Detekcija i identifikacija registarskih tablica na automobilima

Frajtag, Ines

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Ines Frajtag

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

doc. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Ines Frajtag

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru doc. sc. dr. Tomislavu Stipančiću na pruženoj pomoći i savjetima tijekom stvaranja ovog rada. Posebnu zahvalu posvećujem roditeljima i bratu koji su mi bili bezuvjetna podrška i pomoć.

Ines Frajtag

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Ines Frajtag** JMBAG: **0035216582**

Naslov rada na hrvatskom jeziku: **Detekcija i identifikacija registarskih tablica na automobilima**

Naslov rada na engleskom jeziku: **Detection and identification of licence plates on cars**

Opis zadatka:

Računalni modeli temeljeni na algoritmima umjetne inteligencije omogućavaju automatsku detekciju i prepoznavanje znakova na registarskim oznakama na vozilima. Da bi to bilo ostvareno potrebno je koristiti kombinaciju različitih metoda uključujući one koje omogućuju pronalazak registarskih tablica unutar scene te onih koje omogućavaju prepoznavanje znakova.

U okviru završnog rada je potrebno:

- proučiti metode i algoritme za prepoznavanje znakova te lokalizaciju objekata unutar scene koristeći vizijske sustave,
- primijeniti odgovarajuće metode i algoritme te razviti programsku podršku za identifikaciju vozila, lokalizaciju registarskih oznaka te prepoznavanje znakova,
- prilagoditi i razviti programsku podršku koja omogućuje spremanje očitanih podataka na lokalno računalo.

Razvijeno programsko rješenje je potrebno temeljiti na *OpenCV* biblioteci implementiranoj kroz Python programski jezik. Model je potrebno eksperimentalno evaluirati koristeći vizijski sustav. U radu je potrebno navesti korištenu literaturu te eventualno dobivenu pomoć.

Zadatak zadan:
9. 5. 2022.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

SADRŽAJ I
POPIS SLIKA II
SAŽETAK III
SUMMARY IV
1. UVOD 1
2. RAČUNALNI VID 2
2.1. Zadaci i karakteristike 3
2.2. Strojno učenje 3
3. NEURONSKE MREŽE I ANPR 5
3.1. Faze prepoznavanja korištenjem neuronskih mreža 5
3.2. ANPR 7
3.2.1. Rad ANPR sustava 8
4. OPENCV 10
5. KORIŠTENJE OPENCV BIBLIOTEKE ZA DETEKCIJU I IDENTIFIKACIJU REGISTARSKIH TABLICA 11
5.1. Detekcija i smjer kretanja automobila u realnom vremenu 11
5.2. Detekcija i identifikacija registarskih tablica 14
5.2.1. Pretvorba u sive nijanse – Grayscale 15
5.2.2. Otkrivanje i pronalazak kontura – Canny detektor rubova 16
5.2.3. Prepoznavanje znamenaka – segmentacija 19
5.2.4. Spremanje podataka 21
5.2.5. Rezultati prepoznavanja 21
6. KRITIČKO RAZMIŠLJANJE 27
7. ZAKLJUČAK 29
LITERATURA 30
PRILOZI 32

POPIS SLIKA

Slika 1. Primjer korištenja Canny detektor rubova 6
Slika 2 . Shematski prikaz rada ANPR-a 7
Slika 3. Faze ANPR sustava [10] 8
Slika 4. Sustav koji je predložen za optimalan rad ANPR-a 9
Slika 5. Dio koda odgovoran za ubacivanje karnela i pronalazak kontura 12
Slika 6. Praćenje smjera kretanja automobila 13
Slika 7. Prikaz rada aplikacije za detekciju i smjer kretanja 13
Slika 8. Rad programa i nakon prolaska određenog broja automobila 14
Slika 9. Originalna slika za vršenje testiranja 15
Slika 10. Primjer *RGB Grayscale* metode 15
Slika 11. Dio koda koji pretvara sliku u sive nijanse 16
Slika 12. Slika u sivim nijansama – grayscale 16
Slika 13. Kod vezan za Canny detektor rubova 17
Slika 14. Implementacija Canny detektora rubova 17
Slika 15. Dio koda za pronalazak kontura i njihovo sortiranje 18
Slika 16. Uspoređivanje kontura i pronalazak tražene sa 4 ruba 18
Slika 17. Pronalazak točne konture 18
Slika 18. Kod za izdvajanje prepoznate registarske tablice 19
Slika 19. Prepoznati i izdvojeni dio slike 19
Slika 20. Prepoznavanje znakova OCR sustavom 19
Slika 21. Dio koda za ispisivanje prepoznate tablice na slici 20
Slika 22. Uokvirena registarska tablica nakon identifikacije 20
Slika 23. Kod za ispis u terminalu Pythona 20
Slika 24. Kod za pohranu iščitanih podataka 21
Slika 25. Prikaz popunjavanja tablica u Excelu putem Pythona 21
Slika 26. Uspješno prepoznate američke registarske tablice 22
Slika 27. Uspješno prepoznate hrvatske registarske tablice 23
Slika 28. Uspješno prepoznavanje tablica 24
Slika 29. Kraj programa u ovom stupnju zbog nemogućnosti prepoznavanja rubova tablice . 25
Slika 30. Početna fotografija za testiranje 25
Slika 31. Ispis djelomično prepoznate tablice 26
Slika 32. Dio koda koji je potrebno mijenjati 27

SAŽETAK

U ovom radu analizirat će se i opisati na koji način se vrši detekcija automobila u stvarnom vremenu, ali isto tako kako detektiramo i identificiramo registarske tablice na automobilima. OpenCV je biblioteka na kojoj će se temeljiti algoritmi u ovom radu, prilikom prepoznavanja rubova i kontura bitan je Canny detektor rubova, a samo iščitavanje znamenaka sa prepoznate registarske tablice vrši se optičkim sustavom za raspoznavanje znamenaka (OCR). Kako bi se spremili rezultati sa očitane tablice, kodom će se spojiti Python i Microsoft Excel i generirati tablicu sa rezultatima.

Ključne riječi: OpenCV, registarske tablice, algoritam, konture, Canny, OCR, detektiranje

SUMMARY

In this work, we will analyze and describe how car detection is done in real time, but also how we detect and idnetify license plates on cars. OpenCv is the library on which the algorithms in this work will be based. The Canny edge detector is essential when recognizing edges and contours. And the character reading from the recognized license plate is performed by an optical character recognitin system (OCR). In order to save the results from the license plate the code will connect Python and Microsoft Excel and generate an Excel file with results in it.

Key words: OpenCV, license plate, algorithm, contour, Canny, OCR, detection

1. UVOD

U današnje vrijeme kada su svi ljudi izuzetno užurbani i kada se broj automobila povećao jako je bitno paziti na sigurnost na cesti i na mjestima gdje se vozila ostavljaju, npr. parkirališta. Sama sigurnost prati se kamerama, a time je sve više rasprostranjena i činjenica da se lako može njima detektirati i identificirati registarska tablica, a njome se otkriva i sam vozač, tj. osoba kojoj pripada vozilo. Detekcija i identifikacija registarskih tablica dolazi iz područja umjetne inteligencije, točnije, iz područja računalnog vida koji se sve više primjenjuje kako bi se olakšao nadzor na prometnicama. Samom umjetnom inteligencijom (AI) provodi se proces automatizacije gdje se registarske tablice sada prepoznaju putem računala tako da se iz snimke prepoznaje prostor registarske tablice, a određenim algoritmima i metodama se očitavaju podaci. Računalnim vidom računalo vidi i razaznaje na slikama i videozapisima objekte koji su potrebni za daljnji tijek. U ovom radu koristit će se OpenCV biblioteka programskih funkcija koja se može koristiti kod različitih programskih jezika, a ovdje će se koristiti Python. Korištenjem OpenCV biblioteke računalni vid se koristi u stvarnom ili kraćem vremenu. Objasniti način na koje se provodi detekcija i identifikacija registarskih tablica te koje su metode i algoritmi korišteni je cilj ovog rada.

2. RAČUNALNI VID

Računalni vid (CV – Computer Vision) područje je umjetne inteligencije koje nastoji razviti sposobnost razumnog iščitavanja informacija iz digitalnih fotografija i videozapisa. Sam računalni vid mogao bi biti zaseban ogranak pod umjetnom inteligencijom i strojnim učenjem jer se koristi općim algoritmima i specijaliziranim metodama za prepoznavanje objekata. Razlika između računalnog vida i strojnog vida bila bi u tome što strojni vid funkcionira kao proces kombiniranja analize slike s drugim metodama sa ciljem što bolje automatizacije i robotizacije procesa, a sam računalni vid koristi temeljne tehnologije za automatsku analizu slike. Sam računalni vid uči računala o pozicijama i kretanju objekata na slici, a istovremeno ih raspoređuje i u klase. Kako bi se prepoznali uzorci ili objekti potrebne su razne metode koje se svakodnevno moraju usavršavati i prilagođavati novonastalim tehnologijama. Sa svakim je unaprjeđenjem također poželjno da se ono testira na više primjera, tako je i u računalnom vidu i metodama koje su u njega uključene uvijek testirano što je više moguće različitih varijacija pojedinih metoda i primjera.

Najranija istraživanja u području računalnog vida provodila su se 1950-ih za otkrivanje krajnjih granica objekata i razvrstavanja najjednostavnijih objekata u kategorije. Već u 1970-ima je računalni vid predstavljen ljudima i raširen u komercijalnoj uporabi gdje je pomoću njega bilo moguće interpretirati tipkani ili rukom pisani tekst pomoću optičkog prepoznavanja znakova. Nakon što je Internet postigao veliku popularnost u 1990-ima, naglo se povećala i količina vizualnih podataka, koja raste i dalje, povećao se također i broj algoritama i metoda kojima se omogućuje prepoznavanje određenih osoba ili objekata na slikama ili videozapisima, a također se omogućuje i razvrstavanje po kategorijama. Sa povećanjem računalnog vida povećao se i broj novih hardvera i algoritama, a samim time je i točnost prepoznavanja dostigla skoro savršenstvo od 100%.

