Detekcija prometne trake

Šimunović, Bruno

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij računarstva

DETEKCIJA PROMETNE TRAKE

Završni rad

Bruno Šimunović

Osijek, 2021.

SADRŽAJ

1. UVOD

U današnje vrijeme tehnologija jako brzo napreduje te to izaziva želju za većom sigurnosti tijekom svakodnevnog života. Trenutno, cestovni promet nije u najboljem položaju ako se govori o sigurnosti što nam mogu pokazati podaci o broju nesreća tijekom godine. Kao primjer uzima se 2019. godina u Republici Hrvatskoj koja je imala 31.367 prometnih nesreća. Unutar te godine najveći udio su imale nesreće tipa udar vozila u parkirano vozilo sa 15,4% i slijetanje vozila s ceste sa 14,5% [1]. Ovakvi tipovi se mogu reducirati uporabom tehnologije poput ADAS (engl. *Advanced driver-assistance systems*) koja smanjenje ljudski utjecaj i čini vožnju sigurnijom. Takva tehnologija, uporabom pametnog računala upozorava vozača na određene nepravilnosti i opasnosti te smanjuje vjerojatnost prometne nesreće.

Suvremeni automobili imaju ugrađenu ADAS tehnologiju koja pomoću senzora ili kamera unutar automobila promatra i pomaže vozaču tijekom vožnje [2]. Takva tehnologija je trenutno važna za sigurnost prometa te su potrebna rješenja raznih inženjera da bi napredovala. Unutar ovoga završnog rada fokusira se na ADAS tehnologiju za detekciju prometne trake koja omogućava vozaču da ostane unutar svoje trake. Za početak potrebno je znati da prometnu traku definiraju dvije paralelne linije na cesti kojima se označava njezin prostor [3]. Izgled prometne trake najčešće je vrlo jednostavan, zbog čega ne postoje nekakve karakteristične značajke kojima se linije prometne trake mogu otkriti. Proces otkrivanja prometne trake je težak jer je potrebno proći kroz sve uvjete koji nam mogu otežati detekciju ili dati krive rezultate. Neki od takvih uvjeta su loša vidljivost zbog vremena kao kiša, magla, mrak ili nešto drugo poput izlizanih linija ili da ih u opće nema. Potrebno je još provjeriti vrstu linija kako bi se što bolje prikazala traka na videu.

Za razvijanja algoritma za detekciju prometne trake korištene su slike dobivene s prednje kamere u automobilu. Algoritam će detektirati vozačevu prometnu traku. Potrebno je da algoritam bude prilagodljiv jer kamera može biti pod utjecajem raznih nepovoljnih uvjeta. Algoritam je razvijen u višem programskom jeziku zvanom Python 3.9.6.

U narednom poglavlju detaljnije su opisane postojeće metode detekcije prometnih traka te su objašnjeni općeniti načini rada dosadašnjih algoritama kao i njihove prednosti i nedostatci. Nakon toga u trećem poglavlju objašnjava se proces izrade algoritma za detekciju prometnih traka.

Četvrto poglavlje opisuje testiranje ispravnosti rada algoritma i rezultate testiranja na par video primjeraka iz prometa. U petom poglavlju dan je zaključak rada.

1.1. Zadatak završnog rada

Zadatak rada je implementirati algoritam za detekciju prometnih traka. Potrebno je detektirati vlastitu prometnu traku. Prometne trake se detektiraju pomoću obrade slika iz snimke snimljene prednjom kamerom automobila. Nakon obrade slike, prikazati trake na originalnoj snimci.

2. PROBLEM DETEKCIJE PROMETNE TRAKE SA SLIKE DOBIVENE IZ VIDEA I POSTOJEĆI ALGORITMI

Tehnike računalnog vida su glavni alat koji omogućavaju sposobnost shvaćanja okoline za detekciju, identifikaciju i praćenje prometnih traka. Detekcija traka se uglavnom sastoji od pronalaska određenog uzorka poput oznaka traka na cesti. Općenito, kada se govori o metodama detekcije prometnih traka možemo ih podijeliti u tri kategorije: metode zasnovane na postavljanu modela prometnih traka iz detektiranih rubova (engl. *model based methods*), metode zasnovane na značajkama u slici (engl. *Feature based methods*) te metode zasnovane na modelima neuronskih mreža (engl. *neural network based methods*). Svaka od tih metoda ovisi o ulaznom videu s prednje kamere automobila te se pomoću njega prati prometna traka. Također, važno je reći da sve metode imaju slične probleme kao ovisnost o kameri, vremenskim uvjetima, drugim objektima, te kvaliteti prometnih traka. Dodatno, potrebno je uzeti u obzir i karakteristike pametnog računala automobila da bi sve moglo raditi u stvarnom vremenu. Generalno se koriste razni algoritmi kako bi se uklonile smetnje, ali potrebno je se pripaziti da obrada nije jako zahtjevna jer je vrijeme obrade podataka ulaznog videa značajno. Zbog toga većina algoritama nije prikladna za uporabu u stvarnom vremenu, te se ne koriste. Od navedenih metoda detekcije, danas se najviše upotrebljava metoda neuronskih mreža jer je došlo do značajnog napretka tehnologije u zadnjem desetljeću u području računalnog vida. Neuronske mreže su pouzdane i omogućavaju detekciju prometnih traka u stvarnom vremenu. Metoda se bazira na treniranju računala s velikom skupinom podatka da „razmišlja kako čovjek“ tj. prepoznaje zadane objekte. Ona nije podložna redizajnu postavljenih kriterija nego se sama prilagođava promjenama ulaznih podataka. U današnjici, takva metoda se primjenjuje u širokom rasponu poput prepoznavanja govora, analize podataka te predviđanje rezultata (npr. inflacija/deflacija valute, detekcije prometne trake). Uporaba ovakvog algoritma za stvarnu autonomnu vožnju zajamčena je sigurnost ali i dalje postoje neki problemi zbog koji je moguća nesreća iz krivih procjena algoritma kao npr. spajanje prometnih traka zbog čega je i dalje potreban ljudski nadzor. LaneNet je jedan od takvih algoritama. Algoritam koji se sastoji od dvije neuronske mreže kako bi se detekcija prometnih traka izvršila u dva različita scenarija [4]. Dok se jedna neuronska mreža koristi za detekciju rubova i predlaganje mjesta gdje se pikseli trake nalaze druga se koristi za određivanje oblika prema prijedlogu prve neuronske mreže. To se sve izvršava

