Procjena brzine vozila na temelju snimke kamere koja zamjenjuje retrovizor

Ćosić, Luka

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

Sveučilišni studij

PROCJENA BRZINE VOZILA NA TEMELJU SNIMKE KAMERE KOJA ZAMJENJUJE RETROVIZOR

Diplomski rad

Luka Ćosić

Osijek, 2018.

Sadržaj

1. UVOD

S razvojem tehnologije stvaraju se i napredni sustavi koji su namijenjeni automatizaciji određenih procesa, kako bi se minimizirala mogućnost ljudske greške. Takvi sustavi nisu zaobišli ni automobilsku industriju u kojoj se sve više koriste napredna rješenja, kako bi se vožnja automobilom učinila što je više moguće komfornijom, sa ciljem smanjenja prometnih nesreća. Napredni sustavi za pomoć vozaču (engl. *Advanced Driver-Assistance Systems - ADAS*) su sustavi koji, koristeći se različitim senzorima, dobivaju informacije iz okoline. Informacije koje se senzorima dobivaju iz okoline obrađuju se na automobilskom računalnom sustavu. U ovom radu bit će opisan sustav koji prati stanje iza vozila u kretanju, a koji se temelji na analizi video signala dobivenih preko kamera koje zamjenjuju retrovizorska zrcala. Prvi korak pri rješavanju ovog problema predstavlja zaključivanje koji od objekata koji se pojavljuju u video snimci predstavljaju relevantne objekte, a koje od njih treba zanemariti. Način zaključivanja koji od objekta je relevantan može se razlikovati od algoritma do algoritma. Tako neki koriste prepoznavanje prometne trake i u njoj pretražuju vozila, dok neki koriste oduzimanje pozadine kako bi dobili objekt od interesa, itd. Broj okvira koje algoritam može obraditi u sekundi (engl. *frame per seconds - FPS*) je jedan od ključnih parametara ovog algoritma. Nakon prepoznavanja relevantnih objekata slijedi računanje brzine prepoznatog objekta te se na temelju toga vrši odlučivanje smije li vozač promijeniti prometnu traku ili ne smije.

Algoritam koji će biti opisan u ovom radu vrši prepoznavanje vozila s okvira video snimke te se prepoznati objekt koristi za daljnju analizu. Nakon prepoznavanja slijedi proračun brzine objekta koja može biti manja, veća ili jednaka od vozila na kojem se nalazi kamera. Izračunavanjem brzine vozila u retrovizoru algoritmu se omogućuje odlučivanje kako treba postupiti prilikom promjene prometne trake (npr. promijeni prometnu traku, nemoj promijeniti prometnu traku, itd.), a u toj odluci ne smije biti mjesta za pogrešku jer pogrešna odluka može dovesti do ljudskih žrtava. Rezultati algoritma su: procijenjena udaljenost nadolazećeg vozila, procijenjena brzina nadolazećeg vozila i vrijeme koje je preostalo da nadolazeće vozilo dođe do mjesta kamere.

U drugom poglavlju opisana su postojeća rješenja koja se bave detekcijom vozila i proračunom brzine vozila. Treće poglavlje opisuje rješenje za procjenu brzine nadolazećeg vozila vidljivog u retrovizoru stvorenog u sklopu ovog diplomskog rada, a u četvrtom poglavlju predstavljena je evaluacije stvorenog rješenja. Peto poglavlje donosi zaključke rada.

2. PROCJENA BRZINE VOZILA U VIDEO SEKVENCI

Postoje razna rješenja za procjenu brzine vozila iz video snimke, a svako od tih rješenja ima svoju primjenu. Zbog postojanja mnoštva gotovih rješenja za procjenu brzine, na početku ih je potrebno proučiti i spoznati što je moguće više problema koji mogu utjecati na pogrešku procjene, s ciljem smanjenja pogreške i povećavanja učinkovitosti vlastitog algoritma. Algoritam za procjenu brzine vozila iz video snimke može naći različite primjene, zbog čega se radovi koji rješavaju navedeni problem često razlikuju te na drugačiji način vrše procjenu. U nastavku će biti opisano nekoliko postojećih rješenja.

Prvo od promatranih postojećih rješenja [1] polazi od toga da se kamera nalazi pričvršćena za most iznad ceste te se iz snimke s te kamere vrši proračun brzine u nekoliko koraka. Početni korak algoritma je pronaći točku u kojoj se spajaju sve linije ceste, a linije ceste se izdvajaju pomoću *Canny edge* algoritma (algoritam koji pronalazi rubove na slici). Nakon pronalaska točke nestajanja (engl. *vanishing point*) vrši se transformacija slike u ptičju perspektivu, u obzir se uzima samo dio slike koji je interesantan, tj. cesta. Algoritam koristi oduzimanje pozadine (engl. *background subtraction*) kako bi pronašao vozila na slici. Prije nego algoritam krene u otkrivanje vozila, potrebno je stvoriti pozadinsku sliku. Sustav boja koji se koristi u algoritmu je RGB te se za svaki kanal definira srednja vrijednost intenziteta i standardno odstupanje od srednje vrijednosti. Potom se vrši uzimanje 4 uzastopne okvira iz video snimke iz kojih se stvara pozadinska slika, tako da se zanemaruju svi elementi slike (engl. *pixels)* koji se ne nalaze u definiranom intervalu boja. Nakon što je stvorena pozadinska slika, za svaki slijedeći okvir videa vrši se sljedeće: binarizacija slike s definiranim pragom, morfološko zatvaranje kako bi se dobile cjeline, otkrivanje grumenčića koji predstavljaju vozila (engl. *detect blobs*). Otkrivena se vozila navedenim postupkom prate sve dok ne izađu iz scene. Normalizirana kros-korelacija koristi se za praćenje svih otkrivenih vozila. Udaljenost između dviju linija na cesti i pomak vozila između dva video okvira dovoljni su za procjenu brzine vozila. Prednosti algoritma su u tome što omogućuje praćenje vozila koja mijenjaju prometnu traku i onih koji se kreću velikom brzinom. Nedostatak algoritma je u tome što ne može raspoznati dva nedovoljno udaljena vozila, odnosno ne prepoznaje gdje jedno vozilo završava, a drugo počinje [1].

Drugo proučavano rješenje [2] smješta kameru pored ceste te snima cestu s koje se vrši otkrivanje (engl. *detect*) vozila te potom proračun njegove brzine. Navedeni algoritam je podijeljen u nekoliko koraka [2]:

- Postupak predobrade (engl. *preprocessing*),

- Modeliranje pozadine (engl. *background modelling*),
- Otkrivanje objekta (engl. *foreground detection*),
- Provjera valjanosti podataka (engl. *data validation*),
- Razlika između video okvira (engl. *inner frame difference*) i
- Procjena brzine (engl. *speed estimation)*

Postupak predobrade koristi se za prostorno izglađivanje video okvira kako bi se uklonio šum kamere ili kako bi se smanjio utjecaj vremenskih uvjeta (kiša, snijeg, itd.). Ovaj algoritam kao i prethodno navedeni koristi oduzimanje pozadine kako bi otkrio objekte od značaja, ali je razlika u tome što se u ovom algoritmu oduzimanje pozadine vrši na binarnim slikama, odnosno slikama u kojim svaki element slike može imati vrijednost ili „1“ ili „0“. Dobro modelirana pozadina omogućuje bolje otkrivanje objekata od značaja te postupak modeliranja pozadine predstavlja najbitniji korak u postupku oduzimanja pozadine, jer se pomoću modelirane pozadine može načiniti sustav koji je robustan na promjene u pozadini, ali dovoljno osjetljiv za otkrivanje objekata od interesa. Oduzimanjem prethodno modelirane pozadine dobivaju se kandidati za objekte od interesa. Dobiveni se kandidati provjeravaju i zadržavaju se samo oni koji mogu predstavljati objekte od interese te se tako smanjuje greška algoritma. Kako bi se sa sigurnošću moglo utvrditi radili se o pokretnom ili nepokretnom objektu, koristi se razlika između video okvira koja je osjetljiva na zadani prag i ako je vrijednost između dva uzastopna video okvira veća od zadanog praga radi se o pokretnom objektu. Nakon utvrđivanja koji od otkrivenih objekata je vozilo, vrši se računanje brzine istog. Za svaki otkriveni objekt računa se prijeđeni put u elementima slike. Informacija o prijeđenom putu u elementima slike mora biti zabilježena kao polje iste veličine kao što je veličina video okvira, kako bi imali informaciju o elementima slike na specifičnim koordinatama u okviru. Pomoću navedene informacije i brzine izmjene okvira (engl. *frame rate*) moguće je procijeniti brzinu vozila. Navedeni algoritam procjenjuje brzinu vrlo točno s minimalnom pogreškom [2].

Treće od promatranih rješenja [3] temelji se na transformaciji ulaznih video okvira u binarne slike te se nakon transformacije vrši procjena brzine vozila. Okviri videa su slike u sivim tonovima (engl. *grayscale*). Svaki okvir je dimenzija $M \times N$ elemenata slike i svaki okvir je označen rednim brojem i. Ovaj algoritam ne vrši detekciju vozila nego su predefinirana dva detekcijska područja, ulazno detekcijsko područje i izlazno detekcijsko područje. Detekcijska područja su postavljena na svakom od ulaznih video okvira i nalaze se na istom mjestu unutar slike. Svaki se od ulaznih video okvira transformira u binarnu sliku neovisno o prethodnim video okvirima. Transformacija u binarnu sliku ne obavlja se na svakom elementu slike zasebno nego se uzimaju u obzir i susjedni

elementi slike. Za svaki element slike uzimaju se veličine gradijenata (dva u horizontalnom smjeru i dva u dijagonalnom smjeru). Slijedi utvrđivanje maksimalne vrijednosti svakog od gradijenata te usporedba s unaprijed postavljenim pragom. Takav se proces obavlja za sve elemente slike osim rubnih. Ovim postupkom se ne dobiva standardna binarna slika nego se dobiva slika u kojoj svaki element slike predstavlja rub ukoliko ima vrijednost „1“, odnosno nazivaju se rubni elementi. Detekcijska područja mogu biti u dva stanja: slobodno (engl. *free*) i zauzeto (engl. *occupied*). Stanje detekcijskog područja se određuje na temelju aritmetičke sume rubnih elemenata u definiranom području. Ukoliko je suma rubnih elemenata veća od vrijednosti postavljenog praga, stanje područja se mijenja iz slobodnog u zauzeto stanje. Područje prelazi iz stanja zauzeća u slobodno stanje ukoliko je suma rubnih elemenata manja od vrijednosti postavljenog praga. Navedeno vrijedi za oba područja. Nakon što područje prelazi iz slobodnog stanja u stanje zauzeća pamti se redni broj video okvira koji će kasnije služiti za proračun brzine. Nakon što vozilo prođe kroz ulazno detekcijsko područje, stanje područja se mijenja i pamti se redni broj video okvira $i_{(ini)on}$, redni broj video okvira u kojem je vozilo prošlo kroz ulazno detekcijsko područje. Prolaskom vozila kroz izlazno detekcijsko područje mijenja se stanje tog područja te se također pamti redni broj video okvira u kojem se to dogodilo, $i_{(fin)on}$, redni broj video okvira u kojem je vozilo prošlo kroz izlazno detekcijsko područje, te slijedi proračun brzine iz pomoću brzine izmjene okvira f, udaljenosti detekcijskih područja d, formula (2-1) [3]:

$$v = \frac{fd}{i_{(fin)on} - i_{(ini)on}} \tag{2-1}$$

Ukoliko je brzina izmjene okvira dana u okvirima u sekundi i udaljenost između detekcijskih područja u metrima, brzina se dobiva u metrima u sekundi. Testiranje je provedeno kamerom koja snima 30 okvira u sekundi. Algoritam se temelji na jednostavnim računskim operacijama, što ga čini vrlo brzim. Točnost procjene brzine ovom metodom se smanjuje povećanjem brzine vozila. Korištenjem kamera koja snima 60 okvira u sekundi povećava se točnost algoritma.

Dodatna postojeća rješenja koja se bave problemima procjene brzine mogu se pronaći u [4 – 7].

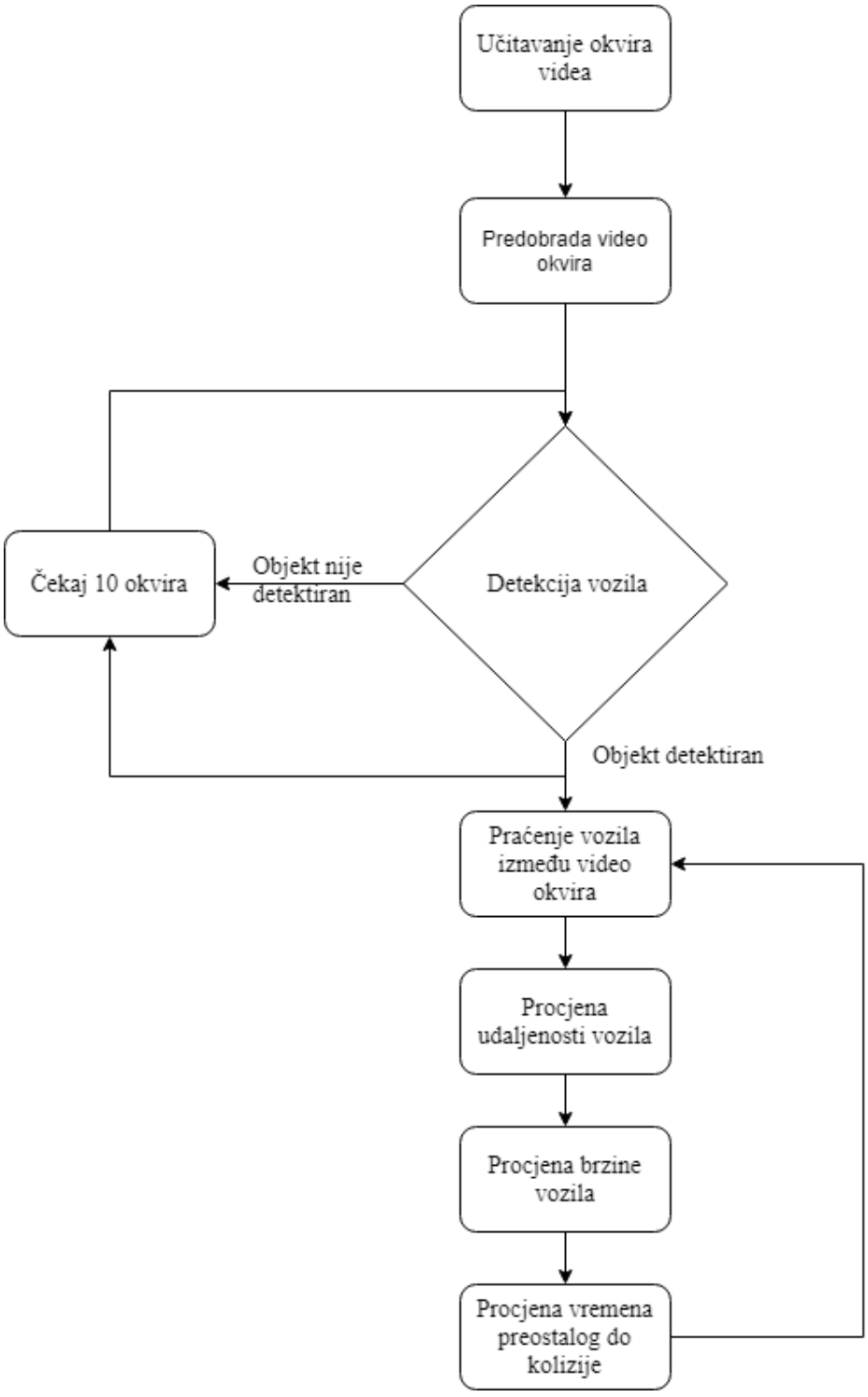
3. ALGORITAM ZA PROCJENU BRZINE VOZILA VIDLJIVOG U RETROVIZORU

Procjena brzine vozila koje je vidljivo u retrovizoru pomaže vozaču prilikom izmjene prometnih traka na cesti. Na temelju informacije o procijeni brzine vozila koje dolazi strage, vozač može procijeniti smije li ili ne smije promijeniti prometnu traku. Procjena brzine obavlja se na temelju video snimke kamere koja se nalazi na retrovizoru vozila. Kamera koja snima promet iz vozila na kojem je pričvršćena, zbog kretanja vozila konstantno mijenja pozadinu te je nemoguće koristiti metodu oduzimanja pozadine koja se koristi u ranije opisanim rješenjima. Također pri procjeni brzine potrebno je uzeti u obzir brzinu vozila na kojem se nalazi kamera. Algoritam razvijen u sklopu ovog diplomskog rada razvijen je u programskom jeziku *Python* korištenjem *OpenCV* biblioteke koja je namijenjena za stvarno vremenski računalni vid (engl. *computer vision*). Konstantna promjena pozadine, kretanje vozila koje nosi kameru i promjena osvjetljenja u području snimanja kamere su ograničenja koja utječu na rad algoritma. Algoritam za procjenu brzine podijeljen je u četiri dijela:

1. Predobrada video okvira
2. Detekcija nadolazećeg vozila
3. Praćenje detektiranog vozila između uzastopnih video okvira
4. Procjena brzine, udaljenosti i vremena preostalog do kolizije

Na slici 3.1. može se vidjeti blok dijagram na kojem je prikazan način na koji algoritam rješava problem procjene brzine nadolazećeg vozila.

