleDegree(
                            Emgu.CV.PointCollection.PolyLine(
                               approxContour.ToArray(), true)[j]))).
                               All(angle => angle >= 80 && angle <=
                               100);
                    if (isRectangle) boxList.Add(CvInvoke.MinAreaRect
                       (approxContour));
                }
            }
        }
    }

    var largestBox = boxList.OrderByDescending(box => box.Size.Width
       * box.Size.Height).FirstOrDefault();
    return largestBox.Equals(default(RotatedRect)) ? null : new Mat(
       originalImage, CvInvoke.BoundingRectangle(largestBox.
       GetVertices().Select(Point.Round).ToArray()));
}
```

4.3.6. Čitanje teksta

Funkcija 'ExtractLicensePlateText' (vidi Algoritam 6) ima ključnu ulogu u procesu prepoznavanja teksta na registarskoj tablici. Ova funkcija koristi OCR (Optical Character Recognition) tehnologiju putem Tesseract engine-a kako bi prepoznala i izdvojila tekst iz slike registarske tablice.

Potrebno je za početak kreirati string varijablu koja će za svoju vrijednost poprimiti tekst koji se pročita sa registarske oznake.

Zatim se koristi TesseractEngine, instanca Tesseract OCR engine-a, koja je postavljena da koristi engleski jezik ("eng") i zadani način rada (EngineMode.Default). TesseractEngine se instancira unutar using bloka, što osigurava da će svi resursi biti pravilno oslobođeni nakon završetka rada s OCR-om.

Slika se zatim konvertira u Pix format (format koji Tesseract koristi) pomoću PixConverter.ToPix. Nakon toga, slika se procesira pomoću engine.Process, što pokreće OCR proces nad tom slikom.

Rezultat OCR-a je tekst koji se dobiva metodom page.GetText. Ovaj tekst se potom filtrira kako bi se uklonili svi znakovi koji nisu slova ili brojevi, koristeći Regex.Replace, čime se osigurava da ostanu samo relevantni znakovi registarske tablice.

Algoritam 6: Algoritam za čitanje teksta sa registarskih oznaka

```
public string ExtractLicensePlateText(Mat image) {
    string text;
    using (var engine = new TesseractEngine(@"./tessdata", "eng",
       EngineMode.Default)) {
        using (var pix = PixConverter.ToPix(image.ToBitmap()))
        using (var page = engine.Process(pix)) {
            text = page.GetText();
        }
    }
    return Regex.Replace(text, @"[^a-zA-Z0-9]", string.Empty);
}
```

4.3.7. Algoritam za detekciju boje

Funkcija 'DetectCarColor' (vidi Algoritam 7) uzima putanju do slike, koristi 'GetDominantColor' da bi odredila dominantnu boju, te vraća boju u 'Color' formatu.

'GetDominantColor' učitava sliku s diska u boji koristeći 'CvInvoke.Imread', koji se koristi za obrada slika u Emgu.CV. Slika se učitava u 'Mat' objekt. Zatim se slika smanjuje na 100x100 piksela pomoću 'CvInvoke.Resize', što smanjuje broj piksela i olakšava izračunavanje prosječne boje. Linearna interpolacija osigurava da su boje u smanjenoj slici što bliže originalu.

Nakon smanjenja slike, koristi se 'CvInvoke.Mean' za izračunavanje prosječne boje svih piksela. Ova metoda vraća 'MCvScalar', koji sadrži tri vrijednosti: 'V0' (plava komponenta), 'V1' (zelena komponenta) i 'V2' (crvena komponenta). Ove komponente su konvertirane u 'byte' tip i vraćene kao tuple.

Tuple su podatkovne strukture koje omogućavaju pohranu skupa različitih vrijednosti u jednoj jedinici. U ovom slučaju, tuple '(byte, byte, byte)' sadrži tri komponente boje. Elemen-

tima tuplea pristupa se pomoću automatski dodijeljenih imena poput 'Item1', 'Item2', itd., ili imenovanih svojstava ako se koriste u nekim programskim jezicima.