Da se postigne savršenstvo u prepoznavanju objekata na slici, računalni vid trebao bi procesirati podatke na isti način kao i ljudski mozak. Ta spoznaja i dalje nije dokazana. Istraživanja o funkcioniranju ljudskog mozga temelje se samo na teorijama koje nije moguće sa točnošću definirati, a samim time postoje samo neke podudarnosti između ljudskog i računalnog vida: senzori kamere su ljudsko oko, fotografija/videozapis se obrađuje preko algoritama, a isto se sve viđeno ljudskim okom obrađuje u centrima za vid. Na kraju je krajnji rezultat jednak, računalnim vidom se interpretira slika, a ljudskim vidom percipira stvarnost.

2.1. Zadaci i karakteristike

Najvažniji zadatak i karakteristika računalnog vida je detekcija objekata. Samom detekcijom smatra se klasifikacija objekata i određivanje položaja, u ovom radu se usredotočimo na digitalne slike i videozapise. U ovom radu gledat će se metode kojima se raspoznaju znamenke na slikama, a isto tako promatrat ćemo i granice u kojima se nalaze znamenke. Da se razumije što bolje sadržaj na slikama on se mora izdvojiti i biti jednako uočljiv na slici kao i čovjeku u stvarnosti jer se time olakšava računalu da pomoću algoritma analizira sadržaj slike. U pravom sustavu vida on mora u svakom trenutku razaznati iz beskonačnog broja scena ono što se traži, a to ujedno mora biti i smisleno.

Napredak računalnog vida vidljiv je u svim područjima, ali najistaknutiji je ipak u području optičkog prepoznavanja znakova, navigaciji za robote, medicinskom snimanju, izgradnji 3D modela, automobilskoj sigurnosti, nadzoru te snimanju pokreta i prepoznavanju otisaka prstiju i biometriji. Tipične funkcije koje se nalaze u svim sustavima računalnog vida su: dohvaćanje slike, preuređivanje i obrada, ekstrakcija značajki, detekcija, obrada na visokoj razini i donošenje odluka. [1] Dohvaćanje slike je proces u kojem se digitalna slika obrađuje senzorima, prilikom obrađivanja slike se provjeravaju početni uvjeti kako bi se mogli izvlačiti podaci. Iz slikovnih podataka se zatim izvlače sve značajke, tzv. ekstrakcijom, a segmentacijom ili detekcijom se smanjuje potreban prostor koji se gleda i usredotočuje se na samo jedan prostor (okvir). Daljnjom obradom se taj prostor pretvara u točkice i dijeli na skupove kako bi se prepoznali znakovi, točnije značajke koje nas zanimaju, a samim donošenjem odluke odlučujemo jesmo li zadovoljni postignutim programom ili nismo.

2.2. Strojno učenje

Lakše rješavanje problema u računalnom vidu dalo je strojno učenje u kojemu se ne treba ručno kodirati svako pravilo za aplikaciju već su umjesto toga napravljene manje značajke kojima je lakše pojedinačno pronalaziti rješenja te sve zatim sklopiti zajedno. Linearna regresija, logička regresija i stabla odluke dio su algoritma za statističko strojno učenje kojima se uzimaju uzorci i klasificiraju slike te otkrivaju predmeti na njima. Podvrsta strojnog učenja je duboko učenje koje pruža malo drugačiji pristup. Ono se oslanja na neuronske mreže. U neuronskim mrežama

se opet smatra da se za svaki problem kroz primjer i opću funkciju može naći rješenje. Kao iznimno učinkovita metoda, dubokim učenjem se stvaraju dobri algoritmi u kojima se sve svodi na prikupljanje velikog broja podataka i njihovom stalnom testiranju prilikom najmanjih izmjena.

Upravo zbog toga je detekcija lica ljudi jedan od najpoznatijih primjera dubokog učenja. U tom primjeru je vidljivo da su odabrani unaprijed napravljeni algoritmi, a testirani su na velikom broju ljudskih lica te je kasnije neuronska mreža mogla sama, bez uputa, detektirati lica.

Primjer detekcije lica jedan je samo primjer, ali i sam naziv ovog rada upućuje na to da će se koristiti duboko učenje. [3] Na velikom broju registarskih tablica (koje su posljedica sve većeg broja vozila) se uz pomoć već unaprijed napravljenih algoritama izdvajaju znakovi, sortiraju i identificiraju se sve znamenke na njima. Taj postupak obavlja se na slikama pomoću OCR (optičko prepoznavanje znakova) koncepta, a prilikom toga u proces je uključeno:

- lokaliziranje registarske tablice – najvažniji korak, određuje se položaj pločice, ulazni podatak je slika vozila, a izlazni registarska tablica
- segmentacija znakova – izvlače se znakovi (slova i brojevi) i razvrstavaju u pojedinačne slike
- prepoznavanje znakova – svaki izdvojeni znak se prepoznaje

Upravo zbog ovakvog pristupa je duboko strojno učenje lakše, brže se implementira i razvija. [12] Drugačijim procesima, npr. podudaranjem predložaka, mogao bi se dobiti jednako željen rezultat, ali točnost kakva je sa strojnim učenjem tada nije zagarantirana.

3. NEURONSKE MREŽE I ANPR

Većina učinkovitih sustava za automatsko prepoznavanje registarskih tablica temelji se na umjetnim neuronskim mrežama (ANN). To je sustav koji se svakim danom sve više koristi zbog stalnog broja povećanja vozila, a time ujedno i broja registarskih oznaka. Kao što je već i prije navedeno, prepoznavanje ragistarskih tablica na vozilima važno je za sigurnost i čovjeka i vozila, a njegove glavne karakteristike su: lokaliziranje tablice, segmentiranje znakova i prepoznavanje znakova.

3.1. Faze prepoznavanja korištenjem neuronskih mreža

Počinjemo sa lokalizacijom, pronalaskom, registarskih tablica. Većina tablica pravokutnog je oblika te se sastoji od bijele pozadine i crnih slova. Zbog velikog kontrasta između te dvije boje lakše je zamijetiti prijelazne točke i traženje svesti na manji dio. Detektorom rubova Canny bi se moglo utvrditi gdje je najveća razlika u bojama, točnije, gdje je razlika između crne i bijele.

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\left(i-(k+1)\right)^2 + \left(j-(k+1)\right)^2}{2\sigma^2}\right); 1 \le i,j \le (2k+1)$$

(1)

Buka na slici jedna je od najčešćih negativnih pojava pri otkrivanju rubova, a nju se može spriječiti korištenjem prvog derivata Gaussovog filtera za izglađivanje. Jednadžba (1) pokazuje nam izraz koji je definiran kao Gaussov filter, točnije kao Gaussova jezgra koja utječe na performanse detektora. Ako je veličina jezgre veća onda je osjetljivost detektora na buku niža, ali i pogreška lokalizacije za otkrivanje ruba je veća. Kako bi se nakon izglađivanja pronašao pravi rub uzima se gradijent slike. Jednadžbe (2) i (3) međusobnom povezanošću služe za određivanje gradijenta slike i smjer ruba.

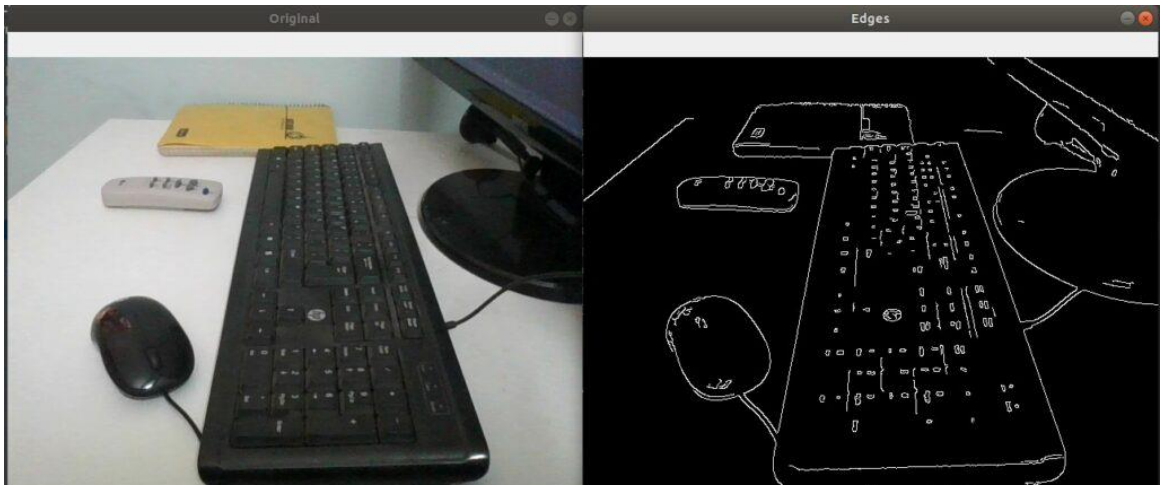
$$\boldsymbol{G = \sqrt{G_x^2 + G_y^2}}$$

(2)

$$\boldsymbol{\theta = atan2(G_y, G_x)}$$

(3)

Slike se mogu usmjeriti u više smjerova, a upravo Canny algoritam koristi filtere za vertikalan, horizontalan i dva dijagonalna smjera. Pomoću gradijenta slike traže se mjesta sa najvećom promjenom vrijednosti intenziteta. To se postiže prolaskom filtera matrice 3x3 kroz rezultate čvrstoće ruba i smjera gradijenta. Međutim, oko nekih rubova se i dalje mogu naći promjene te se uspoređivanjem vrijednosti gradijenta oni definiraju kao slabi, a ujedno ih smatramo tada i lažnim rubovima. Samo oni dovoljno jaki rubovi, sa velikom vrijednosti gradijenta, ostaju kao pravi i tvore rubove registarskih pločica. Tehnika kojom se koristimo za suzbijanje slabijih rubova zove se prorjeđivanje rubova. Kao što je i napisano, uspoređujemo trenutni piksel na slici sa jačinom ruba piksela u pozitivnom ili negativnom smjeru gradijenta. Sačuva se uvijek ona vrijednost koja je veća. Korak kojim se prepoznaju rubovi je ustvari kontrast slike koji sliku čini oštrijom. Kada se loša slika poboljša kontrastom na ovakav način to je tehnika izjednačavanja histogramima.