Predobrada video okvira osigurava smanjenje grešaka i povećanje brzine rada algoritma za procjenu brzine nadolazećeg vozila vidljivog u retrovizoru. Detekcijom dobivamo informaciju gdje se u video okviru nalazi vozilo. Praćenje detektiranog vozila između uzastopnih video okvira omogućuje povećanje brzine obrade video okvira jer samo praćenje zahtjeva manje procesorskog vremena nego detekcija. Procjenom brzine, udaljenosti i vremena preostalog do kolizije dobivaju se informacije na temelju kojih vozač može odlučiti može li sigurno promijeniti voznu traku.



Sl. 3.1. *Blok dijagram novo stvorenog algoritma za procjenu brzine nadolazećeg vozila*

3.1. Predobrada video okvira

Video okvire koji predstavljaju ulaz algoritma potrebno je obraditi prije nego se počne s daljnjim procesiranjem. Predobradom se osigurava smanjenje grešaka i povećanje brzine rada. Konkretno,

predobradom se izdvaja dio video okvira koji je od interesa, a ostatak se zanemaruje. Primjerice, često se u video okvirima nalazi nebo ili okolina ceste koja u ovom slučaju ne predstavlja regiju od interesa te se zbog toga taj dio video okvira izbacuje. Izbacivanjem tog djela video okvira vrijeme obrade istog se značajno povećava.

Kako je vidljivo na slikama 3.2. i 3.3. dijelovi video okvira koji su izrezani neće sadržavati objekt od interesa, odnosno vozilo. Ovim postupkom ubrzava se rad algoritma i smanjuju se pogreške koje mogu nastati prilikom detekcije vozila koje se nalazi pored ceste ili sl.

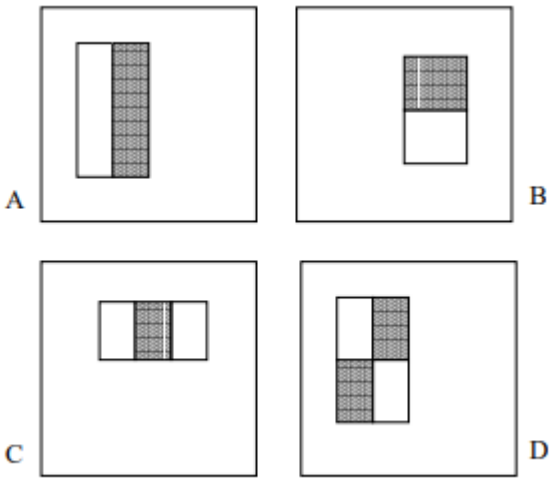
Sl. 3.2. *Video okvir prije predobrade, kamera RDACM23 (1280x720 piksela)*

Sl. 3.3. *Video okvir nakon predobrade, kamera RDACM23*

3.2. Detekcija nadolazećeg vozila

Ključni korak prilikom procjene brzine je detekcija nadolazećeg vozila. Detekcija se temelji na algoritmu koji su opisali P. Viola i M. Jones [8]. Algoritam opisuje način na koji se trenira klasifikator, te način detekcije objekata na slikama. Prvenstvena namjena algoritma je bila detekcija ljudskog lica, ali se pomoću njega mogu detektirati razni objekti [8].

Značajke (engl. *features*) koje algoritam koristi podijeljene su u tri grupe koje se mogu vidjeti na slici 3.4.



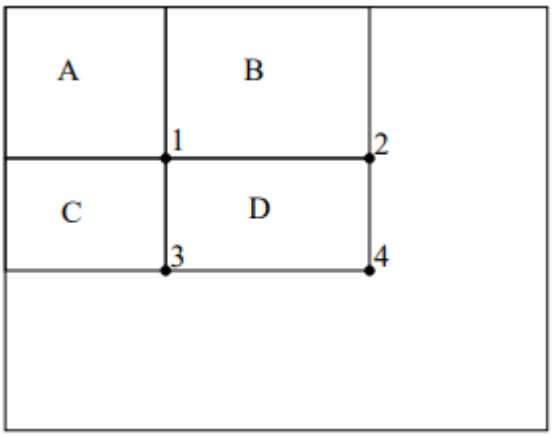
Sl. 3.4. *Značajke koje se koriste za detekciju objekata prilikom korištenja Viola Jones algoritma [8]*

Značajke A i B sa slike 3.4. spadaju u grupu značajki definiranih s dva pravokutnika, značajka C spada u grupu definiranu s tri pravokutnika te značajka D spada u grupu definiranu sa 4 pravokutnika. Suma elemenata slike ispod bijelog dijela značajke oduzima se od sume elemenata slike koji se nalaze ispod sivog dijela značajke. Sustavi koji koriste značajke rade puno brže nego sustavi koji obavljaju operacije nad elementima slike zasebno [8].

Dodatno se ubrzanje izvršavanja algoritma dobiva upotrebom integralne slike (engl. *integral image*). Integralna slika je novi način prikaza slike pomoću kojeg se slika prikazuje kao suma elemenata slike iznad i lijevo od promatranog elementa slike uključujući i njega. Formula (3-1) se koristi za računanje integralne slike od promatranog elementa slike, gdje je $ii(x, y)$ integralna slika, a $i(x', y')$ element slike na lokaciji (x', y') [8], a x' i y' redom predstavljaju redni broj retka i stupca.

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{3-1}$$

Slika 3.5. prikazuje integralnu sliku. Područje A je integralna slike od točke 1, područje označeno sa A i B predstavljaju integralnu sliku točke 2, područja A i C predstavljaju integralnu sliku točke 3, a integralna slika točke 4 su sva područja, A, B, C i D. Suma područja D može se izračunati kao zbroj integralnih slika točaka 4 i 1 umanjen za zbroj integralnih slika točaka 2 i 3, odnosno $4 + 1 - (2 + 3)$. Svaka pravokutna suma može se izračunati pomoću 4 točke. Iz prethodno navedenog može se zaključiti da se razlika suma dva područja, za značajku definiranu s dva pravokutnika, može izračunati pomoću osam točaka, ali kako područja imaju 2 zajedničke točke, razliku sumi područja moguće je izračunati pomoću šest točaka. Nadalje, za značajku definiranu s tri pravokutnika, razlika sumi područja može se izračunati pomoću osam točaka, a značajka definirana sa četiri pravokutnika zahtjeva devet točaka [8].



Sl. 3.5. *Način dobivanja integralne slike [8]*

3.2.1. Učeća klasifikacijska funkcija

Učeća klasifikacijska funkcija (engl. *learning classification function*) na ulazu prima dva skupa slika i skup značajki. Za dani skup pozitivnih (slike koje sadrže objekt koji želimo detektirati) i negativnih slika (slike koje ne sadrže objekt od značaja) i skup značajki moguće je koristiti razne algoritme za učenje klasifikacijske funkcije. Korišten je *AdaBoost* algoritam za odabir malog broja značajki i za treniranje klasifikatora. *AdaBoost* algoritam je algoritam koji poboljšava performanse jednostavnih učećih algoritama. Zbog velikog broja značajki koji se nalaze na slici, potrebno je pronaći manji broj značajki čijom se kombinacijom stvara učinkovit klasifikator [8].

Jednostavni učeći algoritam odabire jednu značajku koja najbolje odvaja pozitivnu i negativnu sliku. Praga klasifikacijske funkcije (minimalan broj primjera koji su krivo klasificirani) određuje

se za svaku značajku. Svaki se jednostavni klasifikator sastoji od značajke i praga. Proces odabira značajki koje će se koristiti prilikom detekcije željenog objekta opisan je u nastavku [8]:

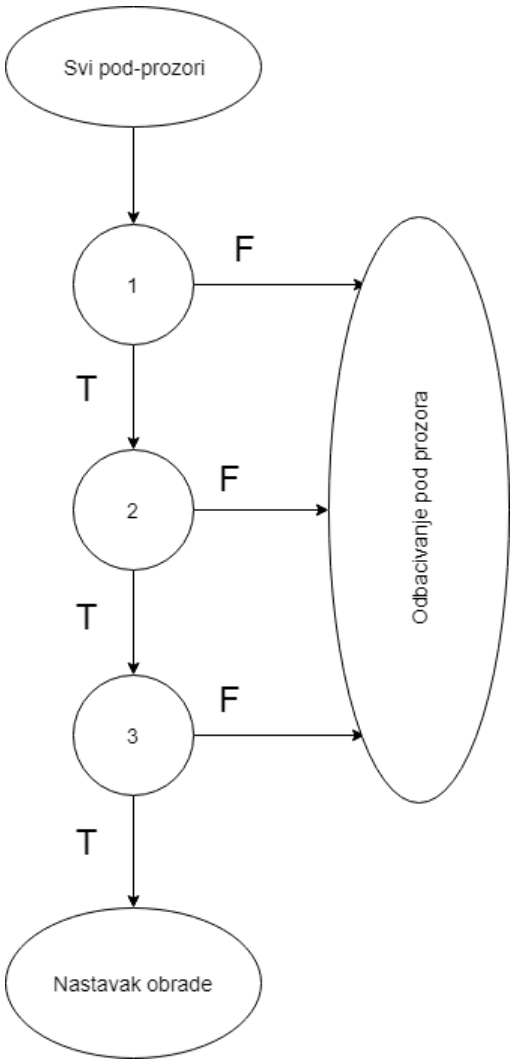
- Skup pozitivnih i negativnih slika dan je redom
- Postavljanje početne vrijednosti težina u odnosu na broj pozitivnih i negativnih slika
- Za svaku iteraciju vrši se:
    - Normalizacija težina te se dobiva raspodjela vjerojatnosti
    - Za svaku značajku trenira se klasifikator koji koristi samo jednu značajku te se računa greška klasifikatora
    - Odabiranje klasifikatora s najmanjom greškom
    - Ažuriranje težina
- Spajanje jednostavnih, odnosno slabih, klasifikatora u jedan jaki klasifikator.

Svaka iteracija vrši odabir jedne od mogućih 160000+ značajki. Odabirom značajki koje se koriste prilikom detekcije stvara se kaskada klasifikatora. Stvaranjem kaskade klasifikatora osiguravaju se detekcijska svojstva, a pritom se drastično smanjuje vrijeme proračuna. Jednostavniji se klasifikatori koriste za odbacivanje većine pod-prozora (dijelovi slike na kojima se može nalaziti objekt od značaja) prije nego se pozivaju složeniji klasifikatori. Proces detekcije svodi se na stvaranje stabla odluke, odnosno kaskade klasifikatora [8].

Na slici 3.6. nalazi se shema kaskade, odnosno stabla odluke. Pozitivan rezultat prvog klasifikatora, na slici označeno s „T“ (engl. *true*), okida drugi klasifikator. Nadalje, pozitivan izlaz iz drugog klasifikatora okida treći, itd. Svaki klasifikator je namješten za ostvarivanje vrlo visoke razine detekcije, odnosno svaki od njih odbacuje većinu negativnih pod-prozora dok detektira skoro sve pozitivne primjere. Serija klasifikatora se primjenjuje na svaki pod-prozor. Početni klasifikatori odbacuju veliki broj negativnih primjera s minimalnom razinom obrade. Daljnja obrada zahtjeva veću razinu računanja pri čemu dodatno odbacuje negativne primjere. Pozitivan ishod svih klasifikatora označava da se na tom pod-prozoru nalazi objekt od interesa. Ukoliko bilo koji od klasifikatora ima negativan ishod, na slici označeno s „F“ (engl. *false*), pod-prozor se odbacuje [8].

Pomoću prikupljenih slika, pozitivnih i negativnih, istreniran je detektor kojim se u videu detektiraju željena vozila. Pozitivne slike predstavljaju slike sa objektom koji je potrebno detektirati, u ovom slučaju željeno vozilo. Objekti koje ne želimo detektirati nalaze se na negativnim slikama. Nakon prikupljenih slika moguće je pokrenuti trening koji može trajati dugo

vremena (ovisno o broju slika i ostalim parametrima). Detaljniji opis postupka može se pronaći u [9].



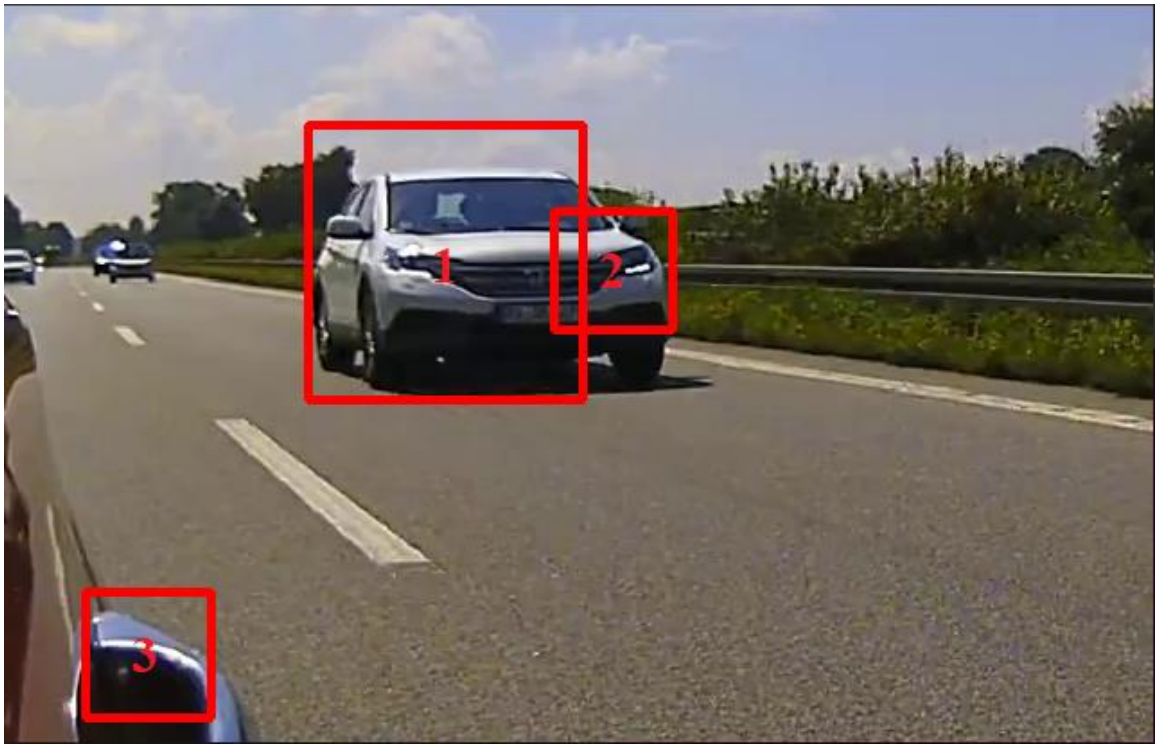
Sl. 3.6. *Shematski prikaz detekcijske kaskade [8]*

Po završetku treninga moguće je vršiti detekciju željenog objekta sa slike ili video snimke. Na slici 3.7. može se vidjeti detektirani objekt. Pre-trenirani detektor nastaje kada se trening provodi u puno faza i tada se na željenom vozilu osim samo vozila detektiraju i detalji (npr. na željenom se vozilu umjesto cijelog vozila detektiraju svjetla), a primjer toga se može vidjeti na slici 3.8.