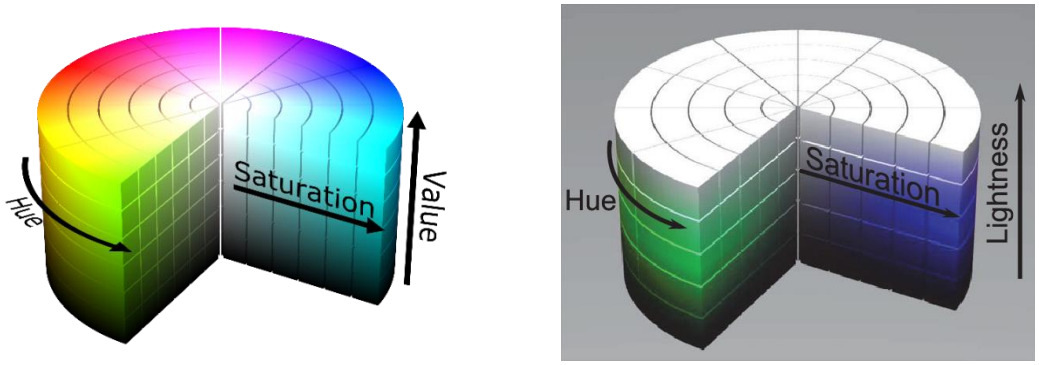
na inverznoj perspektivi slike (Slika.2.1). Ovo je jedan od trenutno popularnijih algoritama i većina novijih se zasniva na njemu.

a) b)

Slika 2.1. *Perspektive prometne trake: a) normalna, b) inverzna (ptičja)*

Druga vrsta algoritma koja se spominje je LaneRTD (engl. *Lane Real-Time Detection*) [5]. Prednost ovoga algoritma je korištenje u stvarnom vremenu i da je potrebna samo jedna kamera, ali je mana da nije precizan prilikom zakrivljene prometne trake ili značajno loših vremenskih uvjeta. Glavni proces ovoga algoritma je uklanjanje smetnji na fotografiji te detekcija rubova prometnih traka s Canny detekcijom rubova (engl. *Canny edge detection*) [6]. Canny detekcija rubova nam omogućava uklanjanje rubova koji nas „ne zanimaju“ kako bi uspjeli dobiti što bolji prikaz konture promatranog objekta. Većinom se primjenjuje na crno bijele slike s primijenjenim Gaussovim filterom [7] koji nam omogućava reduciranje smetnji. Važno je spomenuti da se primjenjuje Hughova transformacija [8] kako bi se pronašle koordinate prometne trake te iz rješenja funkcije moguće je uz dodatnih par metoda prikazati prometnu traku na ulaznom videu. Prema ovoj vrsti detekcije prometnih traka baziran je ovaj rad te će se detaljnije opisati funkcionalnosti.

Dodatno, mogu se spomenuti i neki značajniji prostori boja kako bi se prometna traka što bolje prepoznala s ulaznog videa kao što su HSV (engl. *Hue, Saturation, Value*) i HSL (engl. *Hue, Saturation, Lightness*). Prostori boja imaju zapis u obliku 3 kanala. H, S, V i H, S, L gdje H predstavlja ton boje, S zasićenje boje, V intenzitet boje, a L svjetlinu boje kao što možemo vidjeti na Slici 2.2



a) b)

Slika 2.2. *Prostor boja* [9]*: a) HSV*[1]*, b) HSL*[2]

Tijekom korištenja ovih prostora susreće se s definiranjem praga (engl. *threshold*) kako bi se što bolje izolirale prometne trake te uklonile moguće smetnje. U ovome radu je korišten HSL prostor kako bi se bolje iščitale prometne trake, te su kasnije detaljnije opisane mane ovoga pristupa.

[1] Preuzeto sa https://en.wikipedia.org/wiki/File:HSV_color_solid_cylinder_saturation_gray.png (04.09.2021)
[2] Preuzeto sa https://en.wikipedia.org/wiki/File:HSL_color_solid_cylinder_saturation_gray.png (04.09.2021)

3. ALGORITAM ZA DETEKCIJU PROMETNE TRAKE

3.1. INSTALACIJA POTREBNIH BIBLIOTEKA

Za rješavanje problema završnoga rada potrebno je koristiti tri biblioteke koje su podržane od strane programskog jezika Pythona. Za rješavanje i prepoznavanje prometne trake koriste se biblioteke OpenCV, NumPy i Matplotlib.

3.1.1. OpenCV biblioteka

OpenCV [10] (engl. *Open Source Computer Vision*) je biblioteka otvorenog koda koja sadrži programske funkcije za računalni vid i strojno učenje. Izgrađen je kako bi osigurao zajedničku infrastrukturu za aplikacije računalnog vida i ubrzao korištenje strojne percepcije u komercijalnim proizvodima. Biblioteka sadrži više od 2500 optimiziranih algoritama, među kojima je i mnogo najsuvremenijih algoritama u području računalnog vida i strojnog učenja. OpenCV se primjenjuje u prepoznavanju osoba preko kamere, praćenja ljudskih pokreta, prepoznavanja krajolika, praćenja objekata u pokretu, obavljanju određenih operacija na računalu uz pomoć kamere, itd.. Postoje implementacije za C++, Python i Java programske jezike uz podršku za Windows, Linux, Android i MacOS operacijske sustave. OpenCV se najviše orijentira prema aplikacijama koje informacije primaju u stvarnom vremenu.

