Detekcija objekata ispred vozila pomoću kamere na prednjoj strani vozila

Ciberlin, Juraj

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

Sveučilišni studij

DETEKCIJA OBJEKATA ISPRED VOZILA POMOĆU KAMERE NA PREDNJOJ STRANI VOZILA

Diplomski rad

Juraj Ciberlin

Osijek, 2018.

SADRŽAJ

1. UVOD

Automobilska industrija je u posljednjih desetak godina doživjela znatne promjene. Razina autonomije vozila svakoga dana sve više raste i autonomna vozila postaju svakodnevica. Također, postoje sustavi koji omogućavaju minimiziranje ljudskih pogrešaka prilikom vožnje, smanjenje prometnih nesreća i samim time omogućavaju ljudima veću sigurnost. Naziv takvih sustava je ADAS (engl. *Advanced Driver Assistance System*). Autonomna vozila trebaju obavljati svoje funkcije bez potrebe za intervencijom vozača, a kako bi to bilo moguće treba im ugraditi određeni oblik umjetne inteligencije. Zbog toga je potrebno raspolagati različitim senzorima koji daju informacije iz okoline i računalnim sustavom koji na temelju tih informacija i odgovarajućih algoritama donosi odluke prilikom autonomne vožnje.

Jedna od osnovnih funkcija koju autonomna vozila trebaju obavljati je detekcija objekata ispred vozila. U svakom trenutku treba postojati informacija o objektima ispred njih. Navedenu funkciju je moguće ostvariti korištenjem informacija dobivenih pomoću slike s kamere na prednjoj strani vozila. Na temelju detektiranih objekata (npr. vozila) računalni sustav može pravovremeno donijeti odluku u vožnji (npr. skrenuti kako bi izbjegao vozilo).

Zadatak diplomskog rada je izraditi računalni algoritam za detekciju objekata ispred vozila na osnovu slike dobivene pomoću kamere na prednjoj strani vozila. Algoritam treba detektirati različite objekte tipične za situacije u prometu (vozila, pješake, prometne znakove, semafore) i za svaki okvir video zapisa treba pohraniti lokaciju i dimenzije detektiranih objekata. Algoritam je izrađen u višem programskom jeziku (Python i C++) te je prikazana njegova implementacija na Alpha ploču. Za implementaciju navedenog rješenja korišteni su dostupni paketi za razvoj programske podrške (engl. *Software Development Kit*) i obradu slike.

Rad je strukturiran na sljedeći način. U drugom poglavlju pod nazivom Postojeće metode detekcije objekata opisani su postojeći načini detekcije objekata na slikama. Nabrojane su neke metode detekcije objekata te su odabrane tri metode i detaljnije su opisane. Odabrane metode su: Viola-Jones algoritam za detekciju objekata, YOLO algoritam za detekciju objekata i metoda detekcije objekata korištenjem TensorFlow programskog aplikacijskog sučelja za detekciju objekata. Izvršeno je testiranje navedenih metoda, njihova usporedba te su izdvojene prednosti i nedostatci pojedinih metoda.

U trećem poglavlju pod nazivom Izrada algoritma za detekciju i praćenje objekata opisana je instalacija i podešavanje svih potrebnih biblioteka za izradu algoritma za detekciju i praćenje objekata. Detaljno su opisane OpenCV i dlib biblioteke. Opisana je izrada detektora vozila, pješaka, semafora i prometnih znakova. Opisana je izrada računalnog algoritma za detekciju i

praćenje objekata u video zapisu u Python i C++ programskom jeziku. Za izradu algoritma u C++ programskom jeziku korištena je OpenCV biblioteka, a u Python programskom jeziku, osim navedene biblioteke korištena je i dlib biblioteka.

U četvrtom poglavlju pod nazivom Evaluacija izrađenog algoritma za detekciju i praćenje objekata u video zapisu izvršeno je testiranje izrađenih detektora na testnom skupu slika i prikazani su njihovi rezultati. Također, izvršena je evaluacija izrađenih algoritama za detekciju i praćenje objekata u video zapisu izrađenih u Python i C++ programskim jezicima.