Slika 1. Primjer korištenja Canny detektor rubova

Canny algoritam je prilagodljiv za različite situacije. Moguće je podesiti neke njegove parametre kako bi se povećala učinkovitost. Neki od mogućih promjena mogli bi biti veličina Gaussovog filtera i pragovi. Zbog veličine Gaussovog filtera on radi velika zamućenja i u tom trenu se ne otkrivaju sve linije koje bi bile potrebne za pronalazak rubova. Pragovi su stalno prisutan problem u detekciji. Postavljanjem pragova na previsoku ili prenisku vrijednost dovodi do propuštanja važnih informacija ili do dovođenja nebitnih informacija.

3.2. ANPR

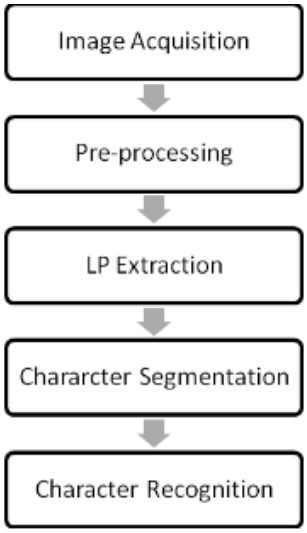
ANPR ili automatsko prepoznavanje automobilskih tablica istraživačko je polje u sklopu računalnog vida koje se bavi prepoznavanje uzoraka, obrađivanjem slike, umjetnom inteligencijom i povezivanjem računala, automobila i ljudi.

Slika 2 . Shematski prikaz rada ANPR-a

Ljudima je često taj sustav poznat i pod drugim nazivima:

- ALPR (Automatic license-plate recognition) – automatsko prepoznavanje registarske pločice
- ALPR (Automatic license-plate reader) – automatski čitač registarskih tablica
- AVI (Automatic vehicle identification) – automatska identifikacija vozila
- CPR (Car plate recognition) – prepoznavanje automobilskih ploča

Glavni zadaci u automatskom prepoznavanju automobilskih tablica su: obrada slike, lokalizacija pločica (registracijskih tablica), segmentacija znakova i prepoznavanje znakova. Ovaj sustav ujedno služi i kao baza podataka u koje je moguće spremati snimljene slike i tekst sa registarskih tablica. Da bi sustav funkcionirao on mora imati različite kutove, udaljenosti, uvjete osvjetljenja, ljestvice i razlučivosti. Također svaki sustav ima i faze kroz koje mora proći da bi se dobio odgovarajući rezultat. Na slici (Slika 3.) prikazan je redoslijed koji se prati kod rada ANPR sustava. Početak rade je dobivanje ili implementacija slike u sustav, sljedeći korak je njezina obrada kako bi moglo doći do ekstrakcije. Završni koraci su segmentacija znakova i njihovo očitavanje i prepoznavanje.



Slika 3. Faze ANPR sustava [10]

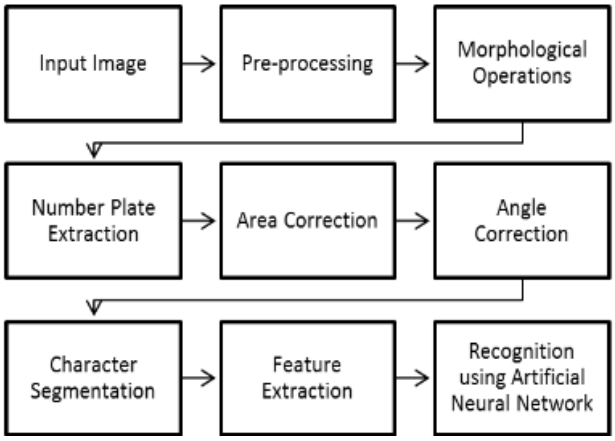
3.2.1. Rad ANPR sustava

Sustav za automatsko prepoznavanje automobilskih tablica izvlači registarske tablice iz određene slike, a to radi u nekoliko koraka. Baza je dohvaćanje slike automobila kamerom. U obzir se trebaju uzimati i kvaliteta kamere i njezino snimanje jer sve to utječe kasnije na kvalitetu slike iz koje se izvlači registarska tablica. Za izvlačenje registarskih tablica poželjno je imati zadovoljene i glavne uvjete kojima se određuju granice, boje na tablicama ili postojanje nekih znakova. Segmentiranjem tih istih znakova tako što se iz njih izvlače informacije u boji, označavanjem ili podudaranjem prema položaju, izdvajaju se svi znakovi i spremaju svaki zasebno kako bi kasnije stvorili cjelinu. Posljednja faza u ANPR sustavu je prepoznavanje segmentiranih znakova. Oni se iščitavaju po sistemu podudaranje sa unaprijed napravljenim predlošcima ili korištenjem klasifikatora, koji su većinom umjetne neuronske mreže. Bitno je naglasiti da bi svi automatski sustavi za prepoznavanje automobilskih tablica trebali raditi u svim uvjetima, bilo da su to problemi sa osvjetljenjem, pozadinama, mogućim defekcijama oblika tablica ili udaljenosti od kamere do automobila.

Najznačajniji i najbitniji algoritmi koji su potrebni softveru za prepoznavanje registarskih tablica su:

- lokalizacija – pronalaženje i izoliranje tablice na slici
- orijentacija i veličina – kompenzacije dimenzija prema potrebnoj veličini

- normalizacija – podešavanje kontrasta i svjetline
- segmentacija – pronalazak znakova na lokaliziranoj tablici
- prepoznavanje znakova
- geometrijska analiza – provjera znakova i položaja za svaku regiju
- razdvajanje prepoznatih vrijednosti u više slika za što pouzdaniji rezultat jer nikada nije sigurno kakve sve mogu biti prvotne slike



Slika 4. Sustav koji je predložen za optimalan rad ANPR-a

4. OPENCV

Open Source Computer Vision Library kraćeg naziva OpenCV knjižnica je algoritama otvorenog koda za računalni vid i strojno učenje. Glavni ciljevi OpenCV projekta su: unaprjeđenje istraživanja kompjuterskog vida pružajući kod i za infrastrukturu, širenje znanja o računalnoj viziji uz mogućnost zajedničkog nadograđivanja svih inženjera, stvaranje aplikacija temeljenih na računalnom vidu koji bi olakšavali svakodnevnicu.

Sama biblioteka posjeduje više od 2500 optimiziranih algoritama, koji su ustvari jedni od najmodernijih, a ujedno i klasičnih algoritama u strojnom učenju i računalnom vidu. Ovi algoritmi često su upotrebljavani u aplikacijama za prepoznavanje lica, objekata, za klasificiranje ljudskih radnji, praćenje pokretnih objekata, spajanje slika. Uz sve to OpenCV pruža i mogućnost dorade i uređivanja slika tako da se tamo ne mogu vidjeti crvenila na očima, praćenje pokreta oči ili scenografija u pokretu. Sam OpenCV ima sučelje u Pythonu, C++, Java i MATLAB-u, a podržavaju ga Windows, Linux, Android i Mac OS. Svaka aplikacija koja je temeljena na OpenCV biblioteci naginje da bude za detekciju u stvarnom vremenu. Ako bilo koji program pišemo od samog početka tada je jasno da se moraju definirati mnoge stvari, a pogotovo ako je riječ o obradi slike i računalnom vidu. Priliko svake obrade slike bitno je definirati točke, pravokutnike, rubove, ali i neke promjene u bojama (kontraste). Kako su strukture već više puta optimizirane za brzinu i memoriju, nije se potrebno brinuti o detaljima implementacije.

Tijekom procesiranja slike dopušta se filtriranje slike, morfološke operacije, geometrijske transformacije, pretvorbe u boji, crtanje, histogrami, analize oblika, analize pokreta i otkrivanje značajki. Detekcijom objekata se otkrivaju prave lokacije samog objekta te nije bitno kakve je on vrste. Upravo je položaj objekta najčešća pogreška i kritičan korak u mnogim sustavima računalnog vida. Najvažniji korak u cijelom programu je prepoznavanje teksta, točnije znamenaka sa registarske tablice. U današnje vrijeme jako je važno moći prepoznati registarske tablice kako bi se očuvala sigurnost i ljudi i vozila u prometu ili na parkiralištima.

5. KORIŠTENJE OPENCV BIBLIOTEKE ZA DETEKCIJU I IDENTIFIKACIJU REGISTARSKIH TABLICA

Kako bi nam bilo omogućeno da započnemo proces detekcije i identifikacije registarskih tablica na automobilima najbolje je za to imati primjer. U ovom radu detekcija vozila prema obrisima tablica i smjer kretanja pogledat ćemo na primjeru videa. Sama identifikacija registarske tablice i spremanje njezine oznake u Excel tablicu dana je na primjeru vlastitog automobila. Za potrebe ovog projekta potrebno je koristiti:

- Visual Studio Code
- Python, verzija 3.11.
- OpenCV, verzija 4.3.0
- Microsoft Excel

Testiranje se vršilo na više slika i video zapisa. Za početak ćemo analizirati kod i način na koji su se prepoznali obrisi tablice i time se smatra da je to automobil koji se kreće u određenu stranu. Prije početka upisivanja koda važno je napomenuti da je potrebno instalirati i osnovne module u Pythonu sa kojima će se kasnije uz OpenCV biblioteku pokretati program. Moduli koji su korišteni u ovom radu su:

- cv2 – modul OpenCv-a
- numpty – modul u sklopu OpenCV biblioteke koji radi sa nizovima
- time – modul za vrijeme

5.1. Detekcija i smjer kretanja automobila u realnom vremenu

Prilikom detektiranja samog automobila potrebno je izraditi video koji sadrži objekte koji se kreću u dva smjera. Nakon što u programu odredio veličinu videa, izradit ćemo i dvije linije koje će nam označavati prostor u kojem se kreću automobili i između čijih ćemo udaljenosti pratiti smjer kretanja automobila. Linije će nam pomoći da se prolaskom kroz prvu liniju prepozna objekt, a do druge linije on bude pribrojen u sveukupni broj automobila koji će prolaziti u istom vremenu. Da se izuzmu moguće smetnje pozadine, poput sjene, pozivanjem funkcije, *createBackgroundSubtractorMOG2()*, koja se nalazi unutar OpenCV biblioteke,

stvorit će se pozadina koja ima samo čiste objekte. Unosom opcija *kernal* korisnik će moći izvršiti napisani kod zbog toga što će mu to biti omogućeno u pozadini. Prepoznavanjem objekata oni će se uokviriti te će se svako njihovo pomicanje pratiti od početne do krajnje linije.