3.1.2. NumPy biblioteka

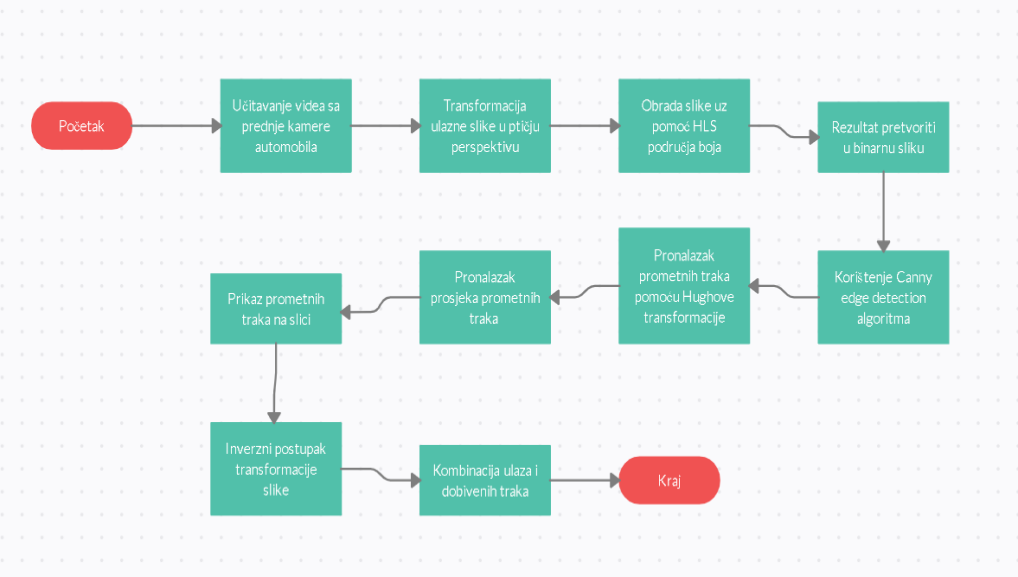
NumPy [11] je biblioteka otvorenog koda za znanstveno računanje u programskom jeziku Python. Sadrži podršku za velike, višedimenzionalne nizove i matrice s velikom zbirkom matematičkih funkcija visoke razine za rad na tim nizovima. Korištenje NumPy u Python-u omogućuje funkcionalnosti koje su usporedive s korištenjem MATLAB programa. NumPy se može koristiti kao višedimenzionalni spremnik generalnih podataka.

3.1.3. Matplotlib biblioteka

Matplotlib je biblioteka za crtanje unutar programskog jezik Python i za njegovo numeričko proširenje NumPy. pruža objektno orijentirani API za ugrađivanje plota u aplikacije pomoću GUI-a opće namjene kao Tkinter, wxPython, Qt ili GTK. U radu se koristi za pronalazak koordinata potrebnih za transformaciju perspektive ulaznog videa.

3.2. OPIS RADA ALGORITMA

Korišteni algoritam za detekciju prometnih traka je podijeljen u više manjih funkcija koje obavljaju razne operacije poput obrade ulaza do kreiranja linija koje su označene na rezultatu algoritma. Potrebno ga je podesiti da radi na fotografijama ili videu. LaneDetection.py je glavni program i u njemu se nalazi cijeli algoritam. U navedenom radu nije korištena kalibracija kamere jer je korišteno više video isječaka pronađenih na internetu, te se kamere razlikuju te nisu dostupne specifikacije. Na Slici 3.1 prikazan je dijagram toka algoritma koji će detaljnije biti objašnjen u nastavku.



Slika 3.1. *Dijagram toka*

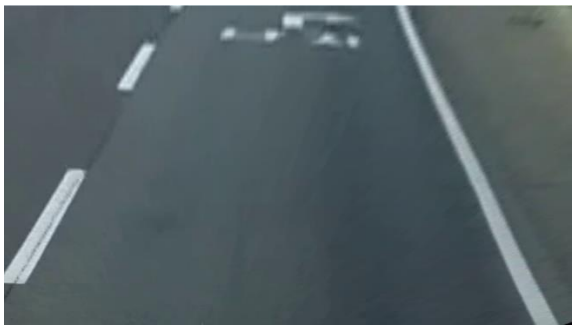
3.2.1. Transformacija perspektive slike

Kada golim okom promatramo objekte na većoj udaljenosti lako se primijeti iluzija da neke stvari izgledaju manje nego što zapravo jesu. Ovako nešto se najbolje može predočiti s putovanjem zrakoplovom. Iako je poznato da su zgrade velike tijekom leta one izgledaju minijaturno i dosta zbliženo, ovako nešto je zanimljivo za vidjeti ali prilikom detekcije prometnih traka to stvara samo problem. Prometne trake na učitanom videu pri velikoj udaljenosti izgledaju spojeno i takav problem se prvo treba riješiti za što bolju detekciju. Prilikom rješavanja problema korištene su funkcije iz OpenCV biblioteke `getPerspectiveTransform()` i `warpPerspective()` koje omogućuju jednostavnu transformaciju slike u drugu perspektivu.

Za korištenje metoda potrebno je odrediti SRC i DST koordinate potrebne za transformaciju. SRC koordinate označavaju 4 točke koje omeđuju područje koje se promatra, dok DST koordinate se koriste za razlučivost slike koja u ovome slučaju ostaje jednaka ulaznoj slici. Također, SRC koordinate su korištene u nastavku za određivanje područja interesa ROI (engl. *Region of interest*). Određivanje SRC koordinate postiže se funkcijom `imshow()` iz Matplotlib biblioteka koja prima ulaznu sliku te je moguće iz dobivenog prozora iščitati tražene koordinate u donjem kutu (Slika 3.2.)

Slika 3.2. *Određivanje SRC koordinata*

Nakon određivanja koordinata uvrštavaju se u spomenute funkcije i dobiva se rezultat kao na Slici 3.3.

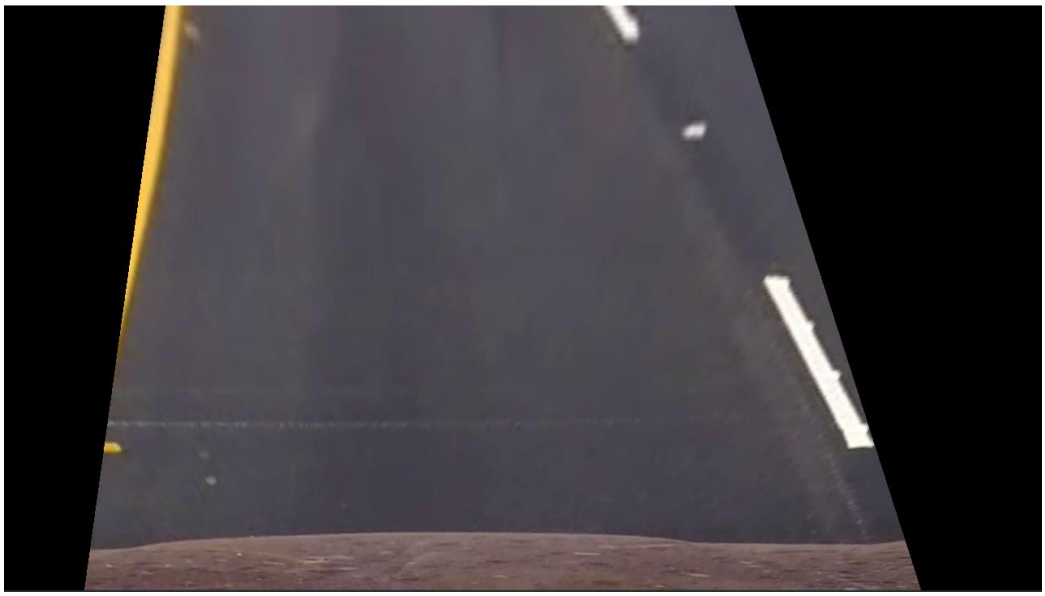


a) b)