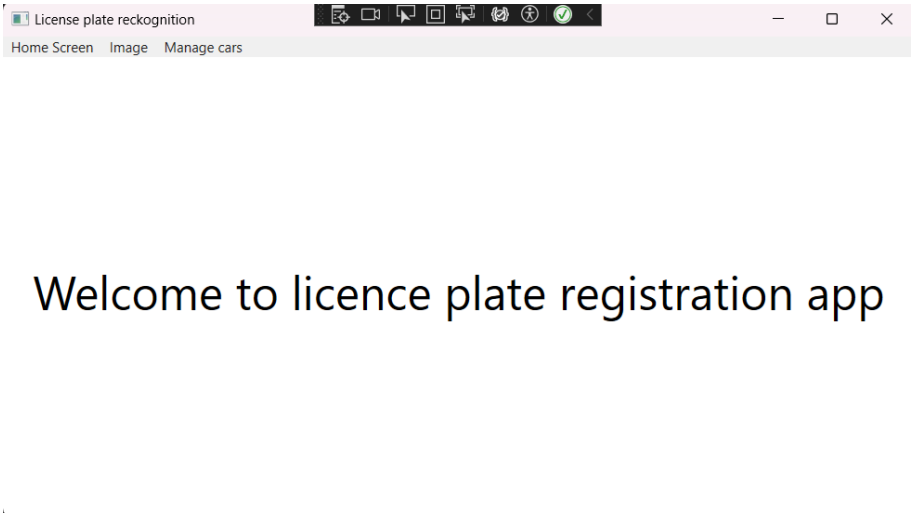
Funkcija 'DetectCarColor' prima ove tri komponente boje (plava, zelena, crvena), i koristi ih za stvaranje 'Color' objekta. Budući da 'Color.FromArgb' očekuje boje u RGB formatu (crvena, zelena, plava), komponente se prenose u obrnutom redoslijedu (crvena, zelena, plava). Rezultantni 'Color' objekt predstavlja dominantnu boju slike i vraća se kao rezultat funkcije.

Algoritam 7: Algoritam za detekciju boje

```
public Color DetectCarColor(string filePath) {
    var dominantColor = GetDominantColor(filePath);
    return Color.FromArgb(dominantColor.Item3, dominantColor.Item2,
       dominantColor.Item1);
}
private (byte, byte, byte) GetDominantColor(string filePath) {
    using (Mat image = CvInvoke.Imread(filePath, ImreadModes.Color))
       {
        Mat resizedImage = new Mat();
        CvInvoke.Resize(image, resizedImage, new Size(100, 100),
           interpolation: Inter.Linear);
        var meanScalar = CvInvoke.Mean(resizedImage);
        return ((byte)meanScalar.V0, (byte)meanScalar.V1, (byte)
           meanScalar.V2);
    }
}
```

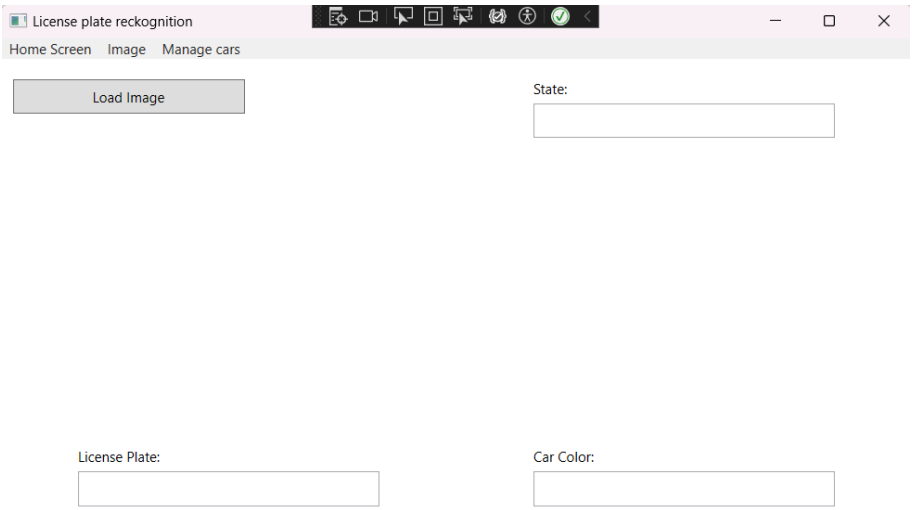
5. Demonstracija uporabe aplikacije

Kada se aplikacija pokrene, otvara se početni zaslon kao što je prikazano na slici 20. Ukoliko se želi pristupiti sustavu za prepoznavanje tablica, potrebno kliknuti na ’Image’.