Na slici 3.7. može se vidjeti dobro detektirani objekt na kojem se mogu vršiti daljnje operacije algoritma, a na slici 3.8. može se vidjeti detektirani objekt uz koji se događaju i greške prilikom detekcije. Pravokutnik označen brojem „1“ je objekt kojeg smo željeli detektirati, u ovom slučaju automobil. Pravokutnik „2“ označava detalj na vozilu koji je detektiran jer se treniranje povodilo

u previše faza i detektor počinje označavati detalje poput prednjih svjetala na automobilu, dok pravokutnik označen brojem „3“ predstavlja grešku prilikom detekcije.

Sl. 3.7. *Detektirano željeno vozilo pomoću dobro istreniranog detektora, kamera RDACM23*



Sl. 3.8. *Pre-trenirani detektor, detekcija detalja na željenom objektu umjesto detekcije samog objekta, kamera RDACM23*

Brzina detektiranja objekata ovisi o veličini video okvira. Što su dimenzije video okvira veće potrebno je više vremena za detekciju. Smanjenjem dimenzija video okvira povećava se brzina izvođenja detekcije, a time i brzina izvođenja algoritma. Nakon dobivanja zadovoljavajućih rezultata detekcije željenog objekta, moguće je prijeći na sljedeći korak algoritma, koji predstavlja praćenje objekta kroz video zapis.

3.3. Praćenje detektiranog vozila između dva uzastopna video okvira

Sustavi koji služe za pomoć vozaču prilikom vožnje moraju raditi u stvarnom vremenu, zbog čega je često neophodna optimizacija brzine rada algoritma. Ubrzanje izvođenja algoritma omogućeno je korištenjem praćenja detektiranog objekta između video okvira, jer sama detekcija zahtjeva više vremena nego praćenje objekta. Praćenjem detektiranog objekta se ne onemogućuje detekcija, nego se na svakom desetom video okviru vrši ponovna detekcija. Ponovnom detekcijom svakih deset video okvira osigurava se detekcija novih objekata od interesa, ako su ušli u video okvir. Za praćenje objekata između video okvira koristi se *MedianFlow* algoritam koji je dio *OpenCV* biblioteke.

MedianFlow algoritam obavlja praćenje prethodno detektiranog objekta tako da se uspoređuju dva susjedna video okvira i vrši se proračun pomaka mreže točaka unutar graničnog područja (engl. *bounding box*) u kojem se nalazi detektirani objekt. Točke koje se nalaze na ulaznom okviru prate se na slijedećem video okviru pomoću *Lucas-Kanade* algoritma, pomoću čega se dobiva raspršeni tok gibanja (engl. *sparse motion flow*) između dva susjedna video okvira. Procjenjuje se kvaliteta procjene svake točke i svakoj se točki pridjeljuje pogreška (npr. greška naprijed nazad (engl. *forward-backward, FB*), normalizirana kros-korelacija (NCC) ili suma razlike kvadrata (engl. *sum-of-square difference, SSD*)). Polovica točaka s najlošijom procjenom, najvećom pogreškom, se odbacuje, a točke koje su ostale koriste se za procjenu pokreta objekta. Izlaz iz algoritma je novi granični okvir u kojem se nalazi objekt koji se prati [10].

Kao što je već navedeno, praćenjem objekata povećava se brzina izvođenja algoritma jer praćenje zahtjeva manje proračuna nego detekcija. Sljedeći korak algoritma je procjena brzine nadolazećeg vozila, koja će biti opisana u nastavku.

3.4. Procjena udaljenosti vozila, brzine vozila i vremena preostalog do kolizije

U današnjim automobilima sve je veći naglasak na sustavima koji pomažu vozaču zbog smanjenja prometnih nesreća. Često kriva procjena brzine i udaljenosti vozila u retrovizoru prilikom

promjene prometnih traka uzrokuje nesreće na prometnicama. Zbog navedenih problema ovaj algoritam nudi pomoć pri procjeni brzine i udaljenosti za sigurniju promjenu prometnih traka. Jedan od glavnih parametara za ove proračune je brzina izmjene video okvira, jer iz tog podatka možemo dobiti vrijeme koje je proteklo između dvije pozicije vozila u videu. Osim brzine izmjene okvira na proračun utječe i vrsta kamere koja se koristi i parametri leće koju kamera koristi. Sama informacija o brzini vozila vozaču ne daje puno informacija, zbog toga je dodana informacija o udaljenosti i informacija o vremenu potrebnog do kolizije kako bi vozač imao više informacija na raspolaganju prilikom odluke o potencijalnoj promjeni prometne trake na cesti. U nastavku će biti opisan proračun udaljenosti detektiranog vozila, brzine detektiranog vozila i vremena koje je potrebno da to vozilo dođe na mjesto kamere.

3.4.1. Procjena udaljenosti vozila

Kamere koje su se koristile prilikom razvijanja algoritma su: RDACM23, kamere mobilnih telefona *Samsung Galaxy S8, Samsung Galaxy A5 (2017)*. Sve te kamere imaju različite parametre te je algoritam potrebno prilagoditi za svaku kameru posebno. Ukoliko se želi koristiti neka druga kamera, potrebno je prilagoditi algoritam kako bi mogao raditi za tu kameru. Parametri koji su potrebni za kameru koja se koristi su žarišna udaljenost leće i širina senzora.

Kamera *RDACM23* je kamera koja se koristi na razvojnim platformama za napredne algoritme za pomoć vozaču. Kamera ima $0{,}005376$ m širinu senzora. Leća koja se koristi na kameri nije poznata (u algoritmu je korištena žarišna daljina od $0{,}0022\ m$).

Kamera mobilnog telefona *Samsung Galaxy S8* kojom su se snimale snimke koje su korištene tokom razvoja algoritma ima širinu senzora $5{,}376\ mm$, a žarišna duljina leće na mobilnom telefonu je $4{,}2\ mm$ [11, 12].

Kamera *Samsung Galaxy A5* (2017) mobilnog telefona ima žarišnu duljina $3{,}6\ mm$, a širina senzora je $5.94\ mm$ [13, 14].

Udaljenost detektiranog vozila, uz brzinu nadolazećeg vozila, predstavlja bitan podatak vozaču jer na temelju ta dva podatka vozač jednostavnije može procijeniti smije li sigurno promijeniti prometnu traku ili ne. Udaljenost od kamere do nadolazećeg vozila se procjenjuje na temelju formule (3-2):

$$d[mm] = \frac{f[mm] \cdot h[mm] \cdot h_{slike}[element\ slike]}{h'[element\ slike] \cdot h_{senzor}[mm]}, \qquad (3\text{-}2)$$

gdje je:

- d – udaljenost objekta od kamere na automobilu u milimetrima,
- f – žarište duljina leće u milimetrima,
- h – visina vozila u milimetrima u stvarnom svijetu,
- h_{slike} – visina video okvira u elementima slike,
- h' – visina objekta u video okviru u broju elemenata slike,
- h_{senzor} – visina senzora kamere u milimetrima [15].

S obzirom da se visina vozila često razlikuje (osobni automobil i kombi vozila), umjesto visine vozila u navedenoj formuli koristila se širina vozila, jer je su vozila po širini sličnija nego po visini. Osim visine mijenja se i varijabla visine vozila u video okviru u širinu vozila. Koristi se širina video okvira i širina senzora kamere umjesto visine video okvira i visine senzora kamere. Tako se dobiva formula (3-3):

$$d[mm] = \frac{f[mm] \cdot w[mm] \cdot w_{slike}[element\ slike]}{w'[element\ slike] \cdot w_{senzor}[mm]}, \tag{3-3}$$

gdje je:

- d – udaljenost objekta od kamere na automobilu u milimetrima,
- f – žarišna duljina leće u milimetrima,
- w – širina vozila u milimetrima u stvarnom svijetu,
- w_{slike} – širina video okvira u elementima slike,
- w' - širina objekta u video okviru u broju elemenata slike,
- w_{senzor} – širina senzora kamere i milimetrima.

Na slici 3.9. vidi se rezultat procjene udaljenosti detektiranog vozila. Procjena udaljenosti se radi u svakom video okviru. Prilikom procjene brzine unosi se pogreška zbog toga što se objekt vrlo rijetko nalazi u sredini leće. Formule (3-2) i (3-3) vrijede samo ako se objekt nalazi u sredini leće, ali kako su leće kamera koje su se koristile dovoljno male, moguće je koristiti navedene formule jer se njima ne unosi velika pogreška. Pogreška koja nastaje procjenom udaljenosti propagirat će se prilikom procjene brzine. Prosječna širina vozila je $1{,}8m$, što je uvršteno u formulu (3-3). Ukoliko je širina vozila manja, odnosno veća od prosječne nastaje pogreška prilikom procjene udaljenosti vozila. Pogreška može nastati i svaki 10. video okvir jer se svakih 10 video okvira vrši detekcija novih objekata, ako su ušli u scenu.

Sl. 3.9. *Procjena udaljenosti detektiranog vozila, kamera Samsung Galaxy S8*

3.4.2. Procjena brzine nadolazećeg vozila

Procjena brzine nadolazećeg vozila predstavlja središnji problem ovog rada. Automatska procjena brzine omogućuje vozaču ugodniju i sigurniju vožnju. Procjena brzine također ovisi o kameri koja se koristi jer se procjena vrši preko ranije procijenjene udaljenosti nadolazećeg vozila. Procjena brzine će sadržavati pogrešku zbog pogreške koju nasljeđuje od djela algoritma koji procjenjuje udaljenost. Brzina nadolazećeg vozila računa se prema izrazu (3-4):

$$v = \frac{d_{i-1} - d_i}{t}, \tag{3-4}$$

gdje je:

- v – procijenjena brzina nadolazećeg vozila u metrima u sekundi,
- d_{i-1}– udaljenost do vozila na prethodnom okviru u metrima,
- d_i – udaljenost do vozila na trenutnom okviru u metrima,
- t – period između dva uzastopna video okvira.

Kako se kamera nalazi pričvršćena na vozilo koje vozi nekom brzinom formula (3-4) se ne može koristiti jer se brzina vozila ne uzima u obzir. Zbog toga potrebno je dodati brzinu vlastitog vozila kako bi procjena brzine bila točnija. Formulom (3-5) se procjenjuje brzina:

$$v = v_0 + \frac{d_{i-1} - d_i}{t}, \tag{3-5}$$

gdje je:

- v_0 – brzina vozila na kojem se nalazi kamera u metrima u sekundi.

U formulama (3-4) i (3-5) se za recipročnu vrijednost broja obrađenih okvira u sekundi uvrštava $33\ ms$, jer sve video sekvence koje su se koristile tokom razvoja algoritma imaju 30 okvira u sekundi.

Slika 3.10. prikazuje procjenu brzine nadolazećeg vozila. U toj brzini se nalazi greška koja nastaje zbog procjene udaljenosti i zbog detekcije novih vozila svakog desetog video okvira. Detekcija vozila svakog desetog video okvira utječe na brzinu obrade video okvira i dolazi do oscilacija prilikom procjene brzine. Kako bi se riješio problem oscilacija prilikom procjene brzine uzima se medijan svih procijenjenih brzina. Medijanom skupa procijenjenih brzina osigurava se procjena koja odgovara brzini oko koje se grupira najveći broj procjena (npr. ukoliko imamo slijedeći skup brzina: 40 km/h, 41 km/h, 15 km/h, 40 km/h, 51 km/h i 40 km/h, medijan navedenog skupa će biti 40 km/h).

Sl. 3.10. *Procjena brzine detektiranog vozila, kamera Samsung Galaxy S8*

3.4.3. Procjena vremena preostalog do kolizije

Vrijeme potrebno do kolizije (engl. *time to colision – TTC)* predstavlja vrijeme koje je potrebno detektiranom vozilu da dođe do mjesta kamere. Ova informacija vozaču govori koliko ima

vremena dok vozilo iza njega dođe. Kako bi se dobila procjena TTC-a koriste se informacije o udaljenosti i procjenjenoj brzini.

$$TTC = \frac{d}{v}\ [s], \tag{3-6}$$

gdje je:

- TTC – vrijeme koje je potrebno nadolazećem vozilu da dođe do mjesta kamere u sekundama,
- d – udaljenost do nadolazećeg vozila u metrima,
- v – procijenjena brzina nadolazećeg vozila u metrima u sekundi.

Informacija o TTC-u omogućuje vozaču procjenu ima li vremena za promjenu prometne trake ili nema. Na slici 3.9. može se vidjeti procjena TTC-a.

Sl. 3.11. *Procjena vremena preostalog do kolizije, kamera Samsung Galaxy S8*

Greška koja se javlja uzrokovana je greškama prilikom procjene udaljenosti i procjene brzine nadolazećeg vozila. Ova informacija daje okvirno vrijeme koje vozač ima na raspolaganju, jer ukoliko nadolazeće vozilo naglo poveća svoju brzinu vrijeme koje vozač ima se vrlo brzo smanjuje.