Samo prepoznavanje određeno je pronalaskom kontura- obrisa tablica koje na sebi imaju slova i pravokutnog su oblika. Kako bi se što lakše našle konture prije njih je dobro sve obraditi u sivoj boji koja je pogodna za njihov nalazak. Slika 5. prikazuje dio koda u kojem su navedeni karneli, binarizacija i dio odgovoran za pronalazak kontura na slici.

```
51  #Kernals
52  kernalOp = np.ones((3,3),np.uint8)
53  kernalOp2 = np.ones((5,5),np.uint8)
54  kernalCl = np.ones((11,11),np.uint8)
55
56
57  font = cv2.FONT_HERSHEY_SIMPLEX
58  cars = []
59  max_p_age = 5
60  pid = 1
61
62
63  while(cap.isOpened()):
64      ret,frame=cap.read()
65      for i in cars:
66          i.age_one()
67      fgmask=fgbg.apply(frame)
68      fgmask2=fgbg.apply(frame)
69
70      if ret==True:
71
72          #Binarization
73          ret,imBin=cv2.threshold(fgmask,200,255,cv2.THRESH_BINARY)
74          ret,imBin2=cv2.threshold(fgmask2,200,255,cv2.THRESH_BINARY)
75          #OPening i.e First Erode the dilate
76          mask=cv2.morphologyEx(imBin,cv2.MORPH_OPEN,kernalOp)
77          mask2=cv2.morphologyEx(imBin2,cv2.MORPH_CLOSE,kernalOp)
78
79          #Closing i.e First Dilate then Erode
80          mask=cv2.morphologyEx(mask,cv2.MORPH_CLOSE,kernalCl)
81          mask2=cv2.morphologyEx(mask2,cv2.MORPH_CLOSE,kernalCl)
82
83
84          #Find Contours
85          countours0,hierarchy=cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
86          for cnt in countours0:
87              area=cv2.contourArea(cnt)
88              print(area)
89              if area>areaTH:
```

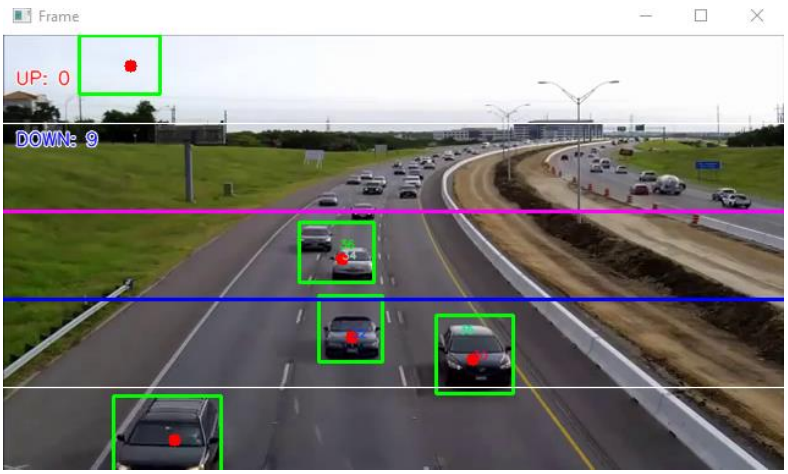
Slika 5. Dio koda odgovoran za ubacivanje karnela i pronalazak kontura

Na temelju tih zadanih uvjeta prati se vožnja automobila. Stalnim uspoređivanjem njegovih koordinata i koordinata linija prema kojima se približava povezano je sa smjerom kretanja koji je određen sa „gore“ i „dolje“. Na slici ispod (Slika 6.) prikazan je dio koda koji određuje smjer kretanja automobila i istovremeno broji koliko je automobila prošlo u kojem smjeru. Ako se na trenutnom položaju videa ne vidi niti jedan objekt tada se poziva pod-program koja će tijekom pomicanja dodavati nove vrste vozila u svoj algoritam kako bi se kasnije proširio cijeli program i bio u stanju prepoznati ne samo automobile nego i sve ostale vrste vozila. Završetkom dovođenja nove vrste u pod-program od početka se provoditi glavni program.

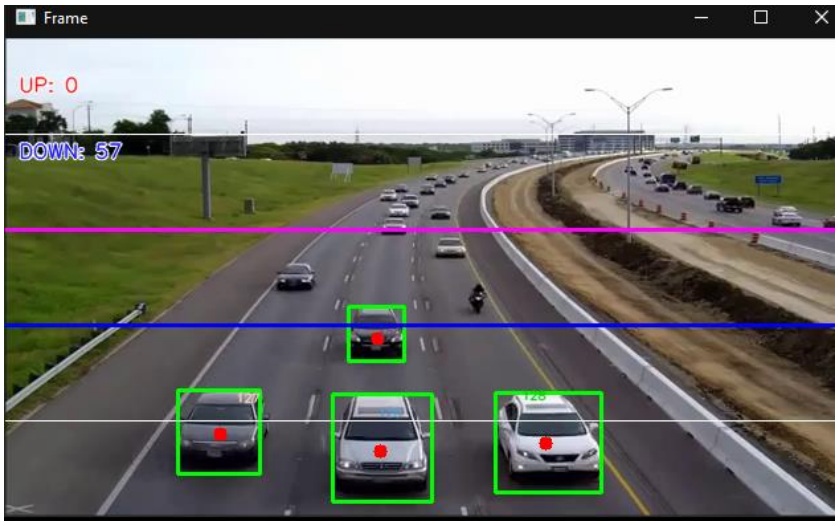
```
####Tracking######
m=cv2.moments(cnt)
cx=int(m['m10']/m['m00'])
cy=int(m['m01']/m['m00'])
x,y,w,h=cv2.boundingRect(cnt)

new=True
if cy in range(up_limit,down_limit):
    for i in cars:
        if abs(x - i.getX()) <= w and abs(y - i.getY()) <= h:
            new = False
            i.updateCoords(cx, cy)

            if i.going_UP(line_down,line_up)==True:
                cnt_up+=1
                print("ID:",i.getId(),'crossed going up at', time.strftime("%c"))
            elif i.going_DOWN(line_down,line_up)==True:
                cnt_down+=1
                print("ID:", i.getId(), 'crossed going up at', time.strftime("%c"))
            break
        if i.getState()=='1':
            if i.getDir()=='down'and i.getY()>down_limit:
                i.setDone()
            elif i.getDir()=='up'and i.getY()<up_limit:
                i.setDone()
        if i.timedOut():
            index=cars.index(i)
            cars.pop(index)
            del i

    if new==True: #If nothing is detected,create new
        p=vehicles.Car(pid,cx,cy,max_p_age)
        cars.append(p)
        pid+=1

cv2.circle(frame,(cx,cy),5,(0,0,255),-1)
img=cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
```

Slika 6. Praćenje smjera kretanja automobila

Rezultat programa u kojem je vidljivo da svi automobili idu u jednom smjeru te da je sigurnost na prometnici osigurana, a svaki od automobila je u jednom trenu zasebno označen, prikazan je na slikama ispod (Slika 7., Slika8).



Slika 7. Prikaz rada aplikacije za detekciju i smjer kretanja



Slika 8. Rad programa i nakon prolaska određenog broja automobila

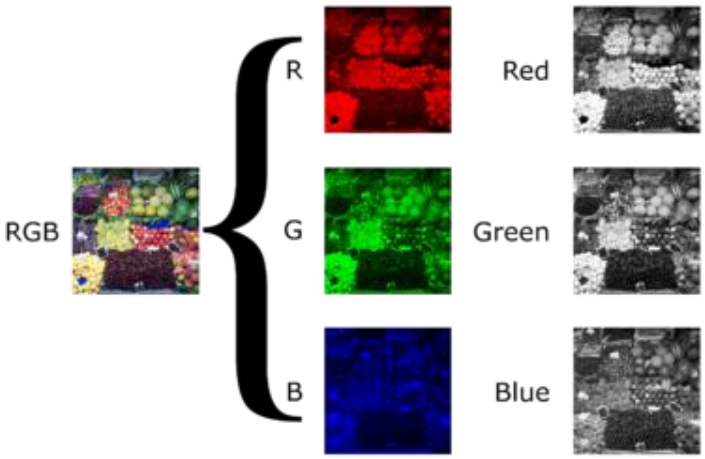
5.2. Detekcija i identifikacija registarskih tablica

U prethodnom poglavlju detektirali smo automobil na temelju pretpostavke da su sve registarske tablice pravokutnog oblika i da na sebi imaju znamenke. Svaki od automobila bio je uočen u okvirima koji su dani – između dvije linije koje smo sami odredili na početku. Bitno je napomenuti da iako su mjestimice bili odjednom uočena i detektirana dva automobila, ona su se svojim pomicanjem razdvojila i time se brojali kao dva zasebna vozila. U daljnjem dijelu rada prikazat ćemo detaljnije, i to na primjeru, na koji način se detektiraju i identificiraju registarske tablice, kako da ih se prepozna na automobilu i kako da se iz njih izvuku podaci. Slika 9. prikazuje vozilo na kojem je registarska oznaka jasno vidljiva i na kojemu će se vršiti prvo testiranje. Kako bi se iščitala registracijska oznaka, odabrana slika se mora pojednostavniti i pripremiti za daljnju obradu. Postupak pojednostavljenja kreće sa pretvorbom boje u sive nijanse pomoću čega se smanjuje veličina slike i ubrzava program, a potom se otkrivaju rubovi koji se daljnjim procesom obrađuju dok se ne pronađe neki rezultat. U idealnom slučaju taj rezultat će biti uspješan i izbaciti na očitanu tablicu u Excelu.