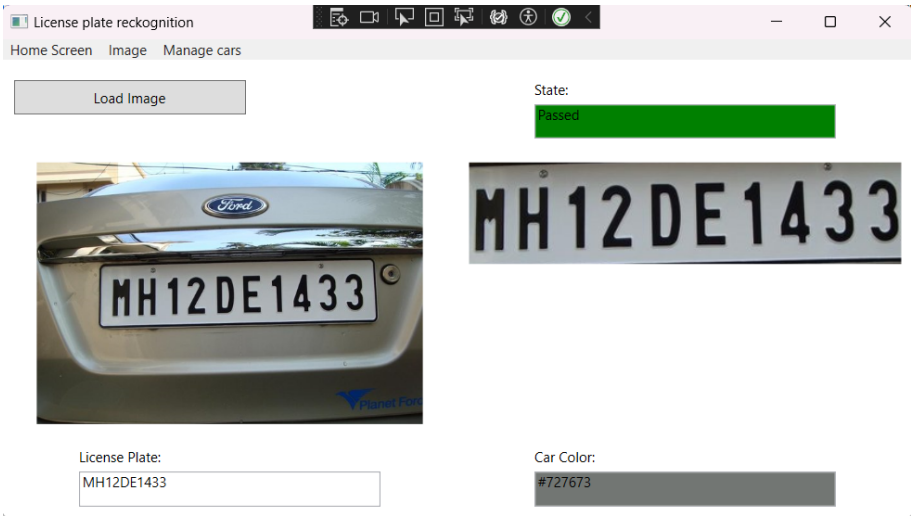
Slika 20: *Home screen*

Na sučelju 21 se pojavljuje gumb za učitavanje nove slike, ’State’ u kojem će pisati ’Passed’ ukoliko je automobil spremljen u bazi ili ’Reject’ ukoliko ne postoji zapis automobila. Na dnu se nalazi ’License plate’ gdje će biti zapisana registracija automobila te ’Car color’ gdje će kvadrat biti obojen bojom automobila i pisat će heksadekadski zapis boje.

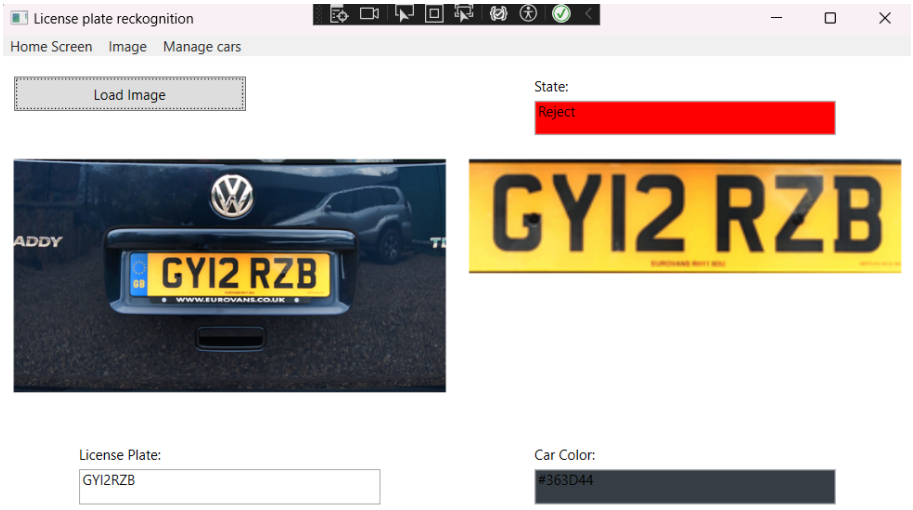


Slika 21: Prazno *Image* sučelje

Kada se klikne gumb ’Load image’ otvara se prozor za izbor slike i potrebno je odabrati sliku koju je potrebno obraditi. Kada se učita slika, obave se sve operacije vezane za procesiranje slike i dobije se stanje kao što je prikazano na slici 22 ukoliko zapis o automobil postoji. Ukoliko zapis ne postoji dobije se stanje kao na slici 23

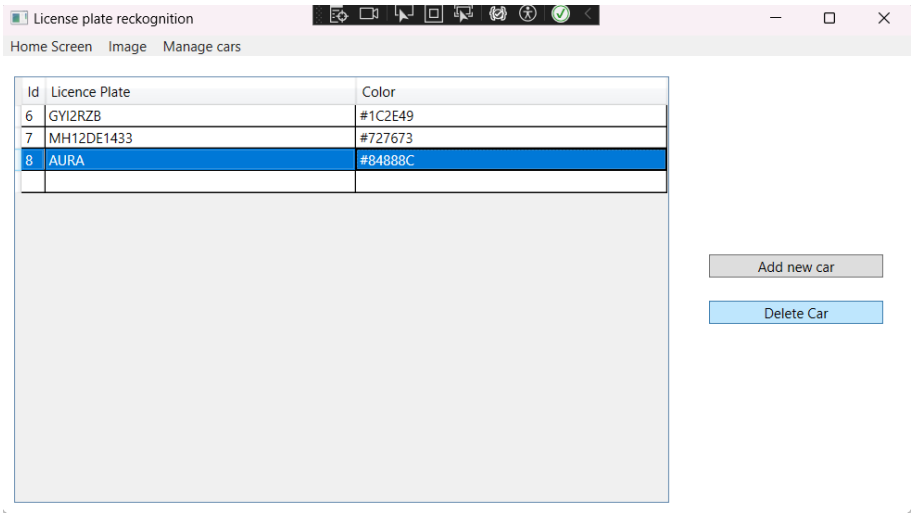