3.5. Pokretanje algoritma

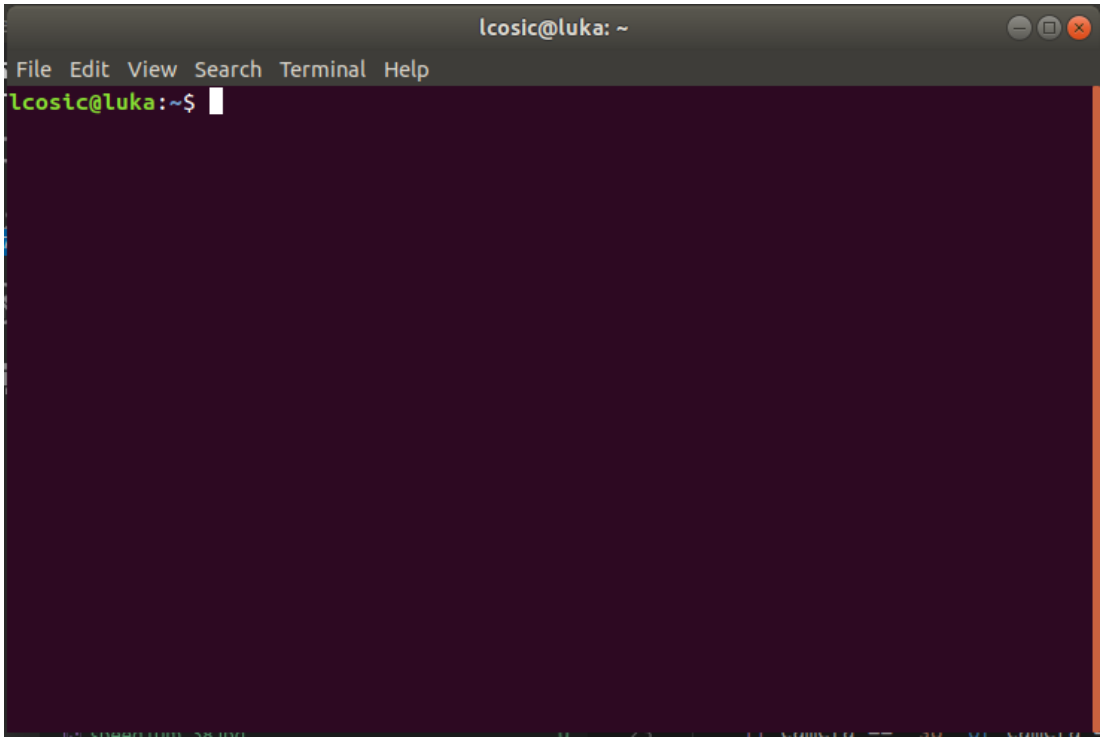
Algoritam je razvijen na operacijskom sustavu *Ubuntu*. Kako bi se algoritam pokrenuo potrebno je imati sljedeće komponente:

- *Python* verziju 3.6.5
- *OpenCV* verziju 4.0.0
- *Python* paket za statistiku

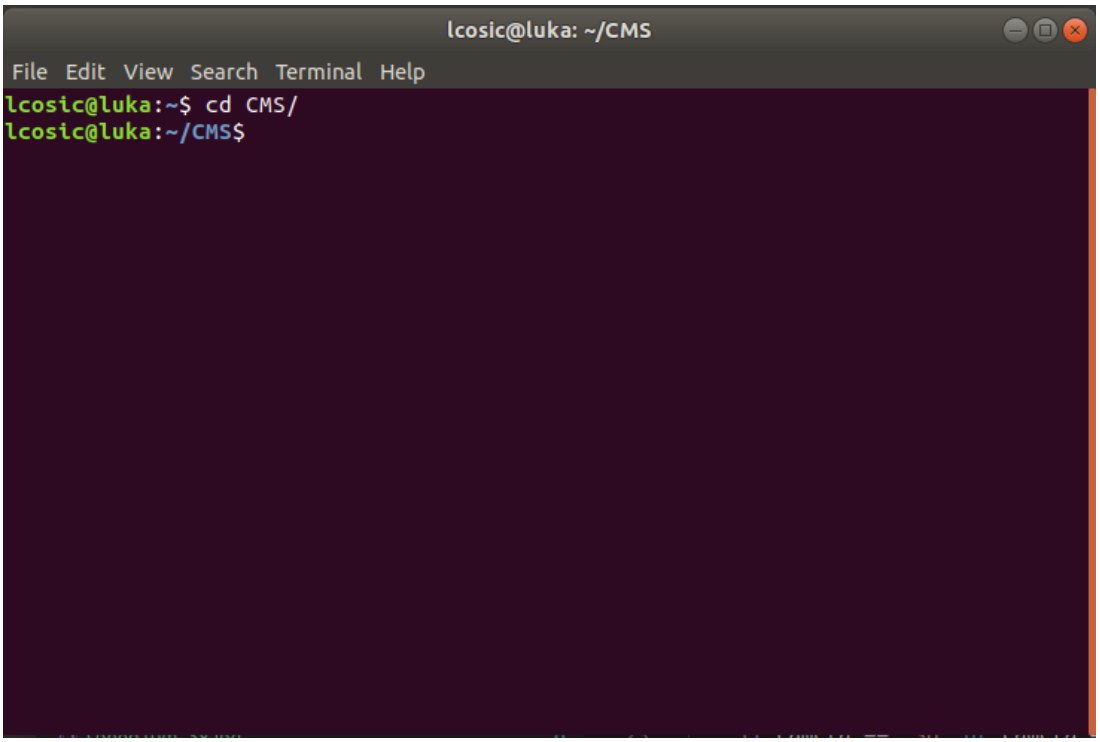
Pokretanje algoritma radi se na slijedeći način:

- Otvoriti „Terminal“, slika 3.12. (kombinacija tipki: *Ctrl* + *Alt* + *T*)
- Postaviti se na lokaciju gdje se nalazi skripta algoritma, (slika 3.13)
- Upisati sljedeću naredbu, (slika 3.14):
    - `python3 procjena_brzine.py <putanja_do_video_snimke> <kamera_kojom_je_snimano>`
- Na slici 3.15. može se vidjeti pokrenuti algoritam.

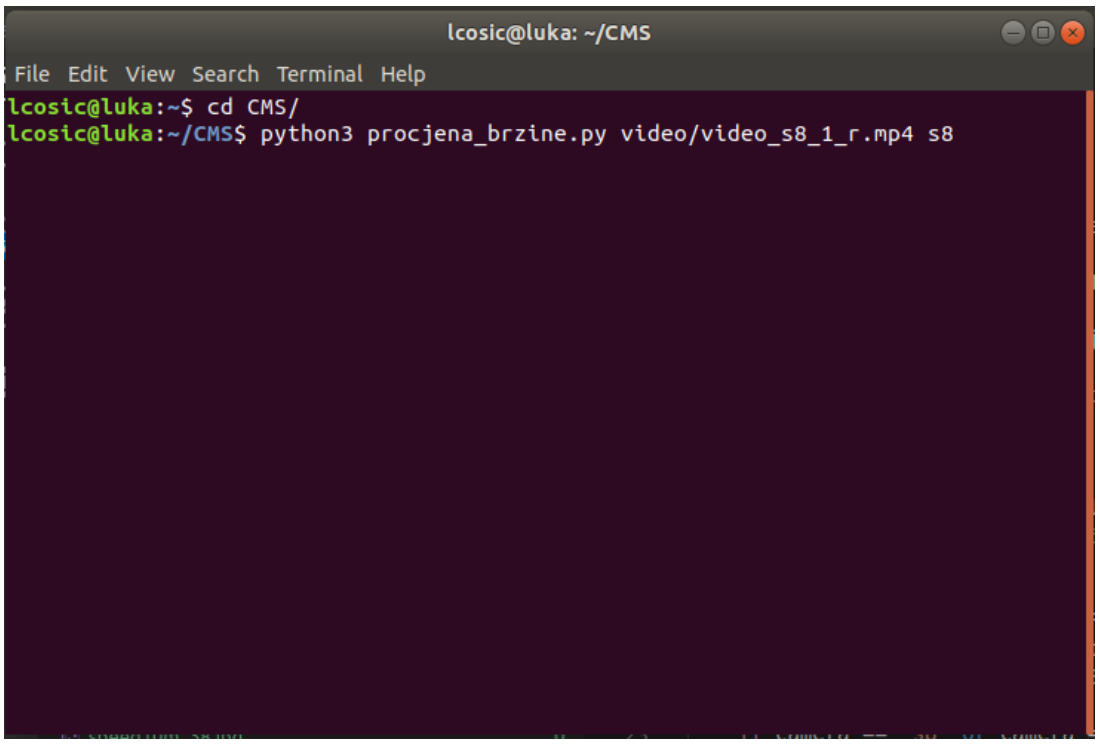
Skripta *procjena_brzine.py* se nalazi u P.3.1. Parametar putanje do video snimke označava video koji skripta treba pokrenuti, a kamera kojom je snimano osigurava da algoritam koristi točne parametre kamere kako bi rezultati algoritma bili točni.



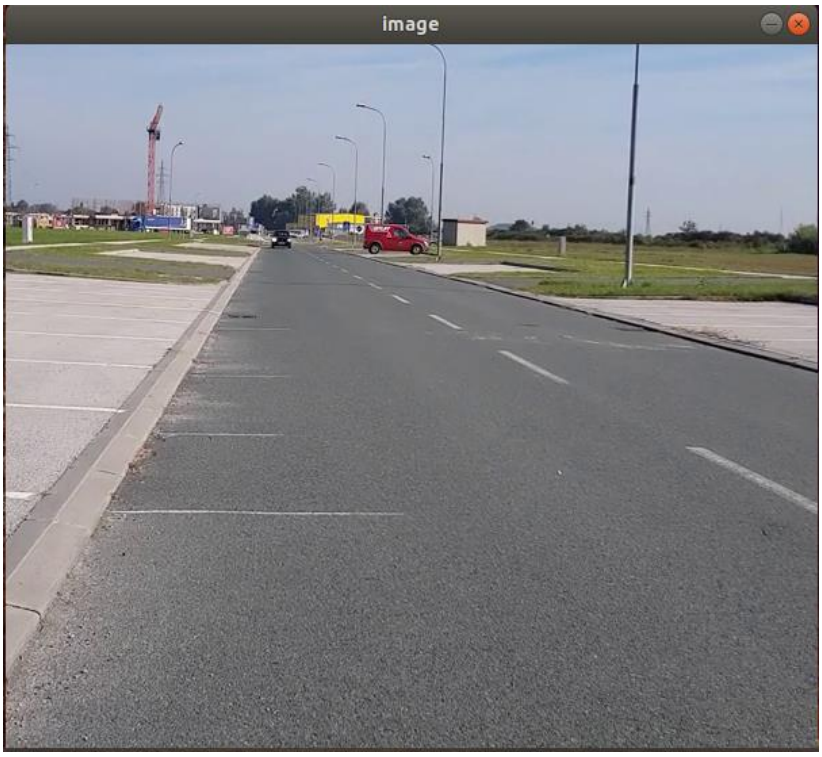
Sl. 3.12. *Terminal na Ubuntu operacisjkom sustavu*



Sl. 3.13. *Promjena putanje do mjesta gdje se nalazi skripta*



Sl. 3.14. *Naredba za pokretanje algoritma*



Sl. 3.15. *Pokrenuti algoritam za procjenu brzine nadolazećeg vozila*

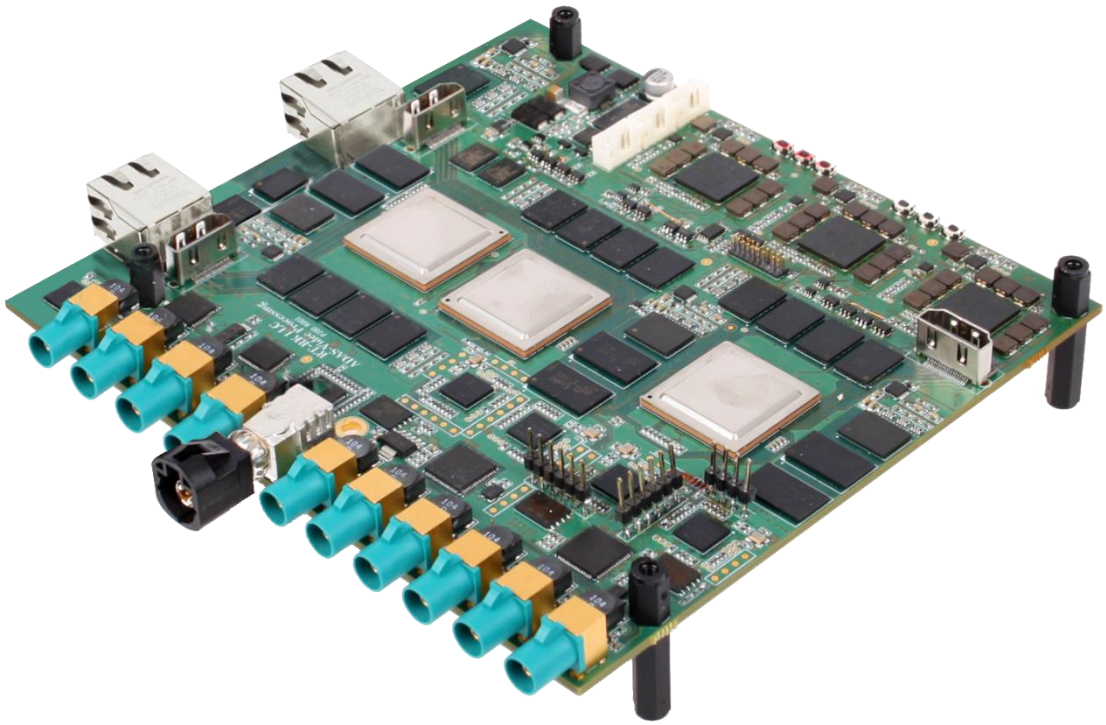
3.6. Implementacija algoritma na ADAS ploču

ADAS (engl. *advanced driver-assistance systems*) su sustavi koji pomažu vozaču tijekom vožnje. Razvijeni su kako bi automatizirali, prilagodili i unaprijedili automobila s ciljem sigurnije i udobnije vožnje. Većina prometnih nesreća nastaje zbog ljudske pogreške te se zbog toga u automobile ugrađuju automatizirani sustavi kako bi se smanjila ljudska greška. Primjeri aplikacija koje se mogu nalaziti na ADAS sustavu su: kontrola brzine, ABS sustav, automatizirano parkiranje, pomoć pri promijeni prometne trake, itd. Autonomnost vozila koje se postiže ugradnjom ADAS sustava razlikuje se u količini potrebe za ljudskom intervencijom prilikom vožnje. Zbog toga razlikujemo 6 razina autonomnosti [16]:

- Nulta razina: vozač kontrolira vozilo u potpunosti (skretanje, ubrzavanje, kočenje, itd.)
- Prva razina: vozilo pomaže vozaču prilikom ubrzavanja i upravljanja vozila. Vozač mora biti u pripravnosti preuzeti kontrolu nad vozilom u bilo kojem trenutku. Prilikom vožnje moguće je uključiti kontrolu brzine (vozilo zadržava brzinu kretanja) ili kontrolu upravljanja (vozilo zadržava smjer kretanja), ali vozač mora kontrolirati sve prilikom vožnje.

- Druga razina: djelomično autonomno vozilo pomaže vozaču prilikom skretanja i vožnje u gužvi. Vozilo može samo ubrzavati, kočiti i preuzeti kontrolu nad upravljačem. Moguće je parkirati vozilo bez kontrole vozača. I na ovoj razini vozač mora kontrolirati automobil i uvijek obraćati pozornost na promet. Primjer pomoći vozača su: prilagodljiva kontrola brzine, detekcija prometnih traka, pomoć pri parkiranju, itd.
- Treća razina: vozilo kontrolira okolinu tijekom vožnje pomoću senzora. Vozač mora preuzeti kontrolu nad vozilom u slučaju grešaka prilikom vožnje.
- Četvrta razina: potpuna autonomnost vozila, ali vozač mora biti prisutan u automobilu i može preuzeti kontrolu nad vozilom. Vozilo može kontrolirati većinu situacija kao što je vožnja u urbanim sredinama bez pomoći vozača.
- Peta razina: vozilo je potpuno autonomno, vozač ne mora biti prisutan u vozilu.

Za razvoj takvih sustava koriste se ADAS ploče. Postoje različite ADAS ploče koje se koriste za razvoj algoritama za pomoć vozaču. Neki od proizvođača takvih ploča su *NVIDIA* [17], *Renesas Electronics Corp.* [18], *Qualcomm* [19]. Za potrebe ovog diplomskog rada korištena je ALPHA ploča koja je bila dostupna [21].