Slika 3.3. *Persepktiva ceste: a) normalna, b) ptičja*

3.2.2. Područje interesa ROI

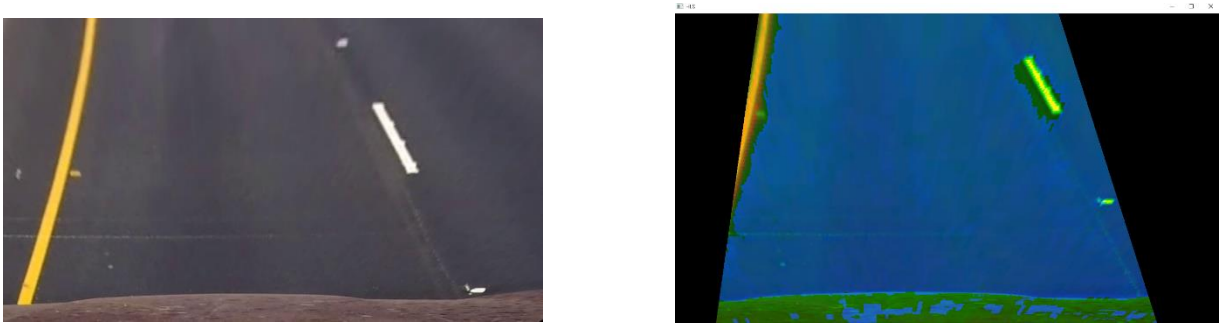
Na rezultatu transformacije perspektive može se prijeti da i dalje se neki dijelovi prikazuju koji nisu potrebni pa je napravljen mali algoritam kako bi se označilo područje interesa. Algoritam se sastoji od dodatnih točki određenih uz pomoć Matplotlib biblioteke i dodatno se koristi OpenCV biblioteka kako bi se ispunio trapez omeđen tim točkama. Funkcija za crtanje trapeza je `fillpoly()` koja se primjenjuje radi stvaranja maske. Maska se stvara uz pomoć funkcije `bitwise_and()` između omeđenog trapeza i ulazne slike, a rezultat je dodatno omeđeno područje koje nam ostvaruje manje smetnji (Slika 3.4.).



Slika 3.4. *Područje interesa*

3.2.3. Filtriranje slike uz pomoć HSL područja boja

U početku filtriranja prvo se određuje način na koji prepoznati prometne trake i te se algoritam definira prema tome. Glavna stvar preko koje se to prepoznaje je njihova boja. Poznato je da prometne trake mogu biti bijele i žute boje te na to treba obratiti pozornost tijekom korištenja filtera. Kao filter za ovaj rad je korišteno HSL područje boja tako što smo ulaznu sliku u RGB području pretvorili u HSL (Slika 3.5.). Ovaj način je odabran jer HSL područje dobro prikazuje svjetlinu slike i lako je izolirati prometne trake. Za uklanjanje što većeg broja smetnji i za korištenje metode detekcije rubova Canny trebamo sliku prebaciti u binarno područje (Slika 3.6). Kako bi se to postiglo prvo su definirani rasponi za bijelu i žutu boju u HSL području i od njih su stvorene maske nad slikom u HSL području. Za ovo su korištene funkcije `inRange()` za definiranje raspona boja. Važno je napomenuti da se različite boje moraju tražiti zasebno i da se dobro odabere raspon za detekciju određenih boja jer pri lošem odabiru neće doći do pouzdane detekcije. Na Slici 3.7. moguće je vidjeti rezultate pojedine maske i njihova kombinacija funkcijom `bitwise_and()` iz OpenCV biblioteke. Kao što se primijeti na Slici 3.7 d) žuta maska ima dobro očitanje ali pokupila je par stvari koje nam nisu značajne dok je bijela lošije očitala zbog loše kvalitete slike. Problem bijele trake se može riješiti prilagođavanjem raspona detekcije bijele boje ali nije poželjno. Na slici bi vidjeli poboljšanje vidljivosti bijele trake, ali bi se učitalo i dosta dodatnih nebitnih stvari. Drugi problem je previše učitanih stvari koje su nepotrebne za detekciju traka i mogu samo stvoriti greške pri detekciji. Ovo se rješava s već spomenutim pojmom područja interesa ROI.