Slika 22: Odabrana slika s automobilom o kojem postoji zapis u bazi



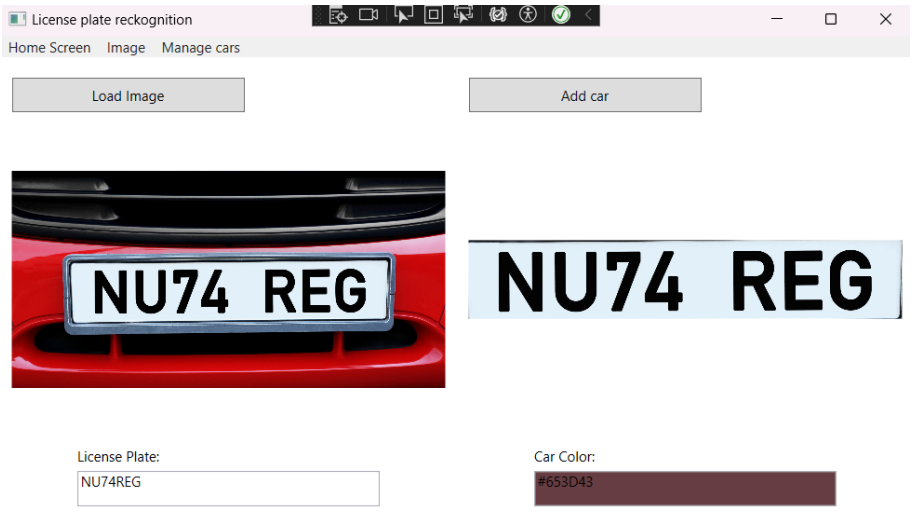
Slika 23: Odabrana slika s automobilom o kojem ne postoji zapis u bazi

Za dodavanje novog zapisa u bazu podataka, potrebno je se pozicionirati na sučelje o zapisima automobila pritiskom na gumb 'Manage cars'. Na sučelju za upravljanje automobilima koje je prikazano na slici 24 mogu se vidjeti svi zapisi koji postoje u bazi podataka, gumb 'Add new car' za dodavanje novog zapisa i gumb 'Delete car' za brisanje podataka odabranom automobilu.



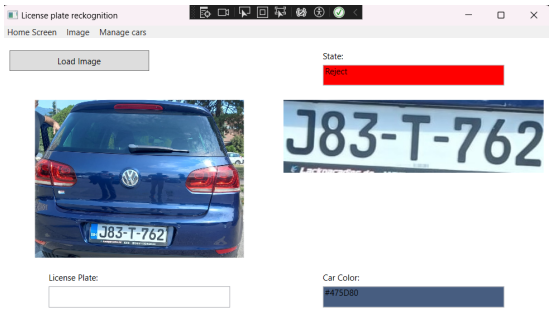
Slika 24: Prikaz sučelja *Manage cars*

Kada se pritisne gump ’Add car’ otvara se novo sučelje, prikazano na slici 25, koje je slično sučelju ’Image’, ali ovdje umjesto informacije o postojanosti zapisa o automobilu, postoji gumb ’Add car’ koji sprema zapis u bazu ukoliko već ne postoji.