Sl. 3.16. *ALPHA ploča korištena u sklopu ovog diplomskog rada [21]*

ALPHA ploča je platforma koja podržava osnovne i napredne upozoravajuće sustave, aktivne kontrolne sustave i polu-autonomne operacije. Sastoji se od 3 TDA2x SoC-a (engl. *system of chip*) procesora koja sadrže 10 jezgri (dvije ARM *Cortex* A15, dvije ARM *Cortex* M4, dvije C66x DSP i četiri EVE). Svaki SoC ima [21]:

- HDMI (engl. *High-Definition Multimedia Interface*),
- DCAN (engl. *Controller Area Network*)
- Ethernet, JTAG, UART (engl. *Universal Asynchronous Receiver-transmitter*) sučelja i
- microSD spremnik
- 1,5 GB memorije

Podržava 10 kamera koje se koriste za potrebe razvijanja algoritama za pomoć vozač. Podržava tri načina rada: preko microSD-a, *debug-a* način rada i putem *flash-a* [21].

VisionSDK je razvojno okruženje koje se koristi za izradu algoritama za ALPHA ploču. Razvojno okruženje temeljeno je na *Links and Chains* arhitekturi. *Link* predstavlja jednu funkcionalnu cjelinu koja obavlja jednu funkciju (npr. slanje video podataka), a *Chain* predstavlja spoj više *linkova*.

Razvijeni algoritam je potrebno implementirati na ALPHA ploču. Algoritam je potrebno prilagoditi arhitekturi ALPHA ploče. Video okviri bi se dobivali preko priključene kamere, te se šalje na jedan od SoC na kojem se obrađuje te se na HDMI izlazu dobiva rješenje. Nažalost implementacija nije uspješno obavljena. Algoritam se pokušao implementirati na više načina.

Implementacija rješenja pomoću *OpenCV* biblioteke na ploči nije uspjela jer ploča nije optimizirana za korištenje funkcija iz biblioteke. Ploča je uspjela obraditi 2 video okvira u sekundi. Implementacija je nastavljena na drugi način.

Korištenje *OpenCV* verzije 1.0.0. koja je napisana u C programskom jeziku je drugi način implementacije. Problem prilikom ovog pokušaja nastaje u djelu koji *parsira* datoteku u kojoj se nalaze težine za detekciju objekata. Prilikom *parsiranja* algoritam popunjava memoriju te nije u mogućnosti pročitati datoteku do kraja. Smanjenjem veličine datoteke koju je potrebno pročitati na $2KB$ također nije uspješno riješen problem.

Posljednji pokušaj implementacije predstavlja pokretanje postojećeg rješenja koje se nalazi na ALPHA ploči. Rješenje je pokrenuto, ali detektor ne radi na najbolji način. Zbog toga je pokušano implementirati vlastiti detektor, ali kako bi to bilo moguće potrebno je dobiti izvorni kôd od *Texas Instruments-a,* što nije bilo moguće.

Nakon navedenih pokušaja zaključeno je da implementacija na Alpha ploču neće biti uspješno obavljena na način kako je to prvotno zamišljeno.

4. EVALUACIJA PERFORMANSI ALGORITMA ZA PROCJENU BRZINE VOZILA VIDLJIVOG U RETROVIZORU

Evaluacija performansi algoritma obavljena je koristeći jedanaest video snimki koje su snimljene kamerama mobilnih telefona *Samsung Galaxy S8* i *Samsung Galaxy A5 (2017)*. Prilikom snimanja video sekvenci od mjesta kamere je izmjerena udaljenost od 5m, 7m, 10m i 15m te su udaljenosti označene na cesti. Na navedenim udaljenostima su se pricijenjivale vrijednosti udaljenosti detektiranog vozila do kamere, brzine detektiranog objekta i TTC-a. Prije samog testiranja video sekvencama je promijenjena dimenzija video okvira (za *Samsung Galaxy S8* sa 3840×2160 elemenata slike na 1280×720 elemenata slike, a za *Samsung Galaxy A5* sa 1920×1080 elemenata slike na 1280×720 elemenata slike) jer su video okviri bili vrlo veliki te bi za obradu takvih video okvira bilo potrebno puno više vremena. Stvarna brzina automobila u video snimkama je 50 km/h. Treba uzeti u obzir kako pokazivač trenutne brzine automobila također ima pogrešku (pogreška pokazivanja nije poznata). Pogreška prikazivanja brzine u automobilu procijenjena je na 10% vrijednosti prikazane brzine što znači da je stvarna brzina automobila 45 km/h. Prolaskom automobilom pored postavljenog mjerača brzine uz cestu brzinom 50 km/h (vrijednost na brzinomjeru automobila), mjerač pokazuje brzinu od 45 km/h te je zbog toga odlučeno da je stvarna brzina automobila 45 km/h. Kamera kojom je snimano se ne pomiče te je brzina kamere 0 km/h. Dodatna greška nastaje zbog pomicanja kamere tokom snimanja, tj. zbog nemogućnosti mirnog držanja kamere. Rezultati evaluacije su prikazani u nastavku.

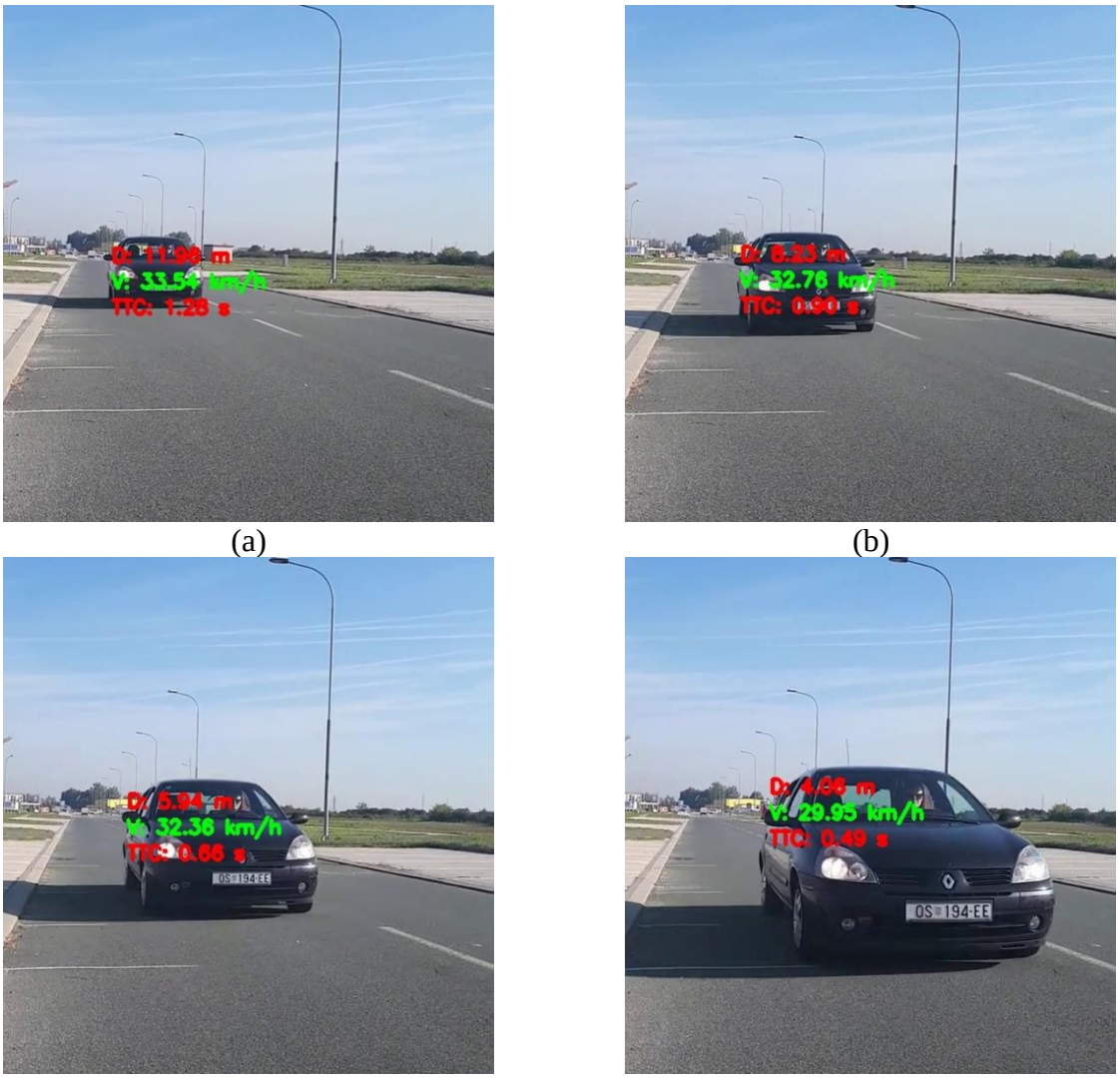
Slika 4.1. prikazuje rezultate testa 1. Na rezultate procjene udaljenosti, brzine i TTC-a utječe postavljeni kut snimanja (smjer snimanja potrebno je postaviti paralelno smjeru gibanja automobila). Pomicanje mobilnog telefona tokom snimanja također utječe na točnost procjene rezultata algoritma. U testu 1 kut snimanja nije dobro postavljen, a tokom snimanja dolazi do pomicanja mobilnog telefona. Zbog navedenih problema rezultati procjene udaljenosti, brzine i TTC odstupaju od stvarnih vrijednosti.

(a) (b)

(c) (d)

Sl. 4.1. *Test 1, video sekvenca a5_1_test.mp4, kamera Samsung Galaxy A5 (2017), automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Slika 4.2. prikazuje rezultate testa 2. Kao i u testu 1 nije dobro postavljen kut snimanja dok je kamera dosta mirnija te nema podrhtavanja video snimke. Iako je video snimka mirnija rezultati procjene odstupaju od stvarnih vrijednosti zbog kuta snimanja. Kut snimanja više utječe na točnost rezultata nego mirna video snimka. Osim kuta snimanja treba uzeti u obzir i pogrešku prilikom procjene udaljenosti što utječe na sve ostale rezultate.

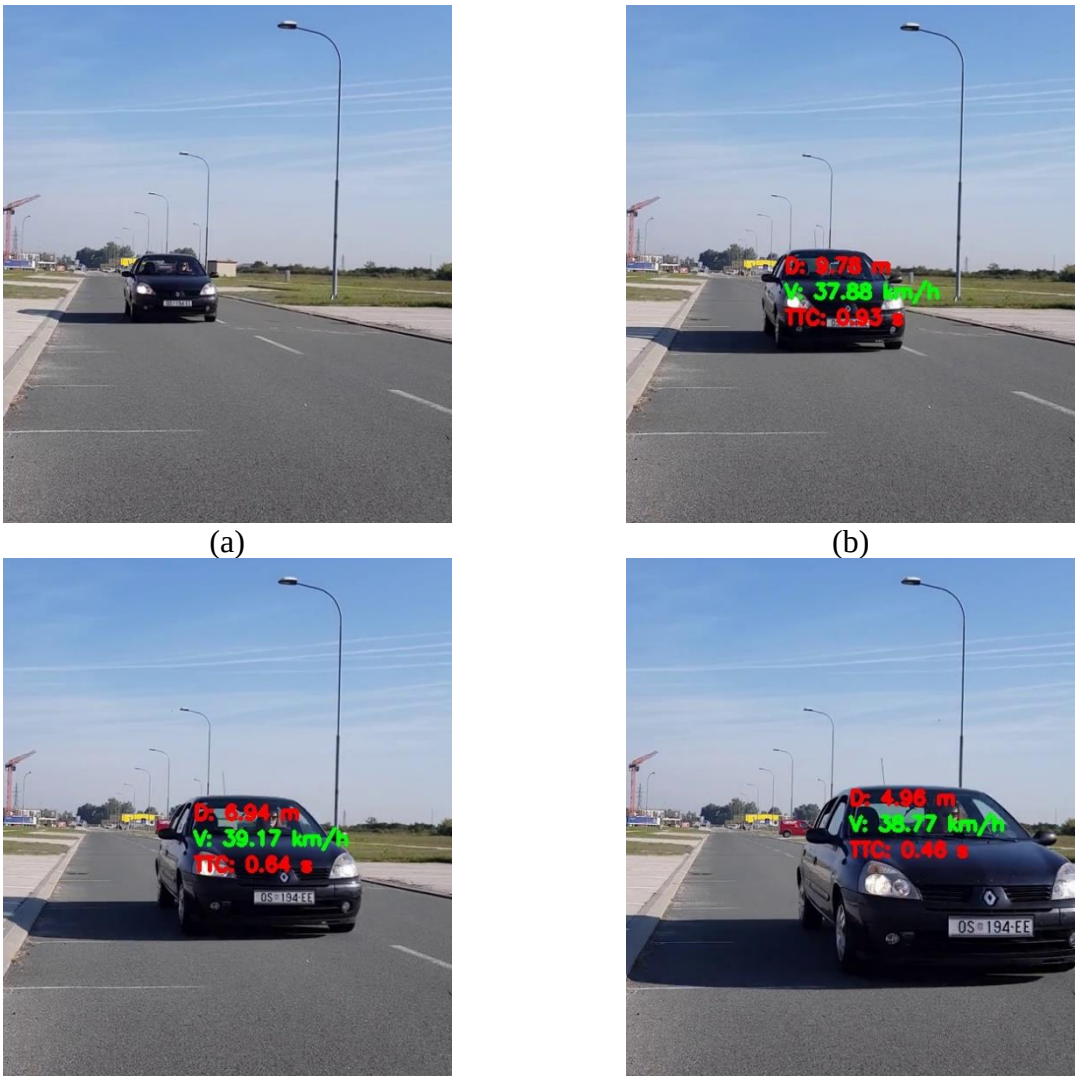


(a) (b)

(c) (d)

Sl. 4.2 *Test 2, video sekvenca a5_2_test.mp4, kamera Samsung Galaxy A5 (2017), automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Test 3 je prikazan slikom 4.3. Na 15m udaljenosti od kamere automobil nije detektiran. Bolju detekcija se može postići korištenjem bolje istreniranog detektora. Na ostalim udaljenostima vozilo je uspješno detektirano. Rezultati procjene udaljenosti, brzine i TTC-a su bliže stvarnim vrijednostima. Algoritam je uspio napraviti bolju procjenu zbog boljeg kuta snimanja. Iako postoji greška procjene udaljenosti, zbog boljeg postavljenog kuta snimanja, algoritam bolje procjenjuje udaljenost detektiranog vozila, brzinu vozila i TTC.