U petom poglavlju pod nazivom Implementacija algoritma za detekciju i praćenje objekata na Alpha ploču opisana je Alpha ploča i njezina upotreba. Osim toga, opisan je Vision SDK, paket za razvoj programske podrške za TDAx SoC-ove koji se nalaze na Alpha ploči. Također su opisani problemi prilikom implementacije vlastitog algoritma za detekciju i praćenje objekata na Alpha ploču te je na kraju iznesen zaključak.

2. POSTOJEĆE METODE DETEKCIJE OBJEKATA

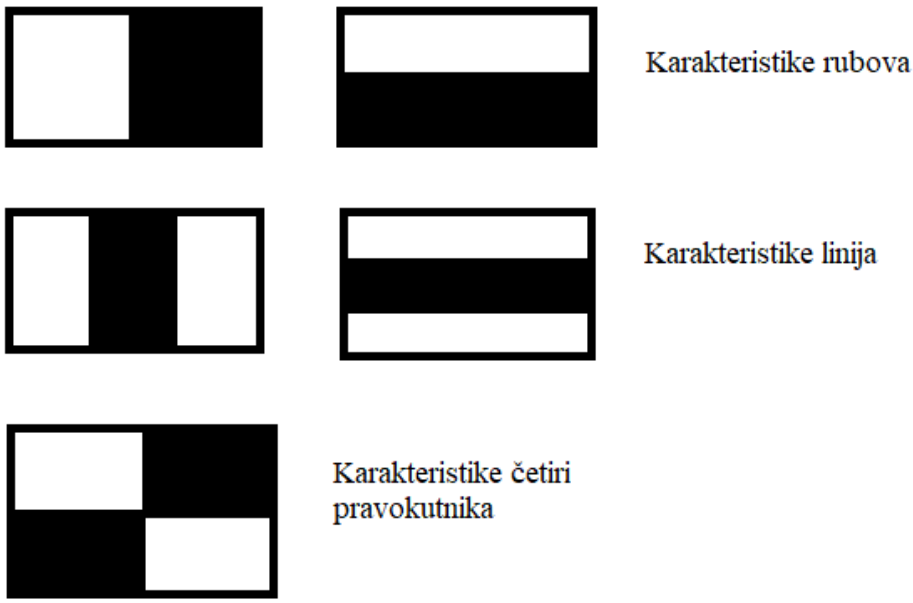
U ovom poglavlju opisani su postojeći načini detekcije objekata. U računalnom vidu detekcija objekata predstavlja jedan od ključnih problema. Detekcija objekata je proces pronalaska objekata (npr. vozila, ljudi, prometnih znakova, lica, itd.) na slici ili video zapisu. Detekciju objekata na slici ili video zapisu moguće je provesti pomoću različitih metoda: segmentacija slike, Viola-Jones algoritam za detekciju objekata, duboko učenje (engl. *deep learning*), detekcija objekata temeljena na značajkama (engl. *feature-based object detection*), itd. U potpoglavlju 2.1. detaljno je opisan Viola-Jones algoritam koji je razvijen u svrhu detekcije lica, ali je korišten i za detekciju različitih objekata [1]. U potpoglavlju 2.2. opisan je YOLO (engl. *You Only Look Once*) algoritam za detekciju objekata [2]. U potpoglavlju 2.3. opisana je metoda detekcije objekata korištenjem TensorFlow programskog aplikacijskog sučelja za detekciju objekata [3]. Na slici 2.1. prikazan je primjer detekcije objekata. Svakom objektu pripada odgovarajući pravokutnik i klasa. Automobili su označeni crvenim pravokutnikom, a čovjek zelenim pravokutnikom.

Sl. 2.1. *Primjer detekcije objekata*

2.1. Viola-Jones algoritam za detekciju objekata

U ovom potpoglavlju opisan je Viola-Jones algoritam za detekciju objekata objavljen 2001. godine [1]. Algoritam je temeljen na strojnom učenju. Korišten je AdaBoost algoritam za učenje koji služi za odabir manjeg broja Haarovih karakteristika iz većeg skupa. Uveden je novi oblik prikaza slike nazvan integralna slika (engl. *integral image*). Integralna slika omogućava brže računanje Haarovih karakteristika. Osim toga, uvedena je i kaskada klasifikatora (engl. *cascade of classifiers*). Ona smanjuje vrijeme računanja i tako ubrzava proces detekcije.