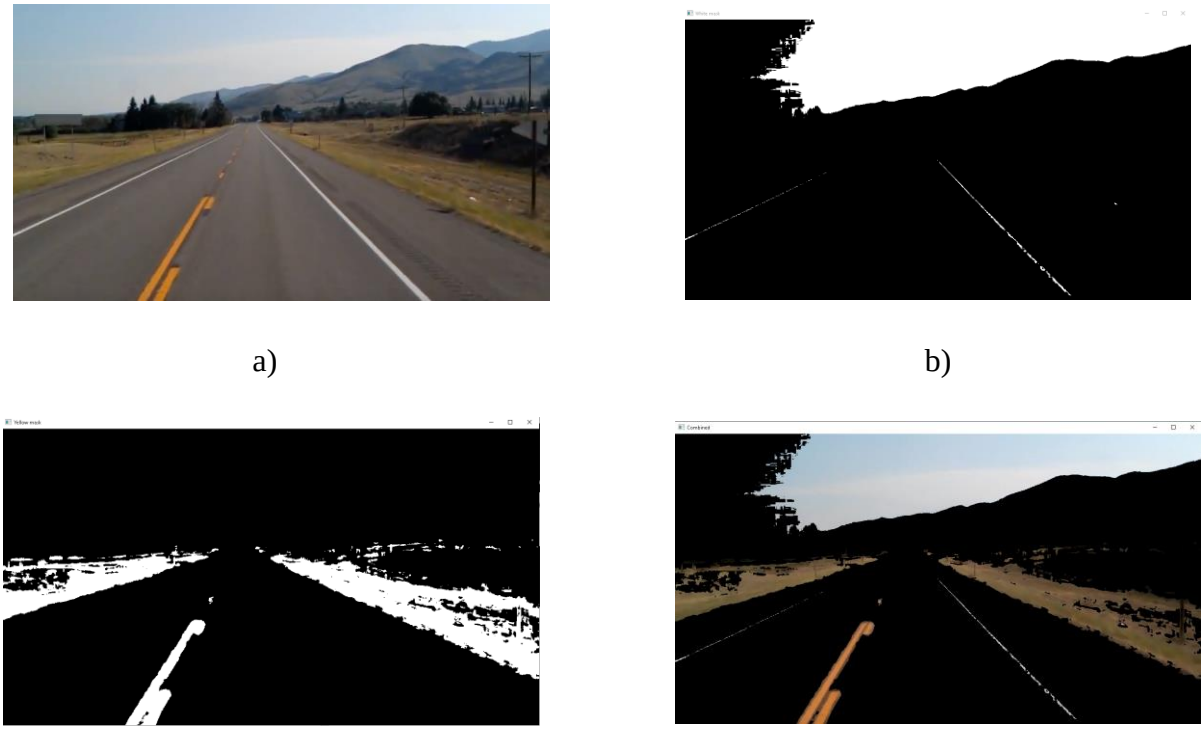


a) b)

Slika 3.5. *Prikaz ceste: a) RGB područje, b) HSL područje*

a) b)

Slika 3.6. *Prikaz ceste: a) normalni, b) binarni*



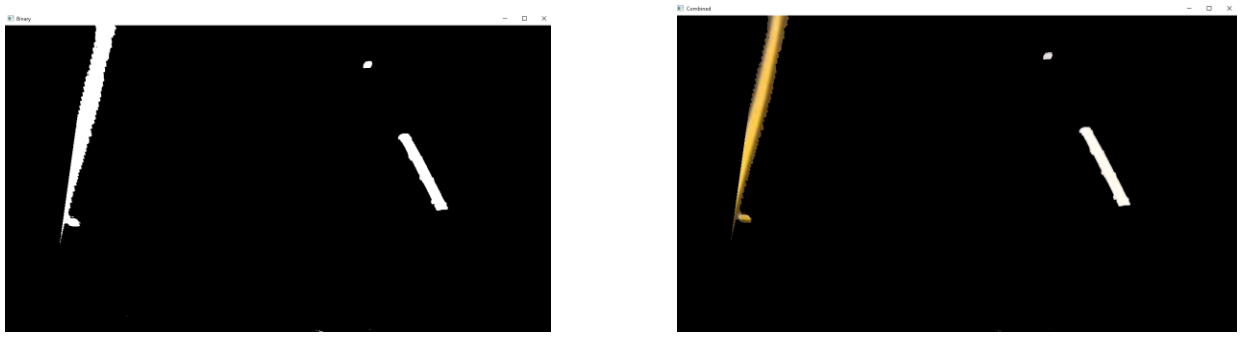
a) b)

c) d)

Slika 3.7. *Prikaz ceste: a) normalan, b) bijela maska, c) žuta maska, d) kombinacija b) i c)*

Nakon što su slike kombinirane i područje interesa određeno, kreiran je algoritam koji sliku prenosi u binarno područje. Kako bi se to postiglo prvo je odvojen L kanal tj. kanal svjetline u kombiniranoj slici te svaka vrijednost novonastale matrice provjerava je li vrijednost elementa u rasponu od [1,255]. Također se koristi funkcija `zero_like()` koja stvara matricu s nulama

prema obliku izdvojenog kanala. Ako se vrijednost nalazi unutar raspona dodjeljuje se vrijednost 255 što nam daje potpuno osvjetljenje tj. bijelu boju u novoj matrici s nulama. Rezultat algoritma je binarna slika (Slika 3.8.).

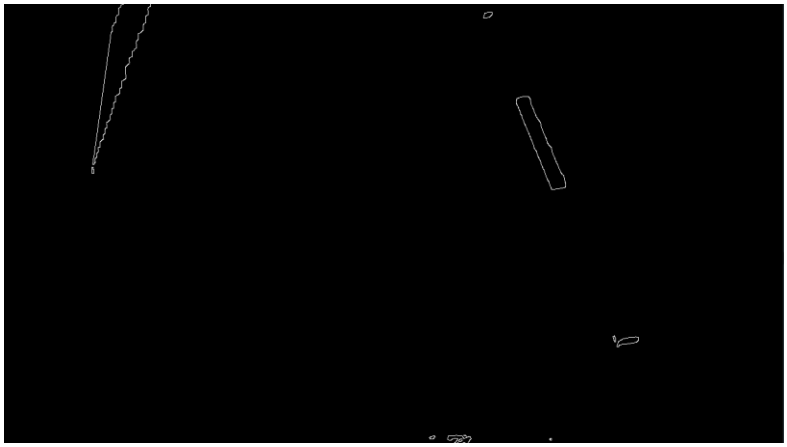


a) b)

Slika 3.8. *Prikaz ceste: a) binarno b) kombinacijom maski*

3.2.4. Detekcija rubova sa Canny algoritmom

U idućem koraku je primijenjen Gaussov filter za uklanjanje potencijalnih smetnji pa se koristi Canny algoritam. Može se reći da nije bilo potrebe za korištenjem Gaussovog filtera jer ga i sam Canny koristi, ali za što precizniju detekciju je korišten. Canny algoritam je jedan od boljih algoritama za detekciju rubova zbog efikasnosti i jednostavnosti, te se na slikama lako prepoznaje smjer detektiranih rubova. Omogućuje biranje granica za detekciju rubova, uklanja veliki broj nebitnih rubova, dok promatrani objekt ostaje lako prepoznat i prepoznaje rubove koji su najvjerojatnije povezani iako se nalaze iznad granica. Algoritam se nalazi u OpenCV biblioteci te se lako poziva uz pomoću naredbe `Canny()`. Za parametre predajemo obrađenu sliku te granice unutar kojih će detektirati rubove na binarnoj slici, a kao rezultat vraća sliku s detektiranim rubovima gdje dolazi do jače promjene gradijenta. Slika 3.9. prikazuje rezultat uporabe algoritma na području interesa.