Slika 9. Originalna slika za vršenje testiranja

5.2.1. Pretvorba u sive nijanse – *Grayscale*

Postupak pretvorbe u sive nijanse ili *grayscale* kao rezultat izbacuje pojednostavljenu sliku koja je tada u sivim nijansama, ali isto tako taj postupak ubrzava vrijeme obrade slike. Vrijeme obrade se tada smanjuje zato što su sive boje pohranjene u dvodimenzionalnim nizovima. U pravilu nema neke opće formule kojom bi se slika pretvorila u sive nijanse već je to u velikoj ovisnosti o veličini boja koje predstavljaju originalnu fotografiju. Korištenjem OpenCV biblioteke u Pythonu korištenje grayscale metode zasniva se na 3 boje – crvena (Red), zelena (Green) i plava(Blue) – RGB i na temelju tih boja se uspostavljaju nijanse sive na svim slikama. (Slika 10.)



Slika 10. Primjer *RGB Grayscale* metode

```
img = cv2.imread('kia1.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(cv2.cvtColor(gray, cv2.COLOR_BGR2RGB))
```

Slika 11. Dio koda koji pretvara sliku u sive nijanse

Slika 11. prikazuje dio koda gdje se pomoću funkcije *cvtColor()* iscrtava verzija slike u kojoj vrijednost svakog piksela je jedan uzorak koji predstavlja količinu svjetlosti. Na slici ispod (Slika 12.) vidljiv je rezultat pretvorbe slike u sive nijanse.

Slika 12. Slika u sivim nijansama – grayscale

5.2.2. Otkrivanje i pronalazak kontura – Canny detektor rubova

Otkrivanje rubova Canny detektorom uključuje matematičke metode koje imaju cilj identifikacije točaka na digitalnoj slici. Ako je jakom mijenja svjetlina na slikama, otkrivene točke se formiraju u skupine i čine rub. Da bi rub bio valjan on je morao proći kroz Gaussov filter kojim se miče šum sa slika, zatim se mjeri čvrstoća ruba gradijentom intenziteta slike. Sljedećom fazom se uvode pragovi kojima se odlučuje koji rubovi postoje, a koji ne. Ako je prag niži, otkrit će se više pragova, ali će sa time dobiveni rezultat biti osjetljiv na buku i otkrit će se neželjeni rubovi. Bitno je provoditi *Canny()* funkciju nakon što je slika pretvorena u sive nijanse jer se jedino tako mogu većinom očitati ispravno rubovi.

```
bfilter = cv2.bilateralFilter(img, 11, 17, 17) #Noise reduction
edged = cv2.Canny(bfilter, 30, 200) #Edge detection
plt.imshow(cv2.cvtColor(edged, cv2.COLOR_BGR2RGB))
```

Slika 13. Kod vezan za Canny detektor rubova

Na slici iznad (Slika 13.) u programu umjesto Gaussovo filtera koristi se funkcija *bilateralFilter()* koja djeluje na isti način kao i Gaussov filter, a uspješno odrađuje zadatak micanja buke sa slike. Slika 14. je slika nakon testiranja *Canny()* funkcijom u kojoj su prikazani mnogi rubovi, a među njima možemo vidjeti i rubove koji tvore konturu oko registarske tablice.

Slika 14. Implementacija Canny detektora rubova

Jednom kada primijenimo Canny algoritam za detekciju rubova slika je spremna da se na njoj pronađu konture koju ti rubovi tvore, a da uz to jedne od njih čine i našu traženu tablicu. Funkcija *findContours()* pronaći će sve vidljive konture na slici i spremiti ih u vektor. Kasnije se uzete konture poredaju po veličini (Slika 15.), tako što se međusobno uspoređuju, a iz njih izdvajamo 10 najvećih. Pretpostavljamo da će u tih 10 ući i kontura koja čini našu tablicu.

```
keypoints = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(keypoints)
contours = sorted(contours, key=cv2.contourArea, reverse=True)[:10]
```

Slika 15. Dio koda za pronalazak kontura i njihovo sortiranje

```
location = None
for contour in contours:
    approx = cv2.approxPolyDP(contour, 10, True)
    if len(approx) == 4:
        location = approx
        break
```

Slika 16. Uspoređivanje kontura i pronalazak tražene sa 4 ruba

Željenu konturu pokušavamo naći funkcijom *approxPolyDP()* koja približno određuje oblik naše konture, uspoređuje ih sa dobivenim konturama i spremamo u vektor *approx.* Vektor tada testira ima li točno 4 ruba te ukoliko je uvjet zadovoljen stvara se maska tako što ju unutar funkcije *zeros()* prvo ispunimo s onoliko nula koliko ima piksela na originalnoj slici u sivim nijansama – *grayscale.* Kao rezultat koda prikazanog na slici iznad (Slika 16.) dobit ćemo crnu pozadinu koja će se popuniti za označenom konturom naše slike ako iskoristimo funkciju *bitwise_and()* (Slika 17.).

Slika 17. Pronalazak točne konture

Nakon dobivene velike slike, zbog lakšeg provođenja iščitavanja znamenaka optičkim sustavom, obreže se dio slike koji čini sama registarska tablica čime se ona izdvaja iz cijelog početnog sustava (Slika 18., Slika 19.).

```
mask = np.zeros(gray.shape, np.uint8)
new_image = cv2.drawContours(mask, [location], 0,255, -1)
new_image = cv2.bitwise_and(img, img, mask=mask)
```

Slika 18. Kod za izdvajanje prepoznate registarske tablice

Slika 19. Prepoznati i izdvojeni dio slike

5.2.3. Prepoznavanje znamenaka – segmentacija

Segmentacija ili prepoznavanje znamenaka najvažniji je korak za bilo koji optički sustav prepoznavanja znakova (OCR). Za točnost OCR sustava bitan je algoritam koji se odabere jer ako postoji dobra segmentacija znakova i točnost prepoznavanja će biti visoka. U ovom primjeru korišten je EasyOCR sustav. Easy OCR je čitač znakova koji ovisi o fontu, a temelji se na podudaranju sa predlošcima. Slika 20. prikazuje kod koji je namijenjen za očitavanje sa registarske tablice.

```
reader = easyocr.Reader(['en'])
result = reader.readtext(cropped_image)
result
```

Slika 20. Prepoznavanje znakova OCR sustavom

Znamenke sa očitane tablice ispisuju se, a istovremeno se i pohranjuju kako bi kasnije bile spremljene u Excel tablicu. Osim segmentacije ovim načinom, moguće ju je temeljiti i na tome da se pretražuju rubne točke i traže rubne vrijednosti objekta. Važno je znati da na moguće poteškoće i nemogućnost očitanja može doći zbog šuma na slici, okvira, razmaka, kuta slikanja, svjetlosti ili osvjetljenja. Kod takvih slučajeva često nam se javlja greška da nisu pronađene konture pa se daljnja segmentacija ne može odraditi. U našem slučaju, koji je uspješan, na originalnoj slici se pokazuje okvir na registracijskoj tablici koja je očitana (Slika 22.).

```
text = result[0][-2]
font = cv2.FONT_HERSHEY_SIMPLEX
res = cv2.putText(img, text=text, org=(approx[0][0][0], approx[1][0][1]+60), fontFace=font, fontScale=1, color=(0,255,0), thickness=
res = cv2.rectangle(img, tuple(approx[0][0]), tuple(approx[2][0]), (0,255,0),3)
plt.imshow(cv2.cvtColor(res, cv2.COLOR_BGR2RGB))
```

Slika 21. Dio koda za ispisivanje prepoznate tablice na slici

Slika 22. Uokvirena registarska tablica nakon identifikacije

```
print(text)
✓ 0.2s
KR 588 JV
```

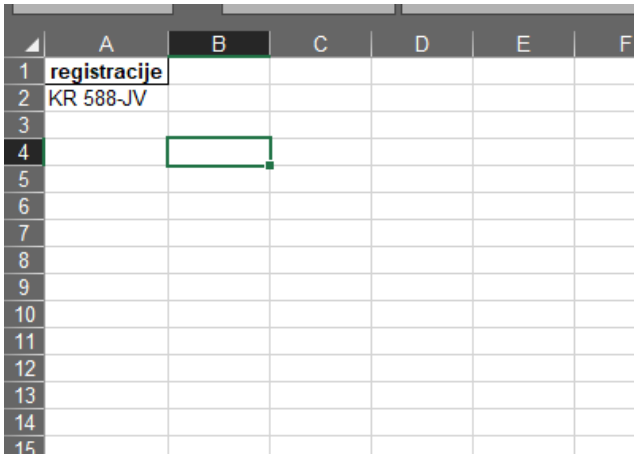
Slika 23. Kod za ispis u terminalu Pythona

5.2.4. Spremanje podataka

Svaki rezultat koji se uspješno očita (makar i samo djelomično) sprema se u liste koje zatim stvaraju novi Excel dokument u kojem pohranjuju navedene podatke (Slika 24., Slika 25.). Funkcijom *append()* dodali smo u niz sve registracije koje će se uspjeti očitati, nakon što se odredi naziv stupca u isti se pohranjuju podaci sa funkcijom *DataFrame().*

```
list1=[]
list1.append(text)
col1="registracije"
data=pd.DataFrame({col1:list1})
data.to_excel('zavrsetak.xls', index=False)
```