Prilikom procesa učenja korištene su pozitivne i negativne slike. Pozitivne slike predstavljaju slike na kojima je prikazan traženi objekt, a negativne slike predstavljaju slike na kojima nije prikazan traženi objekt. Prilikom procesa učenja odabran je manji broj Haarovih karakteristika iz većeg skupa. Postoje tri vrste Haarovih karakteristika. Na slici 2.2. prikazan je izgled Haarovih karakteristika, postoje karakteristike rubova, linija i četiri pravokutnika [1].

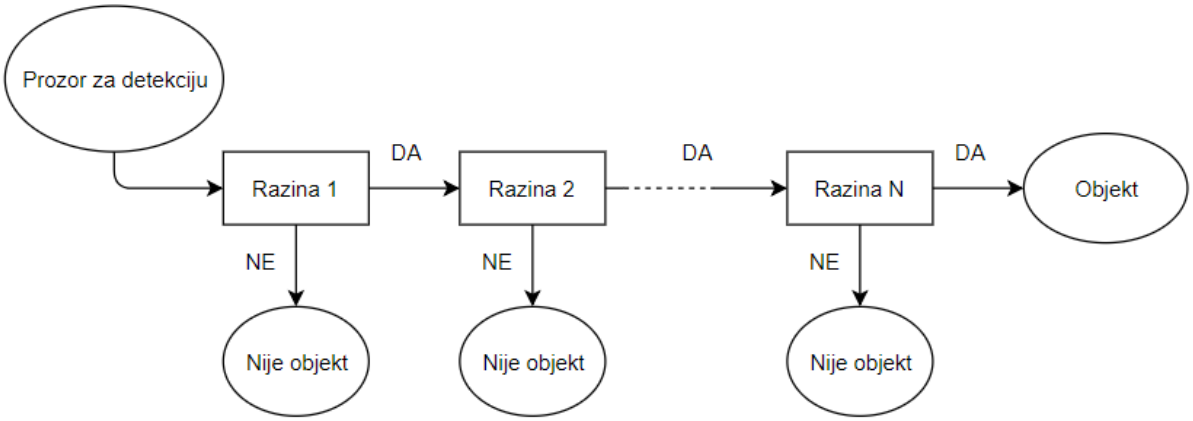


Sl. 2.2. *Izgled Haarovih karakteristika [1]*

Svaka karakteristika predstavlja jedinstvenu vrijednost koja je izračunata oduzimanjem sume piksela unutar bijelih pravokutnika i sume piksela unutar crnih pravokutnika. Na početku procesa učenja određena je dimenzija prozora za detekciju (npr. 24 x 24 piksela) na koju su skalirane sve slike koje sudjeluju u procesu učenja, a ona označava najmanju dimenziju objekta kojega je kasnije u procesu testiranja moguće detektirati na slici. Osim toga, dimenzija prozora za detekciju predstavlja omjer širine i duljine objekta kojega je kasnije moguće detektirati. Ako je dimenzija prozora za detekciju prilikom procesa učenja 24 x 24 piksela, tada je prilikom procesa testiranja

moguće detektirati jedino objekte minimalne dimenzije 24 x 24 piksela i objekte kojima je omjer širine i duljine 1 : 1, odnosno iste su širine i duljine. Zatim je na svakoj slici pozitivnog skupa slika pronađen veći skup Haarovih karakteristika. Nakon toga, korištenjem Adaboost algoritma odabran je manji skup relevantnih Haarovih karakteristika. Prvi korak Adaboost algoritma je pridruživanje vrijednosti slikama, ovisno o tome je li slika pozitivna ili negativna. Ako je slika pozitivna, tada joj je pridružena vrijednost 1, a ako je negativna, tada joj je pridružena vrijednost 0. Sljedeći korak je inicijalizacija težina. Nakon toga, unaprijed zadani broj puta normalizirane su težine, za svaku karakteristiku izrađen je slabi klasifikator ograničen na korištenje samo jedne karakteristike, izračunata je njegova stopa pogreške, odabran je klasifikator s najmanjom stopom pogreške te su ažurirane težine. Nakon svake iteracije odabran je novi slabi klasifikator te je tako odabrano više slabih klasifikatora. Posljednji korak je izračunavanje završnog klasifikatora. Završni klasifikator, odnosno detektor predstavlja binarni, jaki klasifikator sastavljen od više slabih klasifikatora.