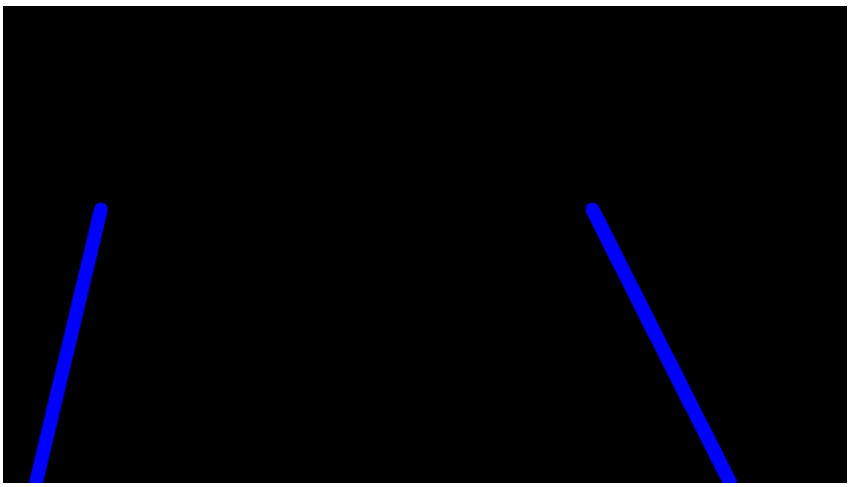
Slika 3.9. *Primjena Canny algoritma*

3.2.5. Pronalazak traka uz pomoć Houghove transformacije

Za pronalazak prometnih traka korišten je algoritam `HoughLinesP()` iz OpenCV biblioteke. On omogućava pronalaženje najboljeg pravca (engl. *best-fit*) koji opisuje prometnu traku. Inače se algoritam koristi za bilo kakvu detekciju oblika, ako se taj oblik može prikazati matematičkim zapisom. Algoritam pronalazi pravac koji siječe najviše točaka u određenom broju piksela kao bi se točno odredio smjer prometnih traka. Tijekom definiranja funkcije važno je definirati broj piksela za prihvaćanje točaka koje sijeku pravac, preciznost u stupnjevima, maksimalnu razmak između pravaca i minimalnu duljinu pravca. Kada su definirani ti parametri dobivaju se određeni pravci koji će predstavljati naše prometne trake. Nakon svega ovoga dobiva se veliki broj pravaca koji opisuju prometne trake tj. više se pravaca pojavljuje na prometnim trakama odjednom. Ovaj problem je riješen s funkcijom `averageLaneInterception()` u kojoj se razvrstavaju pravci s obrađene slike. Razvrstavanje se vrši tako da se provjeravaju pojedine koordinate pravca dobivene `HoughLinesP()` algoritmom te se određuje njihov nagib. Prema njihovim nagibima sortirani su u nizove za lijevi ili desni pravac. Nakon sortiranja još je preostalo da se nađe prosječni pravac što se postiže funkcijom `average()` iz biblioteke NumPy. Pravac se kreira funkcijom `makeCord()` gdje se definiraju njegove nove vrijednosti. Još je ostalo prikazati te pravce na slici.

Kako bi se pravci prikazali na slici kreira se funkcija `DisplayLines()` s parametrima slike i izračunatih prosječnih linija. Funkcija stvara matricu popunjenu nulama u obliku ulazne slike i rastavlja pravac na koordinate. Provjerava se postojanje pravaca i dali su koordinate realne

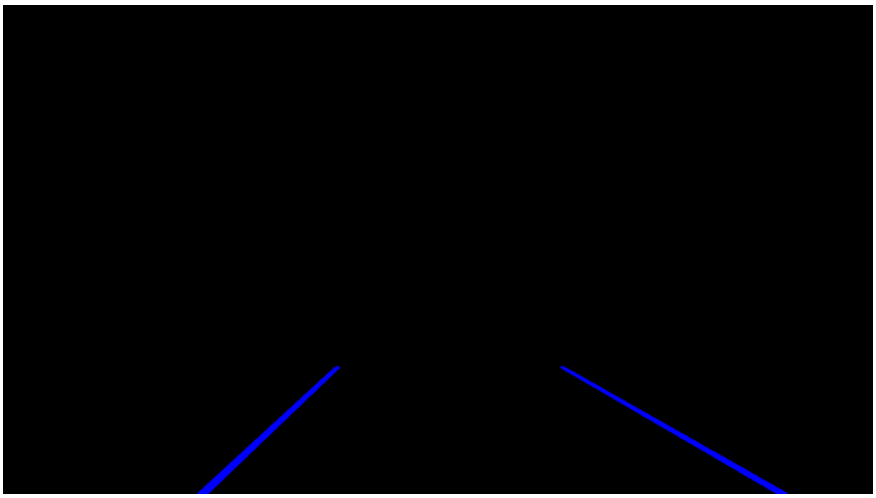
vrijednosti kako ne bi došlo rušenja programa. Nakon provjere koristi se OpenCV funkcija `lines()`, ona omogućava crtanje pravaca na kreiranoj matrici što je ujedno i rezultat funkcije (Slika 3.10.).



Slika 3.10. *Prikaz prometnih traka na crnoj fotografiji*

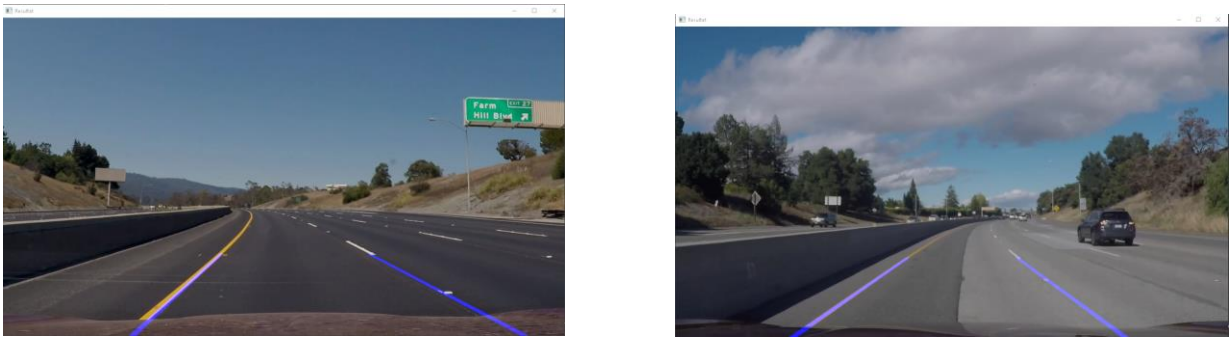
3.2.6. Inverzna transformacija i spajanje rezultata

Kako bi dobili konačni rezultat preostalo je vratiti sliku s prometnim trakama u normalnu perspektivu. Postupak transformacije u normalnu perspektivu je jako sličan postupku za inverznu transformaciju perspektive samo je razlika u tipu zastavice metode `warpPerspective()` gdje se postavlja na `WARP_INVERSE_MAP` i time se postiže željena perspektiva (Slika 3.11.)