Slika 24. Kod za pohranu iščitanih podataka

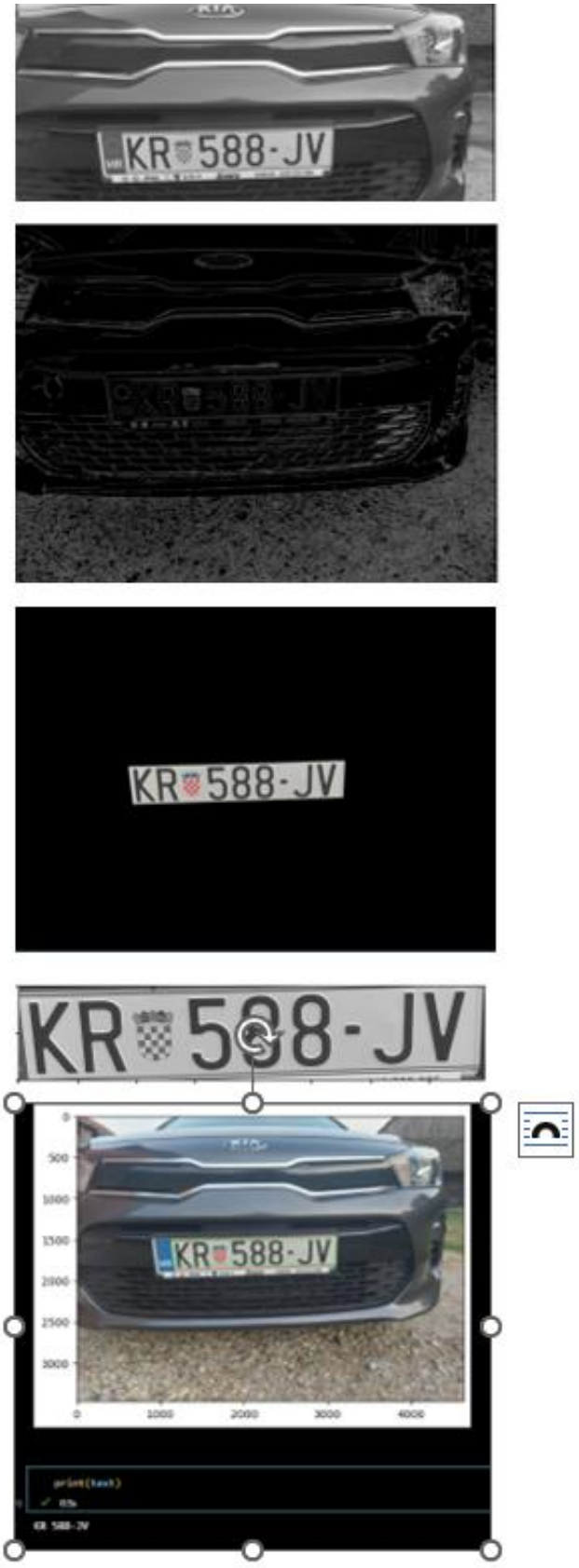


Slika 25. Prikaz popunjavanja tablica u Excelu putem Pythona

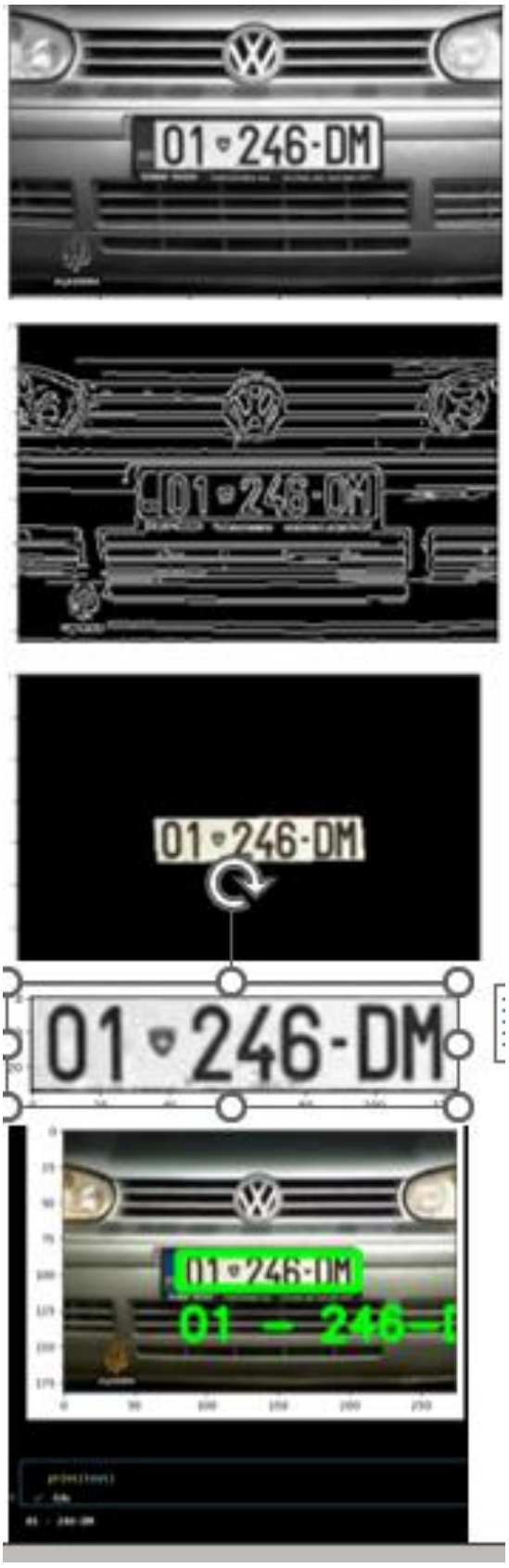
5.2.5. Rezultati prepoznavanja

Testiranjem programa on može biti uspješan ili neuspješan. U slučajevima kada je program uspješan dolazi do ispisivanja registarskih oznaka u Excel tablicu i u terminal programskog jezika Python. Na primjerima koji slijede vidljivo je da su prepoznate tablice više različitih država, ali samo u trenucima kad je registarska tablica dobro vidljiva i kada su jasno prikazane konture (Slika 26., Slika 27., Slika 28.).

Slika 26. Uspješno prepoznate američke registarske tablice

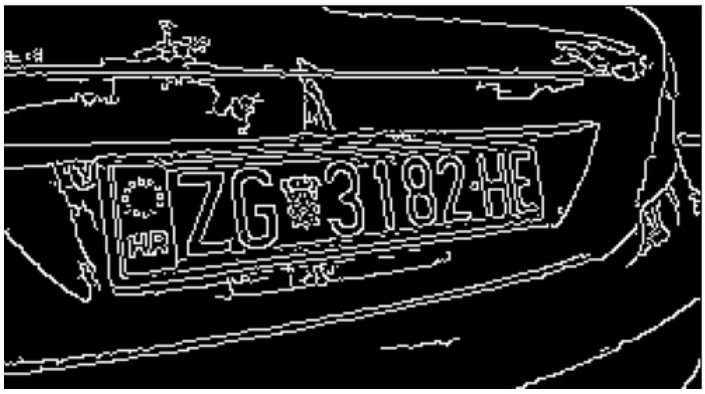


Slika 27. Uspješno prepoznate hrvatske registarske tablice



Slika 28. Uspješno prepoznavanje tablica

Neuspješnim programom smatra se onaj kod kojeg nije bilo moguće *Canny* detektorom rubova odrediti točne rubove registarske tablice te nam je program završio sa ocrtanim konturama i javljanjem greške (Slika 29). Od svih ponuđenih kontura niti jedna ne uokviruje točno registarsku tablicu pa se ona smatra nepostojećom i program tu završava svoj tijek. Moguće greške su što je slika pod kutom i visoke rezolucije te je onda program uočio više kontura od kojih niti jedna nije zadovoljavajuća za prepoznavanje tablice.



Slika 29. Kraj programa u ovom stupnju zbog nemogućnosti prepoznavanja rubova tablice

Druga moguća greška bila bi djelomično prepoznavanje znakova. U takvim slučajevima nedostaju neki znakovi prilikom optičkog očitavanja i rezultat je djelomično spremljen u Excel tablicu. Slika 30. pokazuje originalnu fotografiju sa koje se pokušala razaznati informacija na registarskoj tablici, a na slici ispod nje (Slika 31.) rezultat je prepoznavanja u kojem nisu prepoznate znamenke koje je nalaze poslije grba.

Slika 30. Početna fotografija za testiranje

Slika 31. Ispis djelomično prepoznate tablice

Razlozi zbog kojih može doći do pogreške su razni. U većini slučajeva razlozi su veličina registracijske oznake (različite dimenzije, različit razmještaj slova, grbovi,…), kvaliteta snimljene fotografije, pozicija vozila, pozicija fotografa, kut snimanja, osvjetljenje,…

6. KRITIČKO RAZMIŠLJANJE

Na samom početku bilo detekcije automobila u stvarnom vremenu ili detekcije registarskih tablica postavljeno je isto pitanje, hoće li detekcija biti uspješna. U dijelu rada gdje se detektira automobil u stvarnom vremenu vidljivo je da se svaki automobil prepozna i označen je sa okvirom u boji. Sama detekcija vršena je prema pretpostavci da svaki automobil ima registarsku oznaku sa znamenkama i da je pravokutnog oblika. Također, brojanje automobila napravljeno je pomoću linija. Smatram da se detaljnijim analiziranjem i testiranjem može izbjeći postavljanje linija kao referentne vrijednosti za prepoznavanje objekata. Isto tako, algoritam bi se trebalo nadograditi jer ne prepoznaje kada dva ili više automobila prođu u isto vrijeme i detektira ih kao jedan. Uz to, kvaliteta kamere ili videa trebala bi biti bolje rezolucije, a i sam kut snimanja morao bi obuhvatiti više smjerova kako bi se testiralo radi li program i kada je obuhvaćeno područje sa međusobno suprotnim smjerovima kretanja. Ovaj program dobar je za prepoznavanje kretanja automobila u suprotnom smjeru te za brojanje automobila koji se kreću u jednom smjeru. Možda bi se uz pomoć dodatno prepoznavanja sljedećeg karakterističnog objekta na automobilu moglo pripomoći ovom programu koji bi tada mogao funkcionirati sa dovoljno dobrom učinkovitošću.