Drugi važan pojam koji uvodi Viola-Jones algoritam je kaskada klasifikatora. Njezina najvažnija zadaća je smanjenje vremena računanja i ubrzanje procesa detekcije. Kaskada klasifikatora sastavljena je od više razina, gdje svaka razina predstavlja skupinu slabih klasifikatora. Slabi klasifikatori unutar svake skupine dobiveni su procesom učenja u kojemu je korišten AdaBoost algoritam. Prozor za detekciju čije su dimenzije prethodno definirane prolazi kroz cijelu sliku pomičući se po jedan piksel sve dok ne prođe kroz cijelu sliku. Nakon prolaska kroz cijelu sliku, slika je smanjena te prozor za detekciju ponovno prolazi kroz cijelu sliku. Proces je ponovljen dok ulazna slika ne postane manja od prozora za detekciju. Svaki dio slike kroz koji prođe prozor za detekciju također prolazi i kroz kaskadu klasifikatora koja označava trenutni dio slike kao pozitivan ili negativan. Ako je dio slike označen kao pozitivan, tada postoji mogućnost da je unutar toga dijela slike objekt, a ako je označen kao negativan, tada se smatra kako unutar toga dijela slike nije objekt. Također, ako je dio slike označen kao negativan, tada je klasifikacija trenutnog dijela slike završena i prozor za detekciju prelazi na sljedeći dio slike. Ako je dio slike označen kao pozitivan, tada taj dio prelazi na sljedeću razinu kaskade klasifikatora. Ako dio slike prođe kroz sve razine kaskade klasifikatora, tada je taj dio slike klasificiran kao objekt. Zaključeno je da je uvođenjem razina smanjeno vrijeme računanja i ubrzan proces detekcije zato što je dio slike koji je označen negativno odbačen što je brže moguće [1]. Također, većina dijelova slike ne sadrži traženi objekt. Na slici 2.3. prikazan je način rada kaskade klasifikatora [1].



Sl. 2.3. *Način rada kaskade klasifikatora [1]*

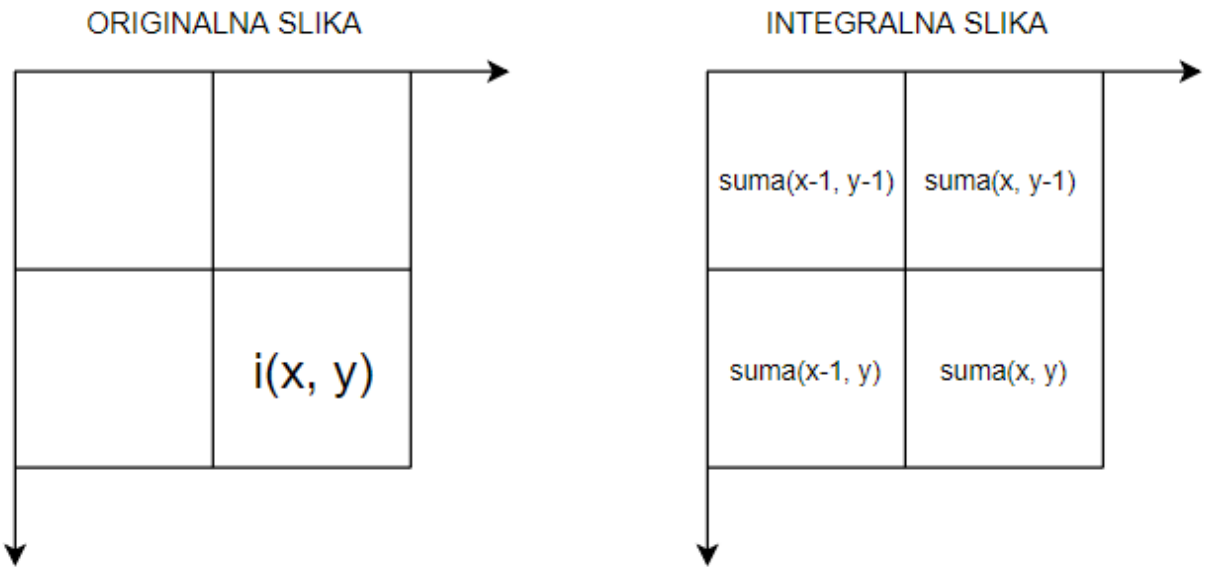
Treći važan pojam koji uvodi Viola-Jones algoritam je integralna slika koja je uvedena radi bržeg računanja Haarovih karakteristika. Integralna slika je prvi puta predstavljena 1984. godine [4]. Izrađena je iz originalne slike korištenjem izraza (2-1). Originalna slika mora biti u nijansama sive boje. Prema [1, 4], izraz za izračun sume vrijednosti piksela integralne slike na poziciji (x, y) je sljedeći:

$$suma(x, y) = i(x, y) + suma(x - 1, y) + suma(x, y - 1) - suma(x - 1, y - 1) \quad (2\text{-}1)$$

gdje je:

- $i(x, y)$ - vrijednost piksela na trenutnoj lokaciji,
- $suma(x\text{-}1, y)$ - suma piksela na lokaciji lijevo od trenutnog piksela,
- $suma(x, y\text{-}1)$ - suma piksela iznad trenutnog piksela,
- $suma(x\text{-}1, y\text{-}1)$ - suma piksela na lokaciji lijevo iznad trenutnog piksela.

Na slici 2.4. prikazan je primjer korištenja izraza (2-1) [4].



Sl. 2.4. *Primjer korištenja izraza (2-1) [4]*

Na temelju izraza (2-1), na slici 2.5. prikazan je primjer izrađene integralne slike iz originalne slike.

ORIGINALNA SLIKA

2	1	5	3	2
4	5	4	2	1
1	3	1	2	2
2	6	4	2	1
3	5	2	1	3

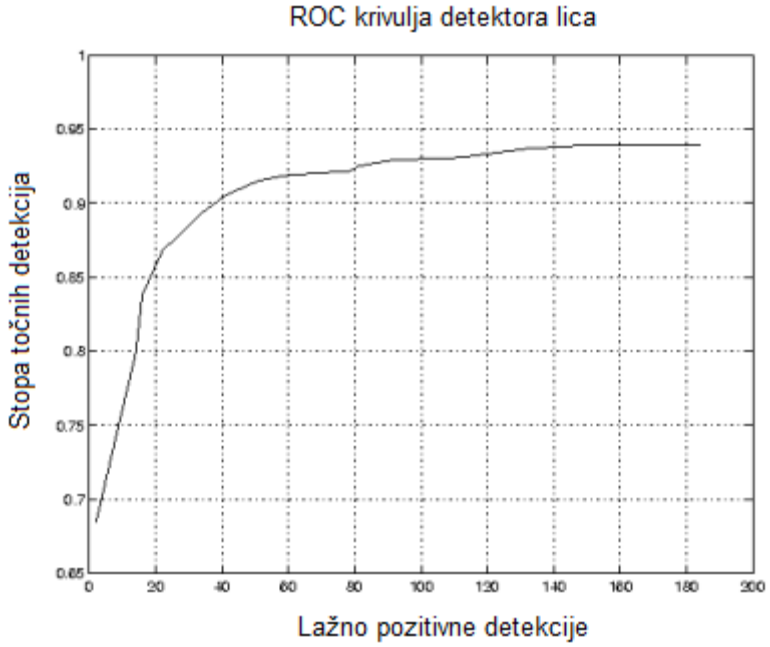
→

INTEGRALNA SLIKA

2	3	8	11	13
6	12	21	26	29
7	16	26	33	38
9	24	38	47	53
12	32	48	58	67

Sl. 2.5. *Primjer izrađene integralne slike iz originalne slike*

Na slici 2.6. prikazana je ROC (engl. *Receiver Operating Characteristic*) krivulja detektora lica objavljenog u [1], a testiranog na MIT-CMU skupu slika [1, 5]. ROC krivulja prikazuje odnos broja lažno pozitivnih (engl. *False Positive*) detekcija i stope točnih detekcija.