Slika 3.11. *Normalna perspektiva prometnih traka na crnoj fotografiji*

Za kraj ostaje samo spojiti ulaznu sliku s filtriranom slikom prometnih traka. `addWeighted()` funkcija omogućava upravo to i ona također pripada OpenCV biblioteci. Potrebno joj je predati ulaznu fotografiju, osvijetljene, pa filtriranu fotografiju i osvjetljenje i u konačnici dobijemo rezultat (Slika 3.12.).



Slika 3.12. *Prikaz prometnih traka na ulaznom videu*

4. EVALUACIJA KORIŠTENOG ALGORITMA ZA DETEKCIJU PROMETNE TRAKE

Prilikom testiranja algoritma korišteno je više video primjeraka i nisu mijenjane postavke u ovisnosti o videima. U nastavku se govori o prednostima i nedostacima algoritma, te pitanju dali je algoritam moguće primijeniti u stvarnom vremenu. Prvo se govoriti o ispravnost prometnih traka, a poslije njegovoj brzini.

4.1. Ispravnost prometnih traka

Za prvi testirani video test.mp4, algoritam detekcije je najbolje prilagođen kao što se primijeti iz primjerenih slika (Slika 4.1.). Iako je vidljivo da se algoritam izvršava kako je željeno mogu se primijetiti neke greške u malim trenutcima gdje očitanje nije savršeno. Do ovoga dolazi zbog malih sijena te je to moguće korigirati s promjenom praga tijekom filtriranja slike. Iz slike se vidi kako algoritmu mala zakrivljenija ne predstavljaju problem. Tijekom promatranja detekcije prometnih traka mjeren je broj okvira po sekundi FPS (engl. *frame per second*). Iznos FPSa je raspon od [19,27] dok je prosjek bio 24 FPSa. Mjerenje je izvršeno unutar konzole.

Slika 4.1. *Rezultati test.mp4 prometne ceste*

Za drugi video koristio se test2.mp4 u kojem automobil prolazi ispod mosta gdje se stvara velika sjena. Tijekom testiranja algoritma očitavanje prometnih traka je uredu sve dok se ne dođe ispod mosta gdje lijeva traka nestaje ali se sve vrača u normalu prilikom izlaska iz sjene kao što prikazuje Slika 4.2. Iz ovoga se zaključuje da algoritam nije dobar prilikom loše vidljivosti prometnih traka. Da bi se to popravilo potrebno je smanjenje praga za očitavanje traka, ali to bi samo više naštetilo detekciji nego što bi pomoglo. Mjereni FPS u ovome videu je u rasponu od [10, 31] dok je prosjek 25.

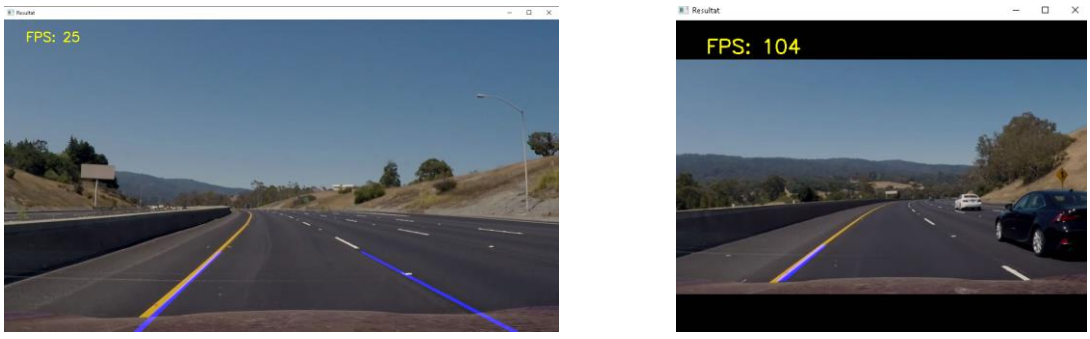
Slika 4.2. *Rezultati test2.mp4 prometne ceste*

Treći video test3.mp4 je najteži test do sada. U videu automobil se vozi jako zakrivljenom cestom i nije sniman prednjom kamerom automobila.. Rezultate se vidi na Slici 4.3. U većem dijelu videa neispravno se očitavaju prometne trake. Prilikom manjih zakrivljenija dolazi do djelomičnog očitanja, ali zato prilikom većega se u potpunosti raspada. U ovome primjeru algoritam nije koristan u većini slučajeva i potrebna su poboljšanja za zakrivljene prometne trake. Mjereni FPS je u rasponu od [18, 28] dok je prosjek 23.

Slika 4.3. *Rezultati test3.mp4 prometne ceste*

4.2. Mjerenje brzine algoritma

Kada govorimo o brzini algoritma važno je prvo spomenuti da je algoritam mjeren na laptopu s procesorom Intel Core i5 7300 HQ i 8 GB RAM memorije. Te specifikacije su dosta doprinijele brzini rada algoritma kao što se primjećuje iz prethodno priloženih podataka i prema tome je moguće izvršavanje u stvarnom vremenu, ali generalno, to nije realno. Moramo pretpostaviti da trenutni automobili neće imati takve specifikacije te pokušati poboljšati algoritam. Jedan od načina kako bi se to riješilo u ovome slučaju je da se smanji rezolucija slike. To bi nam vremenski smanjilo obradu slike te povećalo brzinu rada algoritma za nekoliko puta. Iz usporedbe brzine rada algoritma (Slika 4.4.) pri različitim rezolucijama može se primijeti da pri smanjenoj vremenu obradi slike zaista dobivamo na brzini algoritma.



a) b)