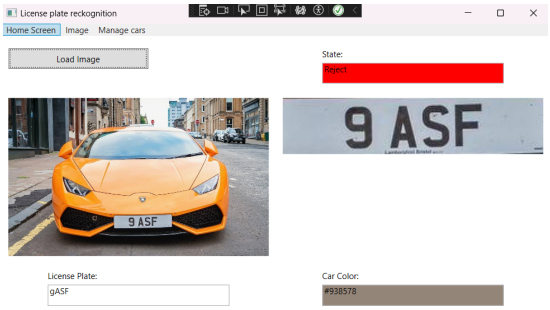


Slika 25: Dodavanje novog zapisa u bazu

Zbog određenih nedostataka, sustav zna ponekad zakazati u vezi čitanja znakova sa registarske oznake. Nekad jednostavno uopće ne može pročitati tekst ukoliko je loše kvalitete kao na slici 26 ili pogrešno pročita slova kao na slici 27.



Slika 26: Primjer lošeg rada Tesseracta



Slika 27: Primjer u kojem Tesseract zamjeni znakove

6. Zaključak

Cilj ovog rada je bio prikazati primjene tehnika procesiranja slika u svrhu prepoznavanja registarskih oznaka automobila. Uzete su neke jednostavne i temeljne tehnike koje se mogu koristiti u tražene svrhe. Bilo je potrebno prvo istražiti načine na koje programeri u praksi prave sustave za detekciju registarskih oznaka automobila. Jednom kad je shvaćen princip na koji se treba obaviti zadatak, bilo je potrebno odabrati odgovarajuće alate za to. Odabrani su OpenCV i Tesseract.

Cilj rada je ispunjen i uspješno su prikazane tehnike i načini procesiranja slike. U Teorijskom dijelu je opisano općenito kako se procesira slika te načini i alati koji su najprikladniji za to, a u praktičnom dijelu je prikazan način na koji su te tehnike primijenjene u C#-u za prepoznavanje registarskih oznaka.

EmguCV me je jako pozitivno iznenadio. Već pri istraživanju alata, Emgu na internet nudi jako dobru i korisnu dokumentaciju o svojim alatima, tako da samo izučavanje nije bilo problem. Početniku je malo teže zbog velikog broja funkcionalnosti, ali i taj problem se brzo savlada. OpenCV kao i Emgu nudi dosta dokumentacije koja je odlična u početnim fazama istraživanja. Što se tiče konkretnog problema kojim se završni rad bavi, OpenCV pruža prilično dobre rezultate, ali zna biti pogrešaka. Na primjer, senzor od automobila, neki list drveta ili bilo koji drugi manji predmet je prepoznao kao pravokutnik. Sa većim predmetima nije imao problema i samo je registarsku oznaku prepoznavao kao pravokutnik kao veći predmet. Rješenje je bilo prilično jednostavno, iz liste pravokutnika koji prepoznao, uzeo sam najveći koji je ujedno i registracija. Još jedna pozitivna stvar za EmguCV je ta što sadrži i funkcije koje su pomogle u detekciji boje automobila. Uz to sve, na službenoj stranici EmguCV koja se nalazi na [9], može se naći dosta korisnih algoritama i tutorijala od kojih su neki korišteni i za ovaj rad.

Tesseract je bio već znatno lošiji po mom iskustvu. Slike koje nisu blizu i izrazito visoke kvalitete često su nečitljive Tesseractu. Iako OpenCV obavi odličan posao sa ekstrakcijom tablice, Tesseract, uglavnom, to ne može pročitati. Na slici 26 može se vidjeti loš posao koji je Tesseract odradio. Tablica je lijepo uslikana, čitka, ali Tesseract nije uspio pročitati tekst. Također, vrlo često zamjeni slova ili brojeve. Na primjer, broj ’9’ zamjeni sa slovom ’g’ kao što je prikazano na primjeru 27 .

Nakon isprobanih alata, EmguCV bih zasigurno ponovo iskoristio i ne planiram mijenjati ni na sljedećim verzijama aplikacije. S druge strane Tesseract se zasigurno mora promijeniti jer rampa mora raditi sa točnošću preko 99%. Preporučujem zamjenu Tesseracta za neki drugi model ili treniranje vlastitog modela.