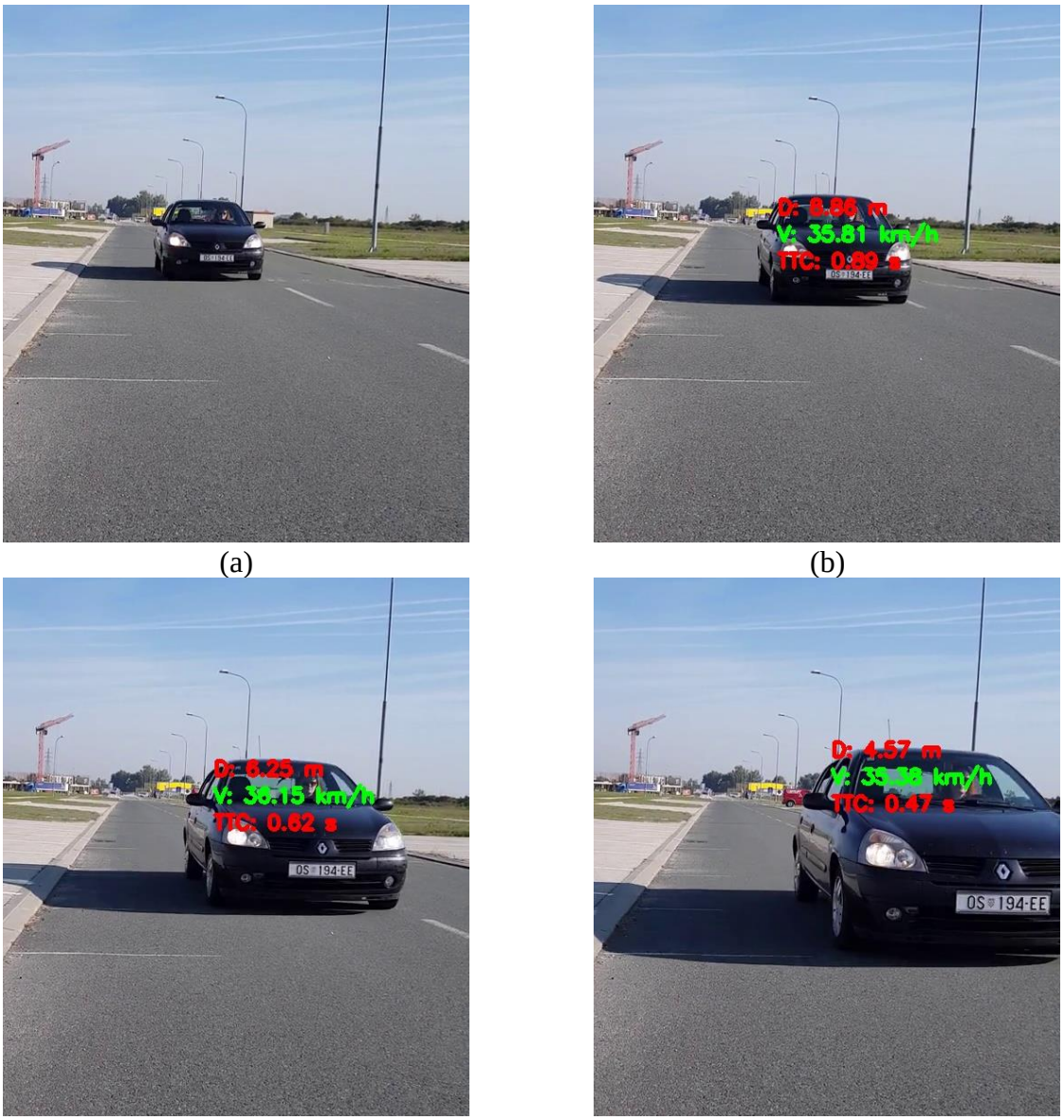


(a) (b)

(c) (d)

Sl. 4.3. *Test 3, video sekvenca s8_1_test.mp4, kamera Samsung Galaxy S8, automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Test 4 prikazan je na slici 4.4. U testu 4 algoritam nije detektirao automobil na 15m udaljenosti od kamere zbog lošije istreniranog detektora. Rezultati procjene udaljenosti, brzine i TTC-a odstupaju od stvarnih vrijednosti više nego u testu 3 zbog manjeg pomicanja kamere tokom snimanja. Kut snimanja je dobro postavljen.

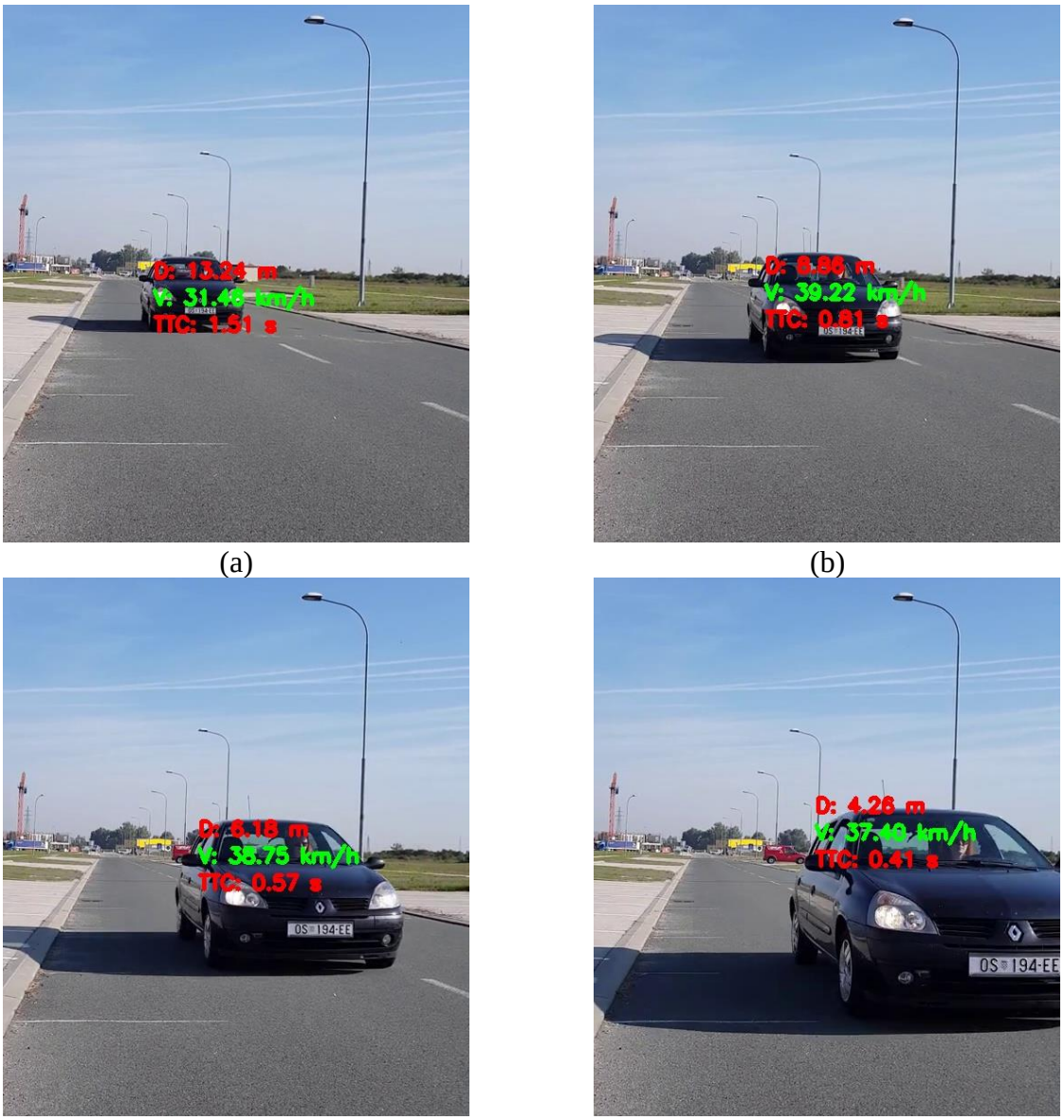


(a) (b)

(c) (d)

Sl. 4.4. *Test 4, video sekvenca s8_2_test.mp4, kamera Samsung Galaxy S8, automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Slikom 4.5. prikazan je peti test. Rezultati testa 5 nemaju veliku pogrešku u odnosu na stvarne vrijednosti što je postignutom mirnijom video snimkom i dobrim kutom snimanja. Kada se automobil nalazi na 15m od kamere procjena brzine odstupa od stvarne vrijednosti. Do takve pogreške dolazi kada se procjena brzine vrši na samo jednoj izmjeni susjednih video okvira, dok se u ostalim slučajima koristi medijan 10 uzastopnih procijenjenih brzina.

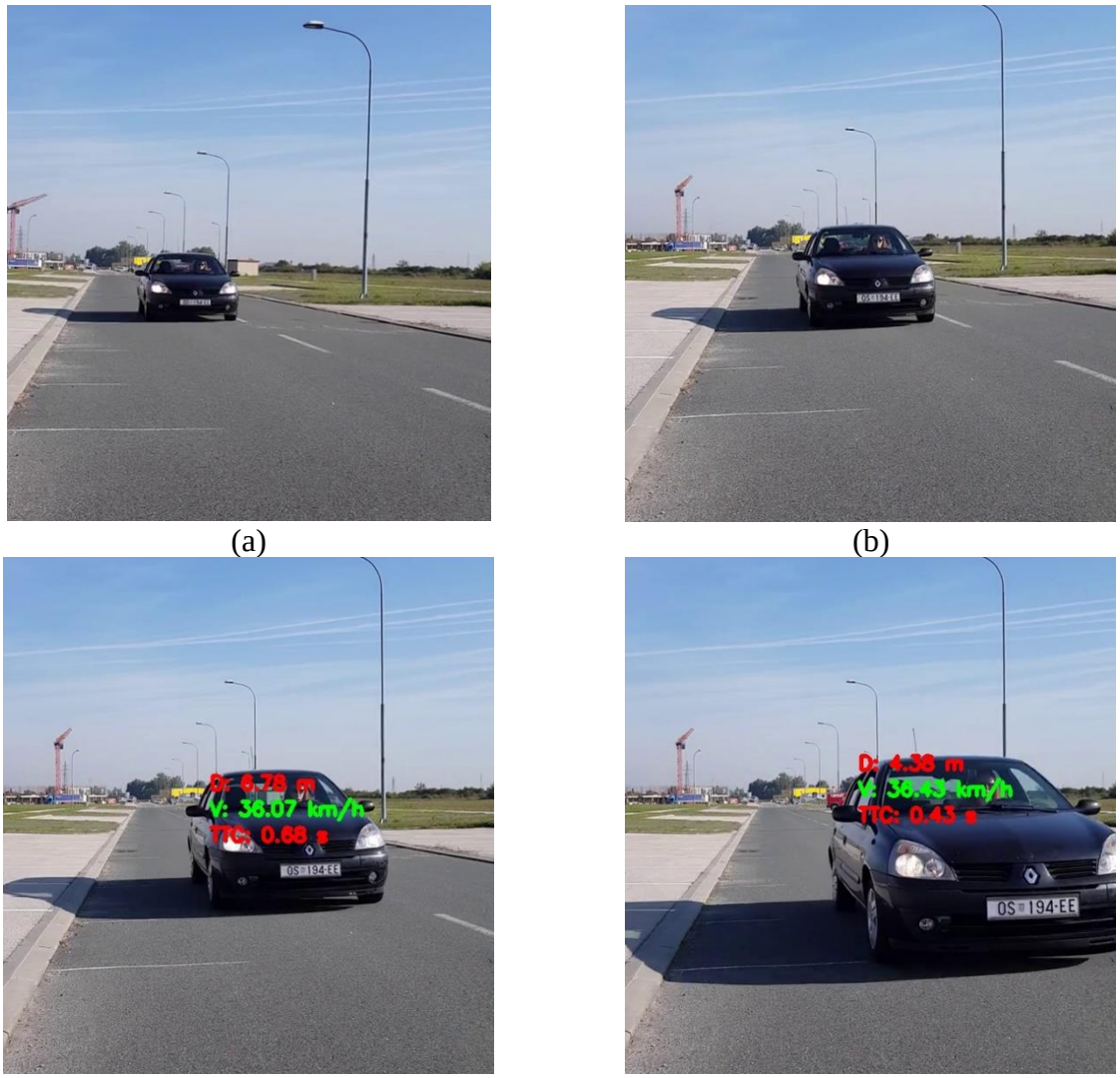


(a) (b)

(c) (d)

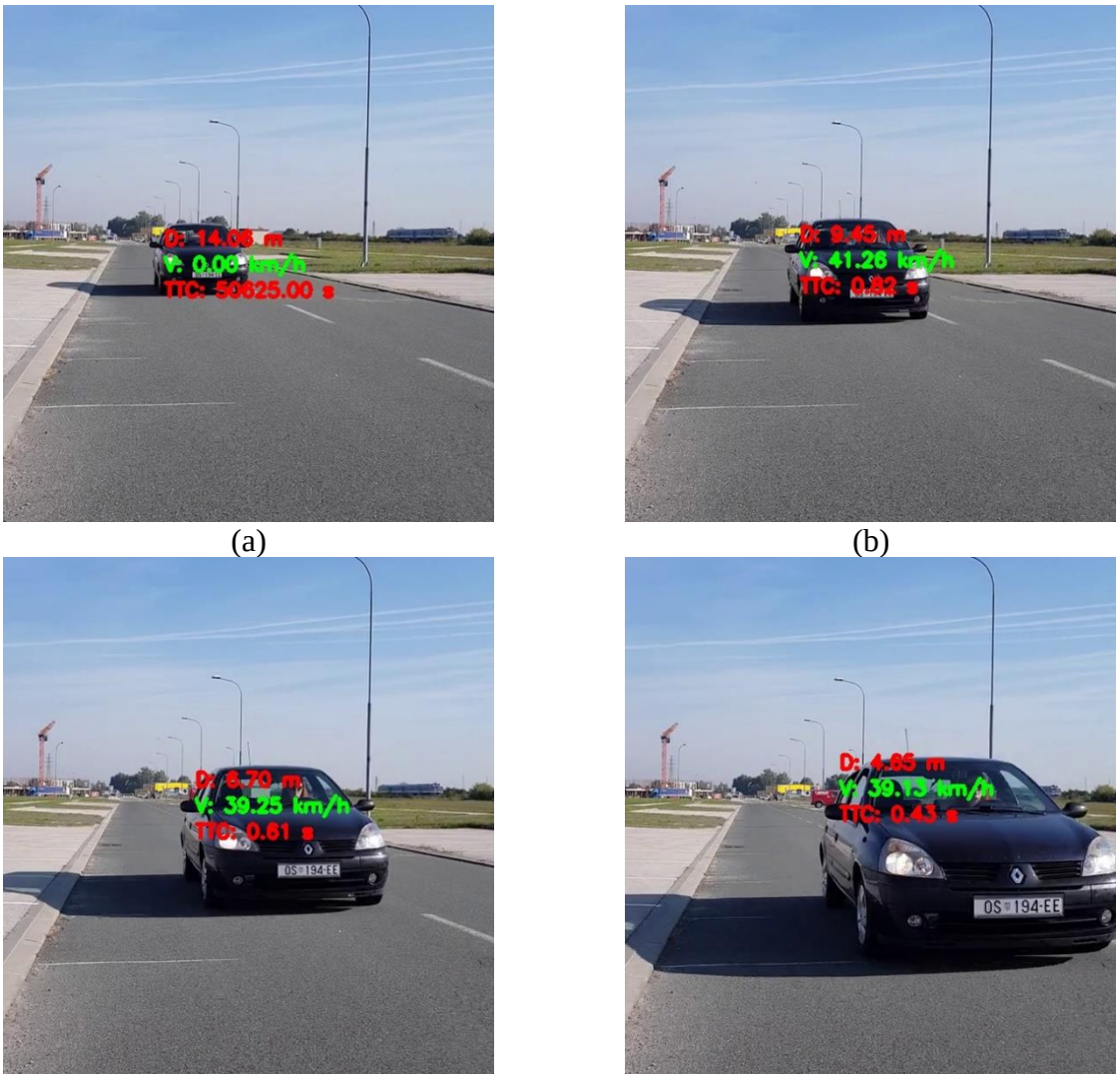
Sl. 4.5. *Test 5, video sekvenca s8_3_test.mp4, kamera Samsung Galaxy S8, automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Slika 4.6. prikazuje 6. test. Automobil nije detektiran na 10m i 15m zbog toga što kamera nije postavljena paralelno s cestom i loše istreniranog detektora. Položaj kamere jako utječe na točnost detekcije. Na ostale dvije udaljenosti procijenjena udaljenost, brzina i TTC odstupaju od stvarnih vrijednosti zbog pomicanja mobilnog telefona prilikom snimanja.