Sl. 2.6. *ROC krivulja detektora lica objavljenog u [1], a testiranog na MIT-CMU skupu slika [1, 5]*

Testiranje Viola-Jones algoritma izvršeno je korištenjem detektora lica dostupnog unutar OpenCV biblioteke [6]. OpenCV biblioteka detaljnije je opisana u potpoglavlju 3.1.1. Testiranje je izvršeno korištenjem procesora Intel Core i5-6200U i slika razlučivosti 640 x 480 piksela te brzina detekcije iznosi 25 slika u sekundi.

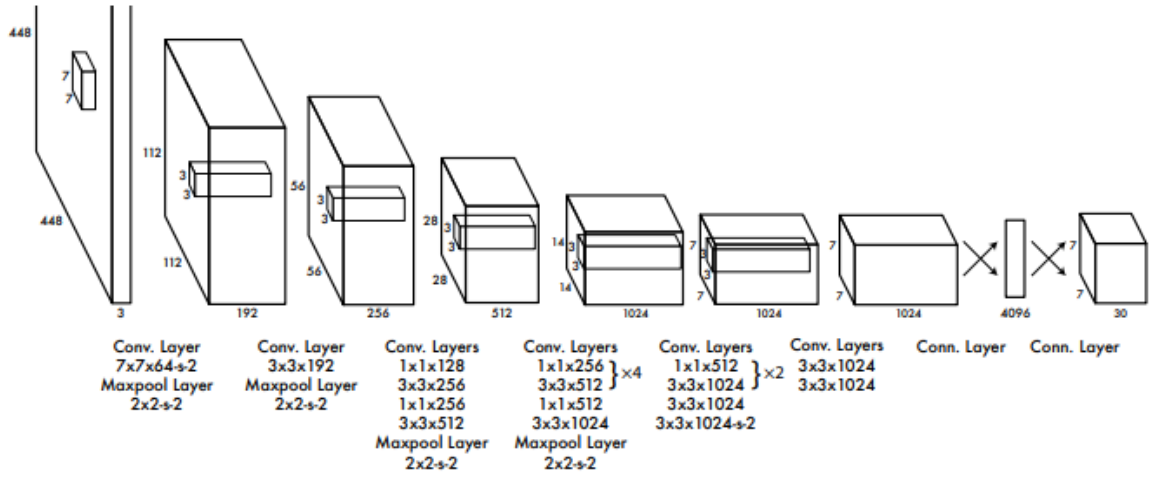
Također, postoje i određena ograničenja i nedostatci Viola-Jones algoritma. Prema [7, str. 141.], nedostatak algoritma je vrijeme trajanja procesa učenja odnosno treniranja. Osim toga, ograničenje algoritma je unaprijed određen omjer širine i duljine detektiranog objekta. Unaprijed određen omjer širine i duljine prozora za detekciju također je i omjer širine i duljine detektiranog objekta.

2.2. YOLO algoritam za detekciju objekata

U ovom potpoglavlju opisan je YOLO algoritam za detekciju objekata objavljen 2016. godine [2]. Navedeni algoritam predstavlja novi pristup detekciji objekata. Algoritam radi tako da ulaznu sliku podijeli na mrežu (engl. *grid*) istih dimenzija. Ako je središte objekta smješteno unutar određenog dijela mreže, tada je taj dio mreže zadužen za detekciju tog objekta. Svaki dio mreže predviđa određeni broj graničnih pravokutnika (engl. *bounding box*) i postotak koji predstavlja vjerojatnost kako je upravo unutar toga graničnog pravokutnika objekt. Svaki granični pravokutnik sastavljen je od pet predikcija. Jedna predikcija predstavlja vjerojatnost, a ostale predikcije

prikazane su sljedećim oznakama: *x*, *y*, *w*, *h*. Oznake *x* i *y* predstavljaju lokaciju središta graničnog pravokutnika u odnosu na dio mreže u kojemu je smješten. Oznake *w* i *h* predstavljaju dimenziju objekta u odnosu na cijelu sliku.