Prilikom detekcije i identifikacije registarskih tablica najveći problemi se stvaraju prilikom otkrivanja okvira registracijske tablice i pri samom očitavanju teksta sa tablice. *Canny* metoda koja se koristi u ovom primjeru trebala bi se dodatno doraditi kako bi rezultat bio u potpunosti koristan. Na testiranju 10 slika, dvije slike registarskih tablica su točno određene, jedna je djelomično, a na ostalima je odabrana kriva kontura. Krivi odabir kontura najčešći je uzrok nastale pogreške u algoritmu. Sa sigurnošću je dio koda prikazan na slici (Slika 32.) odgovoran za krivi odabir kontura jer ukoliko ne ograničimo broj kontura ona će nam svejedno vratiti rezultat koji nije dobar.

```
keypoints = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(keypoints)
contours = sorted(contours, key=cv2.contourArea, reverse=True)[:10]

✓ 1.3s

location = None
for contour in contours:
    approx = cv2.approxPolyDP(contour, 10, True)
    if len(approx) == 4:
        location = approx
        break
```

Slika 32. Dio koda koji je potrebno mijenjati

Veći postotak uspješnosti *Canny* programa je kada je slika koju želimo analizirati u nižem stupnju rezoluciji. Ovo možda zvuči čudno, ali upravo zbog niske rezolucije su pronađene manje irelevantne konture, a program je tako lakše prepoznao traženo. Da bi se povećala uspješnost ovog programa potrebno je uvesti više uvjeta prilikom određivanja vektora *approx*. Kako bi se povećala uspješnost programa, moguće je i *Canny* detektor rubova zamijeniti sa nekom drugom metodom kao što su jednostavno određivanje praga ili prilagodljivo određivanje praga. Jednostavno određivanje praga je metoda segmentacije koja na slikama u sivim nijansama razdvaja objekte od pozadine temeljem intenziteta osvjetljenja. Glavna razlika između *Canny* detektora rubova i jednostavnog određivanja pragova od prilagodljivog određivanja pragova je što prilagodljivo određivanje pragova odmah traži tablicu na temelju znakova, a ne na temelju rubova. Još jedna metoda koja se ne koristi prepoznavanjem rubova je i klasifikator *Haar kaskade*.

Nakon što bi se poboljšalo samo detektiranje registarske tablice, isprobali bi se i drugi algoritmi segmentacije znakova. Kada se koristi OCR njime nismo osigurali točnost znamenaka očitanih sa tablica. Zbog ovisnosti o kutu snimanja slike, razlučivosti i osvjetljenju te neprepoznatih rubova i kontura on može dati pogrešan rezultat. Kao drugi mogući algoritam za isprobavanje očitavanja znamenaka koristi se i *K-NN algoritam*. Algoritam k-najbližih susjeda služi za nadgledavanje strojnog učenja, ali i za klasifikaciju. Krajnji rezultat trebao bi izaći kao točniji zbog načina na koji se algoritam provodi, a to je da svaku konturu očitava kao zaseban znak i očitava te na kraju spaja u jednu cjelinu.

Mogućnosti napredovanja u ovom programu su velike i potrebno ih je stalno razvijati istom brzinom kojom se događaju promjene u polju računalnog vida i samim automobilima. Sa promjenom modela automobila, postoji mogućnost da se promijeni i način na koji izgleda tablica pa nekim standardiziranim metodama nije moguće detektirati istu. Osim dizajnerskih promjena moguće su i promjene u brzinama automobila te se prilikom detekcije u stvarnom vremenu mora postići što brže provođenje programa kako bi se dobili relevantni podaci. Svaki podatak koji je točno očitan može u daljnjim postupcima pomoći u procesu osiguravanja što bolje sigurnosti na prometnica.

7. ZAKLJUČAK

Mogućnost detekcije i raspoznavanja znakova na registarskim tablica proces je koji se razvija već duži niz godina i koji će se i dalje razvijati uz pomoć računalnog vida i umjetne inteligencije. Detekcija i raspoznavanje znakova na tablicama od velike je važnosti za sigurnost prvenstveno u prometu, a zatim i na drugim javnim mjestima. Proces i nije tako jednostavan kako se na prvu činilo, potrebno je uložiti puno više od osnovnog poznavanja računalnog vida. Osim njega, svaki čovjek koji se bavi računalnim vidom istovremeno bi trebao imati i pravo istraživati ljudski vid i percepciju. Jedino se na taj način mogu postići identični rezultati prilikom korištenja računalnog vida.

Kako se i dalje radi na stalnom usavršavanju prepoznavanja registarskih tablica nemoguće je reći da neki program radi sa 100%-om točnošću. Tako smo i na primjeru ovog rada vidjeli da se neke tablice mogu očitati točno, druge djelomično, a postoji i slučaj kada nije moguće uopće detektirati tablicu. Pozitivna strana je što postoji mnoštvo različitih algoritama čijom bi se međusobnom povezanošću i stalnim testiranjem mogao napraviti novi program čija bi učinkovitost bila veća. Izuzetno je bitno i da se proces razvija brzo te da automatizirani sustavi unutar njega otkrivaju i otklanjaju moguće pogreške ljudi. Očito je da se za svaku pojedinu zemlju razlikuju tablice te da bi se trebao osigurati jedan program koji bi mogao razaznati sve tablice, a ne da postoji program za svaku pojedinu državu zasebno. Međusobnom potporom i željom za unaprjeđenjem u sustavu sigurnost moguće je osmisliti takav projekt.

Cilj ovog rada bio je analizirati na koji način se detektiraju i identificiraju registarske tablice te koji bi bio poželjan način spremanja istih. Osim analize, rad je obuhvatio i testiranje kojim se pokušala saznati točnost željenih rezultata. Pomoću OpenCV biblioteke omogućena nam je lakša komunikacija između odabranog programskog jezika (Python) i medija na kojem ćemo proučavati. Bitno je napomenuti da je moguće napraviti testiranja u stvarnom vremenu, ali isto tako i na prethodno napravljenim snimkama. Tijekom testiranja u ovom radu su korišteni prethodno snimljeni videozapisi i fotografije. Zbog velikog broja algoritama koje su pohranjene u OpenCV biblioteci moguće je na temelju njih napraviti dorade na programu kako bi se mogao koristiti i u stvarnom vremenu. Bitno je reći da se učinkovitost ovog programa nikako ne može dokazati testiranjem na samo jednom primjeru. Za provođenje ovakvog i svakog budućeg projekta potrebno je testirati ga na velikom broju primjera i pratiti promjene koje se pritom događaju jer samo tako možemo biti sigurni na koji način se neki dio može poboljšati. Jer napredak je uvijek moguć!

LITERATURA

[1] Jason Brownlee: A Gentle Introduction to Computer Vision: https://machinelearningmastery.com/what-is-computer-vision/ (6.9.2022.)

[2] IBM: https://www.ibm.com/topics/computer-vision (6.9.2022.)

[3] Ilija Mihajlović: Everything You Ever Wanted To Know About Computer Vision: https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e (6.9.2022.)

[4] https://hr.wikipedia.org/wiki/Ra%C4%8Dunalni_vid (6.9.2022.)

[5] Mike Constant: An Introduction To ANPR: https://www.cctv-information.co.uk/i/An_Introduction_to_ANPR (10.9.2022.)

[6] https://en.wikipedia.org/wiki/Automatic_number-plate_recognition (10.9.2022.)

[7] Adrian Rosebrock: OpenCV Tutorial: A Guide to Learn OpenCV: https://www.pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/ (10.9.2022.)

[8] https://en.wikipedia.org/wiki/OpenCV (10.9.2022.)

[9] https://opencv.org/about/ (10.9.2022.)

[10] Anand S. jain, Jayshree M. Kundargi: Automatic Number Plate Recognition Using Artificial Neural Network: International Research Journal of Engineering and Technology (IRJET): https://www.irjet.net/archives/V2/i4/Irjet-v2i4181.pdf (10.9.2022.)

[11] D. Lippaiová, M. Michalko, O. Kainz, F. Jakab: Automatic license plate recognition system using OpenCV library: https://www.academia.edu/37196246/AUTOMATIC_LICENSE_PLATE_RECOGNITION_SYSTEM_USING_OPENCV_LIBRARY (11.9.2022.)

[12] James Wilson: Machine learning & license plate recognition: An ideal partnership: https://bigdata-madesimple.com/machine-learning-license-plate-recognition-an-ideal-partnership/ (11.9.2022.)

[13] Margaret Rouse: Grayscale: https://whatis.techtarget.com/definition/grayscale (17.9.2022.)

[14] https://en.wikipedia.org/wiki/Grayscale (11.9.2022.)

[15] Sicara.ai: Edge Detection in OpenCV: A 15 Minutes Tutorial: https://www.sicara.ai/blog/2019-03-12-edge-detection-in-opencv (12.9.2022.)

[16] https://learnopencv.com/contour-detection-using-opencv-python-c/ (12.9.2022.)

[17] https://wikihrhr.top/wiki/canny_edge_detector (13.9.2022.)

[18] https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm (13.9.2022.)

[19] https://towardsdatascience.com/canny-edge-detection-step-by-atep-in-python-computer-vision-b49c3a2d8123 (15.9.2022.)

[20] https://www.techprofree.com/real-time-number-plates-identifier-and-counter-in-python/ (10.9.2022.)

[21] https://towardsdatascience.com/machine-learning-basic-with-k-nearest-neighbors-algorithm-6a6e71d01761 (15.9.2022.)

[22] https://towardsdatascience.com/computer-vision-detecting-objects-using-haar-cascade-classifier-4585472829a9 (15.9.2022.)

[23] https://hr.myservername.com/hands-python-openpyxl-tutorial-with-examples (16.9.2022.)