Sl. 4.6. *Test 6, video sekvenca s8_4_test.mp4, kamera Samsung Galaxy S8, automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Test 7 prikazan je sliko 4.7. Automobil je uspješno detektiran na svim testiranim udaljenostima. Na udaljenosti 15m od kamere algoritam ne daje rezultat za procjenu brzine i TTC-a jer se prva detekcija dogodila tek na 15m udaljenosti od kamere te algoritam nije uspio izvršiti procjenu. Mirnijom video snimkom i dobrim kutom snimanja dobivaju se zadovoljavajući rezultati (malo odstupanje od stvarnih vrijednosti) na ostalim testiranim udaljenostima do kamere.

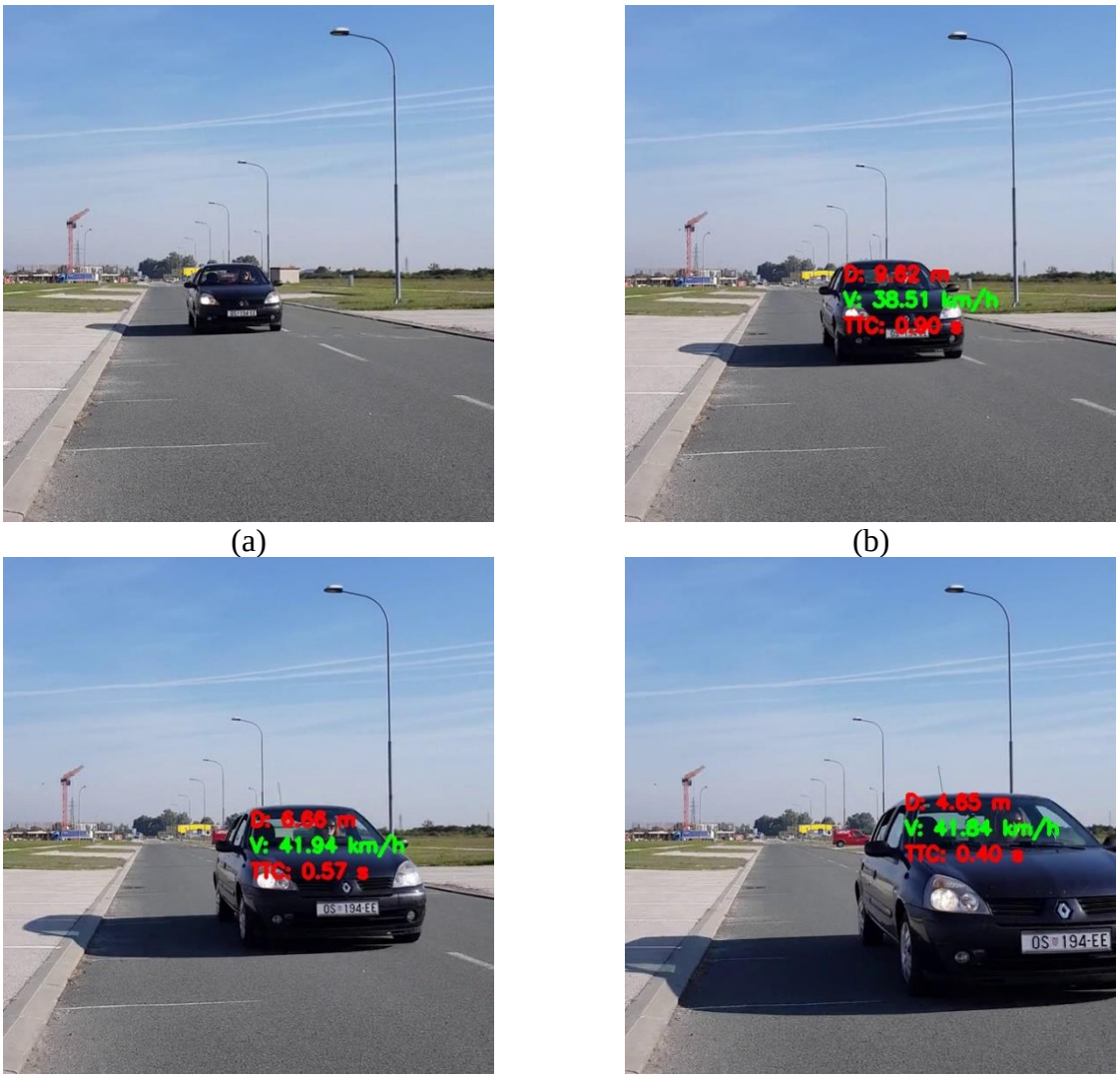


(a) (b)

(c) (d)

Sl. 4.7. *Test 7, video sekvenca s8_5_test.mp4, kamera Samsung Galaxy S8, automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Slikom 4.8. prikazan je test 8. Boljim detektorom omogućena bi bila detekcija vozila na 15m udaljenosti od kamere. Na ostalim testiranim udaljenostima dobivaju se rezultati s malim odstupanjem od stvarne vrijednosti zbog mirne video snimke i dobro postavljenog kuta snimanja.



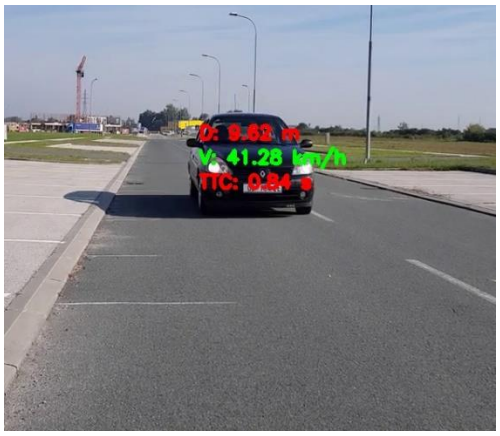
(a) (b)

(c) (d)

Sl. 4.8. *Test 8, video sekvenca s8_6_test.mp4, kamera Samsung Galaxy S8, automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Slika 4.9. prikazuje test 9. Nadolazeće vozilo detektirano je na 15m. Brzina i TTC su procijenjeni vrlo loše jer se procjena izvršila na samo jednoj izmjeni susjednih video okvira. Za procjenu brzine se koristi medijan 10 uzastopnih izračuna brzine nadolazećeg vozila. Loša procjena brzine uzrokuje lošu procjenu TTC-a. Na ostalim udaljenostima procjena je izvršena zadovoljavajuće sa malim odstupanjima od stvarnih vrijednosti.

(a)



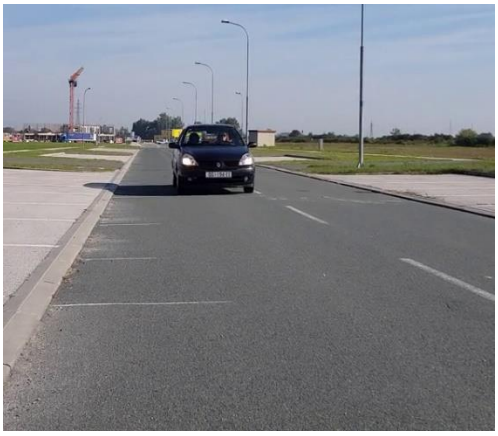
(b)

(c)

(d)

Sl. 4.9. *Test 9, video sekvenca s8_7_test.mp4, kamera Samsung Galaxy S8, automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Slika 4.10. prikazuje test 10. Algoritam ne detektira automobil na 15m udaljenosti od kamere zbog lošeg detektora. Na ostalim testiranim udaljenostima algoritam ima malo veće odstupanje prilikom procjene udaljenosti, brzine i TTC-a zbog malo lošije postavljenog kuta snimanja



(a)

(b)

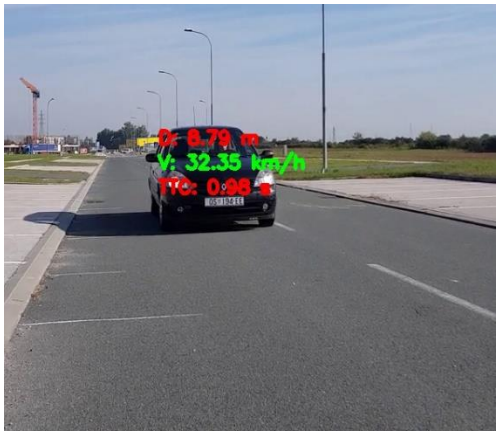
(c)

(d)

Sl. 4.10. *Test 10, video sekvenca s8_8_test.mp4, kamera Samsung Galaxy S8, automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Slika 4.11. prikazuje test 11. Položaj kamere nije paralelan s cestom te uzrokuje lošije rezultate procjene. Automobil je uspješno detektiran na svim testiranim udaljenostima. Procjena brzine i TTC-a nije uspješno izvršena na 15m udaljenosti od kamere.

(a)



(b)

(c)

(d)

Sl. 4.11. *Test 11, video sekvenca s8_9_test.mp4, kamera Samsung Galaxy S8, automobil na (a) 15m, (b) 10m, (c) 7m, (d) 5m udaljenosti od kamere*

Tablica 4.1. prikazuje rezultate procjene udaljenosti algoritma za procjenu brzine vozila vidljivog u retrovizoru. U nekim video sekvencama algoritam nije uspio detektirati automobil na udaljenosti od 15m i na jednoj video sekvenci algoritam ne uspijeva detektirati vozilo na udaljenosti od 10m. Jako osvjetljenje na vozilu nekada utječe na detekciju. Procijenjene udaljenosti odstupaju od stvarne udaljenosti vozila do kamere zbog aproksimacije tijekom proračuna udaljenosti. Automobil se ne nalazi u sredini leće kamere i prosječna širina automobila je 1.8m. Nadalje pogreška nastaje zbog kuta snimanja i pomicanja mobilnog telefona prilikom snimanja video sekvenci.

Tab. 4.1. *Rezultati procjene udaljenosti algoritma za procjenu brzine vozila vidljivog u retrovizoru*

Video sekvenca	Procijenjena udaljenost na 15m od kamere [m]	Procijenjena udaljenost na 10m od kamere [m]	Procijenjena udaljenost na 7m od kamere [m]	Procijenjena udaljenost na 5m od kamere [m]
a5_1_test.mp4	11,05	7,52	5,32	3,90
a5_2_test.mp4	11,96	8,23	5,94	4,06
s8_1_test.mp4	-	9,78	6,94	4,96
s8_2_test.mp4	-	8,86	6,25	4,57
s8_3_test.mp4	13,24	8,86	6,18	4,26
s8_4_test.mp4	-	-	6,78	4,38
s8_5_test.mp4	14,06	9,45	6,70	4,65
s8_6_test.mp4	-	9,62	6,66	4,65
s8_7_test.mp4	14,24	9,62	6,66	4,61
s8_8_test.mp4	-	9,22	6,15	4,25
s8_9_test.mp4	12,78	8,79	6,15	4,23

Tablica 4.2. prikazuje procjenu brzine algoritma za procjenu brzine vozila vidljivog u retrovizoru. Kao i kod procjene udaljenosti algoritam nije uspio procijeniti brzine na nekoliko video sekvenci kada se automobil nalazio na 15m od kamere te na jednoj kada je bio na 10m od kamere. Procjena brzine radi se na temelju procjene udaljenosti nadolazećeg vozila, što znači da zbog pogreške koja nastaje prilikom procjene udaljenosti nastaje i pogreška prilikom procjene brzine. Iako algoritam ima pogrešku procjene brzine, često ta procjena ne odstupa puno od stvarne brzine automobila. Procjena brzine ovisi o brzini izmjene video okvira videa. Za različite video sekvence taj broj je različit pa je potrebno prilagoditi taj parametar video sekvenci na kojoj se vrši testiranje. Položaj kamere i postavljeni kut snimanja utječu na procjenu brzine preko procjene udaljenosti. Algoritam se može koristiti za procjenu brzine vozila vidljivog u retrovizoru ukoliko je kamera koja snima u pokretu i kreće se nekom brzinom. Tada je potrebno zbrojiti brzinu kojom se kamera kreće sa procijenjenom brzinom kako bi se dobio rezultat procjene brzine.

Tab. 4.2. *Rezultati procjene brzine algoritma za procjenu brzine vozila vidljivog u retrovizoru (stvarna brzina vozila je 45 km/h)*

Video sekvenca	Procijenjena brzina na 15m od kamere [km/h]	Procijenjena brzina na 10m od kamere [km/h]	Procijenjena brzina na 7m od kamere [km/h]	Procijenjena brzina na 5m od kamere [km/h]
a5_1_test.mp4	30,05	30,18	29,61	30,26
a5_2_test.mp4	33,54	32,76	32,36	29,95
s8_1_test.mp4	-	37,88	39,17	38,77
s8_2_test.mp4	-	35,81	36,15	35,38
s8_3_test.mp4	31,48	39,22	38,75	37,40
s8_4_test.mp4	-	-	36,07	36,43
s8_5_test.mp4	0,00	41,26	39,25	39,13
s8_6_test.mp4	-	38,51	41,94	41,64
s8_7_test.mp4	28,67	41,28	42,61	42,16
s8_8_test.mp4	-	36,20	38,30	37,68
s8_9_test.mp4	0,00	32,65	32,64	33,77

Tablica 4.3. prikazuje rezultate procjene TTC-a algoritma za procjenu brzine vozila vidljivog u retrovizoru. Pogreška procjene TTC-a nastaje zbog pogrešaka u procjeni udaljenosti i brzine nadolazećeg vozila jer se TTC dobiva omjera udaljenosti i brzine vozila. Na temelju procjene TTC-a vozač ima informaciju koliko vremena ima za određeni manevar (npr. promjena prometne trake). U dva slučaja algoritam daje beskonačni iznos TTC-a zbog toga što u tim slučajevima brzina nije procijenjena. Stvarne vrijednosti TTC-a na udaljenostima 15m, 10m, 7m, 5m su redom 1.2s, 0.8s, 0.56s i 0.4s.

Tab. 4.3. *Rezultati procjene TTC-a algoritma za procjenu brzine vozila vidljivog u retrovizoru*

Video sekvenca	Procijenjeno TTC-a na 15m od kamere [s]	Procijenjeno TTC-a na 10m od kamere [s]	Procijenjeno TTC-a na 7m od kamere [s]	Procijenjeno TTC-a na 5m od kamere [s]
a5_1_test.mp4	1,32	0,90	0,64	0,46
a5_2_test.mp4	1,26	0,90	0,66	0,49
s8_1_test.mp4	-	0,93	0,64	0,46
s8_2_test.mp4	-	0,89	0,62	0,47
s8_3_test.mp4	1,51	0,81	0,57	0,41
s8_4_test.mp4	-	-	0,68	0,43
s8_5_test.mp4	∞	0,82	0,61	0,43
s8_6_test.mp4	-	0,90	0,57	0,40
s8_7_test.mp4	1,79	0,84	0,56	0,39
s8_8_test.mp4	-	0,92	0,58	0,41
s8_9_test.mp4	∞	0,96	0,67	0,45

Pogreške tijekom testiranja nastaju zbog različitog položaja kamere u odnosu na cestu, kutu snimanja u odnosu na smjer kretanja nadolazećeg vozila i pomicanja kamere tokom snimanja. Na

temelju procijenjene udaljenosti procjenjuje se brzina nadolazećeg vozila, a na temelju njih se procjenjuje TTC. Što znači smanjenjem pogreške procjene udaljenosti nadolazećeg vozila smanjuje se pogreška svih rezultata algoritma (procjena brzine i TTC). Definiranjem formule koja u obzir uzima i položaj vozila na leći prilikom procjene udaljenosti nadolazećeg vozila moguće je postići bolje rezultate algoritma.

Video sekvence koje se su se koristile prilikom testiranja mogu se pronaći u P.3.2. koji se nalazi na CD-u.