Slika 4.4. *Rezolucija videa: a) 1280x720 b) 640x480*

5. ZAKLJUČAK

U ovome radu, cilj je bio kreirati algoritam za detekciju prometnih traka preko prednje kamere automobila. Rad je ovisio o značajkama sa slike i linijskom modeliranju prometnih traka. Za rješenje algoritma korištene su tri biblioteke Matplotlib, OpenCV i NumPy u Pythone 3.9.6 programskom jeziku. Većinski dio programa se odnosio na obradu slike preko par različitih filtera. Nakon obavljenog testiranja, možemo vidjeti da algoritam ne daje najbolje rezultate u svim situacijama. Algoritam najbolje radi na ravnoj prometnoj traci i nije osjetljiv na mala zakrivljenija dok pri velikima daje neispravna očitavanja prometnih traka. Prilikom nailaska na prikrivenija područja algoritam djelomično gubi funkcionalnost. Brzina rada algoritma je u prosjeku 24 FPS pri rezoluciji od 1280x720 te je pogodan za rad u stvarnom sustavu s boljom opremom ili lošijem sa smanjenom rezolucijom. Pri smanjenoj rezoluciji vidimo tri puta veće povećanje FPSa što nam govori da su manje rezolucije pogodnije za rad ovakvom tipu algoritma. Algoritam je ispravan u nekim situacijama te ima velikih poteškoća u drugima težim uvjetima na cesti gdje ne može ispravno odrediti detekciju tako da možemo reći da nije spreman za uporabu. Najbolje mjesto gdje bi se mogao primijeniti ovakav algoritam je isključivo na ravnim cestama, a. sve drugo nije prihvatljivo. Tako da bi zaključak bio, kako bi poboljšali algoritam potrebno je koristiti naprednije metode za kreiranje prometnih traka, poput istreniranih neuronski mreža koje bi znatno ubrzale rad i točnost algoritma jer novija tehnologija najbolje se prilagođava težim uvjetima.

POPIS KRATICA

ADAS	Advanced driver-assistance systems	Napredni sustavi pomoći vozaču
LaneRTD	Lane real time detection	Detekcija linija u stvarnom vremenu
OpenCV	Open source computer vision	Računalni vid otvorenog koda
HSL	Hue, saturation, lightness	Ton, zasićenje, svjetlina boje
ROI	Region of interest	Područje interesa
FPS	Frame per second	Okvir po sekundi
API	Application programming interface	Aplikacijsko programsko sučelje
GUI	Graphical user interface	Grafičko korisničko sučelje
RGB	Red, green, blue	Crvena, zelena, plava

LITERATURA

[1] "BILTEN O SIGURNOSTI CESTOVNOG PROMETA 2019." Sep. 04, 2021. [Online]. Available: https://mup.gov.hr/UserDocsImages/statistika/2020/Pokazatelji%20javne%20sigurnosti/bilten_promet_2019.pdf

[2] "What is ADAS?," *Synopsys*, Sep. 04, 2021. https://www.synopsys.com/automotive/what-is-adas.html

[3] D. Špoljar, "Detekcija voznih traka ispred vozila pomoću kamere i upozoravanje na nepoželjno napuštanje trake." Sep. 04, 2021. [Online]. Available: https://repozitorij.etfos.hr/islandora/object/etfos%3A2462/datastream/PDF/view

[4] Ze Wang∗, Weiqiang Ren, Qiang Qiu, "LaneNet: Real-Time Lane Detection Networks for Autonomous Driving." Sep. 04, 2021. [Online]. Available: https://arxiv.org/pdf/1807.01726.pdf

[5] Wael Farag, Zakaria Saleh, "Road Lane-Lines Detection in Real-Time for Advanced Driving Assistance Systems." IEEE, Sep. 04, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/8855797

[6] "Canny edge detection," *OpenCV*, Sep. 04, 2021. https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html

[7] "Gaussian blur," *Science Direct*, Sep. 04, 2021. https://www.sciencedirect.com/topics/engineering/gaussian-blur

[8] "Hough Line Transform," *OpenCV*, Sep. 04, 2021. https://docs.opencv.org/4.5.1/d6/d10/tutorial_py_houghlines.html

[9] "HSL and HSV," *Wikipedia*, Sep. 04, 2021. https://en.wikipedia.org/wiki/HSL_and_HSV

[10] "OpenCV," Sep. 04, 2021. https://opencv.org/about/

[11] "NumPy," Sep. 04, 2021. https://numpy.org/about/

SAŽETAK

Problem ovoga rada je bio opis rada i kreiranje algoritma za detekciju prometnih traka. Za kreiranje algoritma se koriste OpenCV, Matplotlib i NumPy biblioteke. Algoritam je opisan u par koraka, od kojeg je najvažnija obrada slike. Slici je najbitnije prvo transformirati perspektivu, nakon toga slika se obrađuje pomoću HSL područja boja te se time reduciraju smetnje. Rezultat tih funkcija je binarna slika. Primjenom Canny algoritma detektiraju se rubovi slika kojom se završava obrada, nakon toga se pronalaze pravci koji predstavljaju prometne trake uz pomoć Houghove transformacije. Iz dobivenih pravaca stvara se jedan prosječni pravac kako bi se što bolje prometne trake prikazale. Na kraju, vraća se filtrirana slika u normalnu perspektivu i spaja se s ulaznom slikom. To je ukratko opis algoritma, a nakon toga se izvode testiranja. Takav algoritam je dovoljno brz za izvođenje u stvarnom vremenu, ali nije dovoljno efektivan za uporabu u stvarnom prometu.

Ključne riječi: detekcija prometnih traka, HSL područje boja, OpenCV, računali vid

LANE DETECTION

ABSTRACT

The problem of this paper was the description of the work and the creation of an algorithm for the detection of traffic lanes. OpenCV, Matplotlib and NumPy libraries are used for creation of the algorithm. It is described in a few steps, the most important of which is image processing. First, it is essential to transform perspective. After that, the image is processed using the HSL color space and the noise is reduced with that. This results in a binary image and the application of the Canny edge detection algorithm. To describe the image processing, routs representing traffic lanes are then found using the Hough transform. From the obtained routs, one average rout is created in order to show the best possible traffic lanes. Finally, we return the filtered image to normal perspective and merge it with the input image. This is a brief description of the algorithm, after which tests are performed. The algorithm is fast enough to run in real time, but not efficient enough for use in real traffic.

Keywords: lane detection, HSL color space, OpenCV, computer vision

ŽIVOTOPIS

Bruno Šimunović je rođen 20.12.1999 u Vinkovcima. Završio je tehničku gimnaziju u tehničkoj školi Ruđera Boškovića u Vinkovcima s vrlo dobrim uspjehom. Poslije srednje škole upisao fakultet Elektrotehnike, Računarstva i Informacijskih tehnologija u Osijeku, smjer Računarstvo. Poznaje engleski jezik u govoru i pismu.

PRILOZI

1) Programski kod opisanog algoritma sa svim potrebnim videima
    - https://github.com/FizzyShadow/Lane-detection.git (04.09.2021)
2) Skup video snimki korišten za testiranje rada algoritma.