PRILOZI

I. Link za pristup kodu te sam kod za detekciju automobila i njihovo brojanje u realnom vremenu:
https://www.techprofree.com/real-time-number-plates-identifier-and-counter-in-python/

GLAVNI PROGRAM (main.py)

```
import cv2
import numpy as np
import vehicles
import time

cnt_up=0
cnt_down=0

cap=cv2.VideoCapture("surveillance.m4v")

#Get width and height of video
w=cap.get(3)
h=cap.get(4)
frameArea=h*w
areaTH=frameArea/400

#Lines
line_up=int(2*(h/5))
line_down=int(3*(h/5))
up_limit=int(1*(h/5))
down_limit=int(4*(h/5))

print("Red line y:",str(line_down))
print("Blue line y:",str(line_up))
line_down_color=(255,0,0)
line_up_color=(255,0,255)
```

```
pt1 =  [0, line_down]
pt2 =  [w, line_down]
pts_L1 = np.array([pt1,pt2], np.int32)
pts_L1 = pts_L1.reshape((-1,1,2))
pt3 =  [0, line_up]
pt4 =  [w, line_up]
pts_L2 = np.array([pt3,pt4], np.int32)
pts_L2 = pts_L2.reshape((-1,1,2))

pt5 =  [0, up_limit]
pt6 =  [w, up_limit]
pts_L3 = np.array([pt5,pt6], np.int32)
pts_L3 = pts_L3.reshape((-1,1,2))
pt7 =  [0, down_limit]
pt8 =  [w, down_limit]
pts_L4 = np.array([pt7,pt8], np.int32)
pts_L4 = pts_L4.reshape((-1,1,2))

#Background Subtractor
fgbg=cv2.createBackgroundSubtractorMOG2(detectShadows=True)

#Kernals
kernalOp = np.ones((3,3),np.uint8)
kernalOp2 = np.ones((5,5),np.uint8)
kernalCl = np.ones((11,11),np.uint8)

font = cv2.FONT_HERSHEY_SIMPLEX
cars = []
max_p_age = 5
pid = 1

while(cap.isOpened()):
    ret,frame=cap.read()
    for i in cars:
```

```
        i.age_one()
        fgmask=fgbg.apply(frame)
        fgmask2=fgbg.apply(frame)

        if ret==True:

            #Binarization
            ret,imBin=cv2.threshold(fgmask,200,255,cv2.THRESH_BINARY)
            ret,imBin2=cv2.threshold(fgmask2,200,255,cv2.THRESH_BINARY)
            #OPening i.e First Erode the dilate
            mask=cv2.morphologyEx(imBin,cv2.MORPH_OPEN,kernalOp)
            mask2=cv2.morphologyEx(imBin2,cv2.MORPH_CLOSE,kernalOp)

            #Closing i.e First Dilate then Erode
            mask=cv2.morphologyEx(mask,cv2.MORPH_CLOSE,kernalCl)
            mask2=cv2.morphologyEx(mask2,cv2.MORPH_CLOSE,kernalCl)

            #Find Contours

countours0,hierarchy=cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_A
PPROX_NONE)
            for cnt in countours0:
                area=cv2.contourArea(cnt)
                print(area)
                if area>areaTH:
                    ####Tracking######
                    m=cv2.moments(cnt)
                    cx=int(m['m10']/m['m00'])
                    cy=int(m['m01']/m['m00'])
                    x,y,w,h=cv2.boundingRect(cnt)

                    new=True
                    if cy in range(up_limit,down_limit):
                        for i in cars:
```

```
                if abs(x - i.getX()) <= w and abs(y - i.getY()) <= h:
                    new = False
                    i.updateCoords(cx, cy)

                    if i.going_UP(line_down,line_up)==True:
                        cnt_up+=1
                        print("ID:",i.getId(),'crossed going up at', time.strftime("%c"))
                    elif i.going_DOWN(line_down,line_up)==True:
                        cnt_down+=1
                      print("ID:", i.getId(), 'crossed going up at', time.strftime("%c"))
                    break
                if i.getState()=='1':
                    if i.getDir()=='down'and i.getY()>down_limit:
                        i.setDone()
                    elif i.getDir()=='up'and i.getY()<up_limit:
                        i.setDone()
                if i.timedOut():
                    index=cars.index(i)
                    cars.pop(index)
                    del i

            if new==True: #If nothing is detected,create new
                p=vehicles.Car(pid,cx,cy,max_p_age)
                cars.append(p)
                pid+=1

        cv2.circle(frame,(cx,cy),5,(0,0,255),-1)
        img=cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)

    for i in cars:
        cv2.putText(frame, str(i.getId()), (i.getX(), i.getY()), font, 0.3, i.getRGB(), 1,
cv2.LINE_AA)
```

```
            str_up='UP: '+str(cnt_up)
            str_down='DOWN: '+str(cnt_down)
            frame=cv2.polylines(frame,[pts_L1],False,line_down_color,thickness=2)
            frame=cv2.polylines(frame,[pts_L2],False,line_up_color,thickness=2)
            frame=cv2.polylines(frame,[pts_L3],False,(255,255,255),thickness=1)
            frame=cv2.polylines(frame,[pts_L4],False,(255,255,255),thickness=1)
            cv2.putText(frame, str_up, (10, 40), font, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
            cv2.putText(frame, str_up, (10, 40), font, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
            cv2.putText(frame, str_down, (10, 90), font, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
            cv2.putText(frame, str_down, (10, 90), font, 0.5, (255, 0, 0), 1, cv2.LINE_AA)
            cv2.imshow('Frame',frame)

            if cv2.waitKey(1)&0xff==ord('q'):
                break
        else:
            break

cap.release()
cv2.destroyAllWindows()
```

POD-PROGRAM (vehicles.py)

```
from random import randint
import time

class Car:
    tracks=[]
    def __init__(self,i,xi,yi,max_age):
        self.i=i
        self.x=xi
        self.y=yi
        self.tracks=[]
```

```
        self.R=randint(0,255)
        self.G=randint(0,255)
        self.B=randint(0,255)
        self.done=False
        self.state='0'
        self.age=0
        self.max_age=max_age
        self.dir=None

    def getRGB(self):  #For the RGB colour
        return (self.R,self.G,self.B)
    def getTracks(self):
        return self.tracks

    def getId(self): #For the ID
        return self.i

    def getState(self):
        return self.state

    def getDir(self):
        return self.dir

    def getX(self):  #for x coordinate
        return self.x

    def getY(self):  #for y coordinate
        return self.y

    def updateCoords(self, xn, yn):
        self.age = 0
        self.tracks.append([self.x, self.y])
        self.x = xn
        self.y = yn
```

```
    def setDone(self):
        self.done = True

    def timedOut(self):
        return self.done

    def going_UP(self, mid_start, mid_end):
        if len(self.tracks)>=2:
            if self.state=='0':
                if self.tracks[-1][1]<mid_end and self.tracks[-2][1]>=mid_end:
                    state='1'
                    self.dir='up'
                    return True
                else:
                    return False
    else:
                return False
        else:
            return False

    def going_DOWN(self,mid_start,mid_end):
        if len(self.tracks)>=2:
            if self.state=='0':
                if self.tracks[-1][1]>mid_start and self.tracks[-2][1]<=mid_start:
                    start='1'
                    self.dir='down'
                    return True
                else:
                    return False
            else:
                return False
        else:
            return False
```

```
    def age_one(self):
        self.age+=1
        if self.age>self.max_age:
            self.done=True
        return  True


#Class2


class MultiCar:
    def __init__(self,cars,xi,yi):
        self.cars=cars
        self.x=xi
        self.y=yi
        self.tracks=[]
        self.R=randint(0,255)
        self.G=randint(0,255)
        self.B=randint(0,255)
        self.done=False
```

II. Link za pristup kodu te od za detekciju i identifikaciju automobilski tablica i link za *tutorial*:
https://www.youtube.com/watch?v=NApYP_5wlKY ;
https://github.com/nicknochnack/ANPRwithPython

GLAVN PROGRAM (main.py)

```
import pandas as pd
import cv2
from matplotlib import pyplot as plt
import numpy as np
import imutils
import easyocr

img = cv2.imread('kia1.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(cv2.cvtColor(gray, cv2.COLOR_BGR2RGB))
```

```
bfilter = cv2.bilateralFilter(img, 11, 17, 17) #Noise reduction
edged = cv2.Canny(bfilter, 30, 200) #Edge detection
plt.imshow(cv2.cvtColor(edged, cv2.COLOR_BGR2RGB))


keypoints = cv2.findContours(edged.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(keypoints)
contours = sorted(contours, key=cv2.contourArea, reverse=True)[:10]


location = None
for contour in contours:
    approx = cv2.approxPolyDP(contour, 10, True)
    if len(approx) == 4:
        location = approx
        break
location


mask = np.zeros(gray.shape, np.uint8)
new_image = cv2.drawContours(mask, [location], 0,255, -1)
new_image = cv2.bitwise_and(img, img, mask=mask)
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))


(x,y) = np.where(mask==255)
(x1, y1) = (np.min(x), np.min(y))
(x2, y2) = (np.max(x), np.max(y))
cropped_image = gray[x1:x2+1, y1:y2+1]
plt.imshow(cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB))
```

#Očitavanje i ispis tablice

```
reader = easyocr.Reader(['hr'])
result = reader.readtext(cropped_image)
result
text = result[0][-2]
font = cv2.FONT_HERSHEY_SIMPLEX
res = cv2.putText(img, text=text, org=(approx[0][0][0], approx[1][0][1]+60),
fontFace=font, fontScale=1, color=(0,255,0), thickness=4,
lineType=cv2.LINE_AA)
res = cv2.rectangle(img, tuple(approx[0][0]), tuple(approx[2][0]),
(0,255,0),6)
plt.imshow(cv2.cvtColor(res, cv2.COLOR_BGR2RGB))
print(text)
```

#Spremanje u Excel

```
list1=[]
list1.append(text)
col1="registracije"
data=pd.DataFrame({col1:list1})
data.to_excel('zavrsetak.xls', index=False)
```