5. ZAKLJUČAK

Tri su glavna djela algoritma za procjenu brzine vozila koje se vidi u retrovizoru: detekcija nadolazećeg vozila, praćenje detektiranog vozila i procjena brzine istog. Detekcija kao ključni korak algoritma osigurava mogućnost procjene brzine. Praćenje detektiranog vozila između okvira omogućava rad algoritma u stvarnom vremenu jer praćenje detektiranog objekta između dva video okvira zahtjeva mnogo manje procesorskog vremena nego detekcija istog. Zbog pretpostavke da se detektirano vozilo nalazi na sredini leće kamere unosi se pogreška u konačnu procjenu brzine. Implementacijom algoritma na razvojnu platformu, kao što je ALPHA ploča, omogućilo bi se testiranje algoritma u stvarnim uvjetima gdje bi se najbolje mogli vidjeti nedostaci algoritma. Implementacija nije uspješno napravljena zbog ograničenja koja su navedena u 3. poglavlju.

Boljim detektorom omogućila bi se bolja detekcija vozila na većoj udaljenosti. Osvijetljenost vozila je također jedan od problema prilikom detekcije jer jako osvjetljeno vozilo neće uvijek biti detektirano.

Propagacija pogreške procjene udaljenosti utječe na sve rezultate algoritma (procjenu brzine i TTC-a), što znači da se boljom procjenom udaljenosti vozila koje se dolazi odzada omogućuju bi se bolji rezultati algoritma. Na procjenu udaljenosti nadolazećeg vozila utječe položaj kamere u odnosu na cestu, kutu snimanja u odnosu na nadolazeće vozilo i problem pomicanja kamere tokom snimanja. Ukoliko bi kamera postavila na pravi način osigurali bi se točniji rezultati procjene udaljenosti, brzine i TTC-a.

LITERATURA

[1] L. Grammatikopoulos, G. Karras, E. Petsa (GR), Automatic estimation of vehicle speed from uncalibrated video sequences, Internacional symposium on modern tehnologies, education and professional practice in geodesy and related fields, str. 332 – 338, Sofia, 3 – 4 Studeni, 2005

[2] B. Makawana, P. Goel, Moving Vehicle Detection and Speed Measurement in Video Sequence, International Journal of Engineering Research & Tehnology (IJERT), str. 3534 – 3537, Vol. 2 Issue 10, Listopad, 2013

[3] Z. Czapla, Vehicle Vehicle Speed Estimation with the Use of Gradient-Based Image Conversion into Binary Form, Signal Processing Algorithms, Architectures, Arrangements, and Aplications, str. 213 – 216, 20. – 22. Rujan, 2017, Poznań, Poljska

[4] Y.M.P.C.Y. Bandara, U.A.A. Nirishika, T.U. Abeygunawardhane, Frame Feature Tracking for Speed Estimation, International Conference on Advances in ICT for Emerging Regions, str. 29 – 34, 2014.

[5] P. Cika, M. Zukal, Z. Libis, M.K. Dutta, Tracking and Speed Estimation of Selected Object in Video Sequence, TSP, str. 881 – 884, 2013.

[6] A.Y.G. Rao, S.K. Kumar, H.S. Amaresh, H.V. Chirag, Real-Time Speed Estimation of Vehicles from Uncalibrated View-Independent Traffic Cameras, IEEE 2015,

[7] H. Kaneko, M. Morimoto, K. Fujii, Vehicle Speed Estimation by In-Vehicle Camera, WAC 2012, str. 1 – 6, 2012

[8] P. Viola, M. Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, Accepted Conference on Computer Vision and Pattern Recognition 2001

[9] S. Puttemans, Cascade Classifier Training, OpenCV, [28.8.2018]

[10] Z. Kalal, K. Mikolajczyk, J. Matas, Forward-Backward Error: Automatic Detection of Tracking Failures, International Conference on Pattern Recognition, str. 1-4, Istanbul, Turkey, 23. – 26. Kolovoz, 2010.

[11] Device specification, Samsung Galaxy S8 Exynos – Specification, https://www.devicespecifications.com/en/model/402341d2 [3.9.2018]

[12] T. Dempsey, Compare digital camera sensor sizes: 1″-Type, 4/3, APS-C, full frame 35mm, 27. Listopada 2013., T. Dempsey, http://photoseek.com/2013/compare-digital-camera-sensor-sizes-full-frame-35mm-aps-c-micro-four-thirds-1-inch-type/ [3.9.2018]

[13] Device specification, Samsung Galaxy A5 (2017) – Specification, https://www.devicespecifications.com/en/model/534a4087 [3.9.2018]

[14] T. Schiesser, Samsung Galaxy A5 Review Page 5: Camera: 13-Megapixel IMX135, 11. Veljača 2015., 2018 TechSpot, Inc., https://www.techspot.com/review/957-samsung-galaxy-a5/page5.html [3.9.2018.]

[15] M. Grum, How do I calculate the distance of an object in a photo?, Stack Exchange Inc., https://photo.stackexchange.com/questions/12434/how-do-i-calculate-the-distance-of-an-object-in-a-photo, [30.8.2018]

[16] SAE, Automated Driving – Levels of Driving Automation are Defined in New SAE International Standard J3016, izdano: 2014.

[17] NVIDIA DRIVE, NVIDIA Corporation, https://www.nvidia.com/en-us/self-driving-cars/drive-platform/ [10.9.2018]

[18] Advanced Driver Assistance System (ADAS), Renesas Electronics Corporation, https://www.renesas.com/us/en/solutions/automotive/adas.html [10.9.2018]

[19] Drive Data Platform, Qualcomm Technologies, Inc., https://www.qualcomm.com/solutions/automotive/drive-data-platform [10.9.2018.]

[20] RT-RK, Automotive Machine Vision ALPHA reference board on Texas Instruments SoCs, RT-RK, http://www.rt-rk.com/download/rt-rk_ALPHA_ADAS_board.pdf [6.9.2018]

[21] J. Krause, M. Stark, J. Deng, L. Fei-Fei, 3D Object Representations for Fine-Grained 4th IEEE Workshop on 3D Representation and Recognition, at ICCV 2013 (3dRR-13), Sydney, Australija, 8. Prosinac 2013.

SAŽETAK

U ovom diplomskom radu opisan je algoritam za procjenu brzine vozila koje se vidi u retrovizoru. Algoritam za procjenu brzine vozila sastoji se od predobrade video okvira, detekcije vozila u video okviru, praćenju detektiranog vozila i procjene brzine vozila. Predobradom se osigurava veća brzina obrade video okvira tako što se mijenja veličina video okvira (ukoliko je video okvir velikih dimenzija) i samo se obrađuje dio video okvira u kojem se može pronaći vozilo. Položaj vozila u video okviru dobiva se detekcijom, a praćenje detektiranog vozila između okvira omogućuje veću brzinu obrade. Procjena brzine vozila podijeljena je u dva djela: na procjenu udaljenosti i na procjenu brzine na temelju prevaljenog puta između dva video okvira. Algoritam za procjenu brzine vozila testiran je na jedanaest testnih video sekvenci te su postignuti dosta uspješni rezultati, uz postojanje prostora za napredak u radu algoritma.

Ključne riječi: predobrada, detekcija, praćenje, procjena udaljenosti, procjena brzine, ADAS

EVALUATION OF THE SPEED OF THE VEHICLE BASED ON THE CAMERA IMAGE THAT REPLACES THE REARVIEW MIRROR

ABSTRACT

This paper describes vehicle speed estimation algorithm that can be seen in rearview mirror. The vehicle speed estimation algorithm consists of preprocessing, vehicle detection, tracking vehicle between frames and speed estimation. Preprocessing ensures higher speed of video frame processing by changing the size of the video frame (if frame is large enough) and only processing part of video frame in which vehicle can be found. The position of vehicle in video frame is gained by detection and tracking the detected vehicle between the frames ensures higher processing rate. The vehicle speed estimation is divided into two parts: distance estimation between vehicle and camera and speed estimation based on prevailing distance between two video frames. The Vehicle speed estimation algorithm has been tested in eleven test video sequences and quite successful results have been achieved, with the space for progress in algorithm work.

Keywords: preprocessing, detection, tracking, distance estimation, speed estimation, ADAS

ŽIVOTOPIS

Luka Ćosić rođen je 21.10.1994. godine u Osijeku. Osnovnu školu završio je u Tenji nakon čega završava III. Gimnaziju u Osijeku. Upisao preddiplomski sveučilišni studij na fakultetu elektrotehnike u Osijeku gdje dobiva nagradu za postignut uspjeh u studiranju. Na trećoj godini studija 2016. godine sudjelovao je na Elektrijadi u Rimini u Italiji. Nakon toga upisuje diplomski sveučilišni studij na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Trenira rukomet od četvrtog razreda osnovne škole. Posjeduje vozačku dozvolu B kategorije.

U Osijeku, rujan 2018.

Potpis:

_____

PRILOZI

P.3.1. *Python* skripta *procjena_brzine.py*

```
import cv2
import statistics as st
import sys
import time
import os

carCascade = cv2.CascadeClassifier('detektor.xml')
lk_params = dict(winSize = (15, 15), maxLevel = 2, criteria =
(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

def calculateDistance(detectedObjectWidth, cameraParams):
        realObjectWidth = 1.8
        return ((cameraParams['focalLength'] * realObjectWidth *
cameraParams['frameWidth']) / (detectedObjectWidth *
cameraParams['sensorWidth']))

def estimateVelocity(distance):
    velocity = distance / 0.0392
    velocityKMH = 0.001 + (3.6 * velocity)
    return velocityKMH

def tracker(capture, camera):
    video = cv2.VideoCapture(capture)

    if camera == 's8' or camera == 'S8':
        cameraParams = {
                        'focalLength':0.0042,
                        'frameWidth':800,
                        'sensorWidth':0.005376
                        }
        cropParams = [200, 800, 400, 1000]
        # cropParams = [300, 900, 150, 720]
    elif camera == 'a5' or camera == 'A5':
        cameraParams = {
                        'focalLength':0.0036,
                        'frameWidth':800,
                        'sensorWidth':0.00594
                        }
        # cropParams = [300, 900, 150, 720]
        cropParams = [100, 720, 400, 1000]
    elif camera == 'p10' or camera == 'P10':
        cameraParams = {
                        'focalLength':0.00395,
                        'frameWidth':800,
```

```
                    'sensorWidth':0.006415
                    }
        cropParams = [200, 600, 200, 1000]
    elif camera == '23b' or camera == '23B':
        cameraParams = {
                    'focalLength':0.0022,
                    'frameWidth':800,
                    'sensorWidth':0.005376
                    }
    else:
        print("Unsuported camera! Exiting...")
        return

    red = (0, 0, 255)
    green = (0, 255, 0)
    frameCounter = 0
    currentCarID = 0
    previousDistance = {}
    currentDistance = {}
    velocity = {}
    speed = {}
    medianSpeed = {}
    position = {}
    carTracker = {}

    while True:
        if cv2.waitKey(26) == 27:
            break
        rc, image = video.read()
        if type(image) == type(None):
            print(frameCounter)
            break

        image = image[cropParams[0]:cropParams[1], cropParams[2]:cropParams[3]]
        resultImage = image.copy()
        frameCounter = frameCounter + 1

        carIDtoDelete = []

        for carID in carTracker.keys():
            trackedPosition = carTracker[carID].update(image)
            t_x, t_y, t_w, t_h = trackedPosition[1]
            t_x = int(t_x)
            t_y = int(t_y)
            t_w = int(t_w)
            t_h = int(t_h)

            x_center = t_x + 0.5 * t_w
            y_center = t_y + 0.5 * t_h
```

```
            if x_center <= 0 or y_center <= 0:
                  carIDtoDelete.append(carID)
            elif x_center >= cropParams[3]:
                carIDtoDelete.append(carID)

        for carID in carIDtoDelete:
            # print('Tracker deleted: ' + str(carID) + '.')
            carTracker.pop(carID, None)

        if not (frameCounter % 10):
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            cars = carCascade.detectMultiScale(gray, 1.19, 15)
            for (_x, _y, _w, _h) in cars:
                x = int(_x) + 7
                y = int(_y) + 7
                w = int(_w) - 5
                h = int(_h) - 5

                x_bar = x + 0.5 * w
                y_bar = y + 0.5 * h

                matchCarID = None

                for carID in carTracker.keys():
                    trackedPosition = carTracker[carID].update(image)
                    t_x, t_y, t_w, t_h = trackedPosition[1]
                    t_x = int(t_x)
                    t_y = int(t_y)
                    t_w = int(t_w)
                    t_h = int(t_h)

                    t_x_bar = t_x + 0.5 * t_w
                    t_y_bar = t_y + 0.5 * t_h

                    if ((t_x <= x_bar <= (t_x + t_w)) and (t_y <= y_bar <= (t_y +
t_h)) and (x <= t_x_bar <= (x + w)) and (y <= t_y_bar <= (y + h))):
                        matchCarID = carID

                if matchCarID is None and w > 50:
                    bbox = (x, y, w, h)
                    # if bbox[0] < 500 or bbox[1] < 100:
                    tracker = cv2.TrackerMedianFlow_create()
                    tracker.init(image, bbox)
                    carTracker[currentCarID] = tracker
                    previousDistance[currentCarID] = calculateDistance(bbox[2],
cameraParams)
                    currentDistance[currentCarID] = calculateDistance(bbox[2],
cameraParams)
```

```
                    speed[currentCarID] = []
                    currentCarID = currentCarID + 1

        for carID in carTracker.keys():
            trackedPosition = carTracker[carID].update(image)
            t_x, t_y, t_w, t_h = trackedPosition[1]
            t_x = int(t_x)
            t_y = int(t_y)
            t_w = int(t_w)
            t_h = int(t_h)

            t_x_bar = t_x + 0.5 * t_w
            t_y_bar = t_y + 0.5 * t_h
            bbox = (t_x, t_y, t_w, t_h)
            currentDistance[carID] = calculateDistance(bbox[2], cameraParams)
            position[carID] = bbox
            # cv2.rectangle(resultImage, (t_x, t_y), (t_x + t_w, t_y + t_h), red,
3)

        for carID in carTracker.keys():

            cv2.putText(resultImage,
                        'D: ' +  str("%.2f" % round(currentDistance[carID], 2)) +
' m',
                        (position[carID][0], position[carID][1] + 20),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                        red,
                        3
                        )

            velocity[carID] = estimateVelocity(previousDistance[carID] -
currentDistance[carID])
            if len(speed[carID]) == 10:
                speed[carID].pop(0)

            speed[carID].append(velocity[carID])
            medianSpeed[carID] = st.median(speed[carID])

            cv2.putText(resultImage,
                        'V: ' +  str("%.2f" % round(medianSpeed[carID], 2)) + '
km/h',
                        (position[carID][0], position[carID][1] + 50),
                        cv2.FONT_HERSHEY_SIMPLEX,
                        0.75,
                        green,
                        3
                        )

            cv2.putText(resultImage,
```

```
                        'TTC: ' +  str("%.2f" % round(currentDistance[carID] /
(medianSpeed[carID] / 3.6), 2)) + ' s',
                        (position[carID][0], position[carID][1] + 80),
                        cv2.FONT_HERSHEY_SIMPLEX,
                        0.75,
                        red,
                        3
                        )

            previousDistance[carID] = currentDistance[carID]

        cv2.imshow('image' , resultImage)
    cv2.destroyAllWindows()
if __name__ == '__main__':
    capture = str(sys.argv[1])
    camera = str(sys.argv[2])
    start = time.time()
    tracker(capture, camera)
    print(time.time() - start)
```

P.3.2. Video sekvence

Video sekvence korištene prilikom testiranja nalaze se na CD-u.