Za ovakav problem, na internetu se može naći mnogo izvora, referenci i algoritama za python programski jezik koji jest malo više namijenjen za ovu svrhu. Iako je takva situacija, nije bilo većih problema za C#. EmguCV nudi odličnu dokumentaciju uz koju i ne treba previše istraživanja sa strane. Sa trenutačnim iskustvom, ne bih projekt u skorije vrijeme prebacivao na Python jer mi je na ovaj način lakše.

Aplikacija je vrlo korisna i ima vrlo široku primjenu. Za komercijalnu upotrebu, potrebno

je nabaviti potreban hardver i dodatno razviti aplikaciju. Potrebno je dizajnirati da se više ne ubacuju slike, već da se hvataju kadrovi iz snimke koju kamera snima. Kada se dodatno sredi, sustav bi se mogao koristiti za parkirna mjesta, naplatne kućice, privatne, javne garaže, policijske kamere za snimanje neregistriranih automobila i slično.

Popis literature

[1] CloudFactory. „The Beginner's Guide to Computer Vision." Pristupljeno: 29. kolovoza 2024. (2020.), adresa: https://www.cloudfactory.com/computer-vision-guide.

[2] M. Gayashan, „Understanding Computer Vision: A Comprehensive Guide," *Medium*, 2024., Pristupljeno: 29. kolovoza 2024. adresa: https://mgayashan.medium.com/understanding-computer-vision-a-comprehensive-guide-308ccb4e0f04.

[3] A. Sharda, „Image Filters: Gaussian Blur," *Medium*, 2019., Pristupljeno: 31. kolovoza 2024. adresa: https://aryamansharda.medium.com/image-filters-gaussian-blur-eb36db6781b1.

[4] MathWorks. „imgaussfilt - MathWorks Documentation." Pristupljeno: 2024-08-14. (2024.), adresa: https://www.mathworks.com/help/images/ref/imgaussfilt.html.

[5] Wikipedia contributors, *Canny edge detector — Wikipedia, The Free Encyclopedia*, Pristupljeno: 31. kolovoza 2024, 2024. adresa: https://en-m-wikipedia-org.translate.goog/wiki/Canny_edge_detector?_x_tr_sl=en&_x_tr_tl=hr&_x_tr_hl=hr&_x_tr_pto=sc.

[6] Keymakr. „Semantic Segmentation vs Object Detection: Understanding the Differences." Pristupljeno: 31. kolovoza 2024. (2024.), adresa: https://keymakr.com/blog/semantic-segmentation-vs-object-detection-understanding-the-differences/.

[7] *OpenCV Documentation*, Pristupljeno: 2024-08-02. adresa: https://docs.opencv.org/4.10.0/index.html.

[8] Roboflow, *What is OpenCV?* https://blog.roboflow.com/what-is-opencv/, Pristupljeno: 2024-08-14, 2024.

[9] *Emgu CV Documentation*, Pristupljeno: 2024-08-02. adresa: http://www.emgu.com/wiki/index.php/Main_Page.

[10] *Emgu CV Github documentation*, Pristupljeno: 2024-08-02. adresa: https://github.com/emgucv.

[11] Unity Asset Store, *Publisher Page for Unity Asset Store*, Pristupljeno: 29. kolovoza 2024, 2024. adresa: https://assetstore.unity.com/publishers/8462?srsltid=AfmBOoqWQgwKvsXDm2GaMbev1vKt502W1HB6mhaHgHPdOQp7VfuFl5Tt.

[12] EmguCV Project, *EmguCV Releases*, Pristupljeno: 29. kolovoza 2024, 2024. adresa: https://sourceforge.net/projects/emgucv/files/emgucv/.

[13] *Tesseract OCR Documentation - Input formats*, Pristupljeno: 2024-08-07. adresa: https://tesseract-ocr.github.io/tessdoc/InputFormats.html.

[14] W. contributors, *Tesseract (software) — Wikipedia, The Free Encyclopedia*, Pristupljeno: 29. kolovoza 2024, 2024. adresa: https://en.wikipedia.org/wiki/Tesseract_(software).

[15] *Tesseract Documentation*, Pristupljeno: 2024-08-04. adresa: https://github.com/tesseract-ocr/tesseract.

[16] *Frano Šimić - završni rad*, Pristupljeno: 2024-08-07. adresa: https://github.com/fsimic21/zavrsni.git.

Popis slika

Popis tablica

Popis algoritama