Postoje dva YOLO modela za detekciju objekata, a to su osnovni i Fast YOLO model [2]. Osnovni model sastavljen je od 24 konvolucijska sloja nakon kojih slijede dva potpuno povezana sloja. Osim toga, koristi 1 x 1 redukcijski sloj nakon kojega slijede 3 x 3 konvolucijski slojevi. Fast YOLO model koristi manji broj konvolucijskih slojeva u odnosu na osnovni model. Koristi devet u odnosu na 24 konvolucijska sloja osnovnog modela. Također, koristi i manji broj filtera unutar tih slojeva. Za razliku od veličine mreže, svi ostali parametri u procesima treniranja i testiranja u navedena dva modela su isti. Na slici 2.7. prikazana je arhitektura osnovnog YOLO modela za detekciju objekata [2].

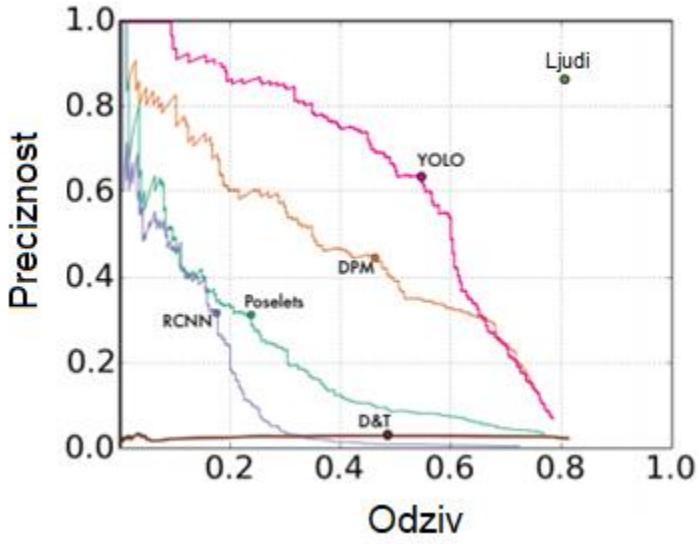


Sl. 2.7. *Arhitektura osnovnog YOLO modela za detekciju objekata [2]*

Za razliku od nekih drugih metoda detekcije objekata, YOLO model obavlja proces učenja odnosno treniranja na cijelim slikama. Prije pokretanja procesa učenja potrebno je podesiti niz parametara. Proces učenja se odvija na skupu slika za učenje, odnosno skupu označenih slika. Na početku su sve dostupne slike u skupu za učenje skalirane na 448 x 448 piksela, a nakon toga je pokrenuta konvolucijska mreža na slikama. YOLO model predviđa više graničnih pravokutnika po dijelu mreže. Za vrijeme procesa učenja odnosno treniranja modela jedan granični pravokutnik zadužen je točno za jedan objekt. Proces treniranja dostupnih YOLO modela izvršen je na COCO [8] i Pascal VOC [9] skupu slika.

Na slici 2.8. prikazane su krivulje koje prikazuju odnos preciznosti i odziva YOLO metode za detekciju objekata i drugih metoda na Picasso skupu slika [2, 10]. Također, na slici 2.8. vidljivo

je kako YOLO metoda za detekciju objekata prikazuje najbolje rezultate na korištenom testnom skupu slika.



Sl. 2.8. *Krivulje koje prikazuju odnos preciznosti i odziva YOLO metode za detekciju objekata i drugih metoda na Picasso skupu slika [2, 10]*

Prema [2], osnovni YOLO model testiran korištenjem grafičke kartice NVIDIA GeForce Titan X i Pascal VOC [9] testnog skupa slika obavlja detekciju objekata brzinom 45 slika u sekundi, dok Fast YOLO model testiran korištenjem iste grafičke kartice i istog testnog skupa slika brzinom 155 slika u sekundi, no s manjom točnošću od osnovnog modela. Zbog brzine detekcije objekata zaključeno je kako bi oba YOLO modela korištenjem prethodno navedene grafičke kartice mogla biti korištena za detekciju objekata u autonomnim vozilima. Također, provedenim testiranjem osnovnog YOLO modela korištenjem grafičke kartice NVIDIA GeForce GTX 950M i slika razlučivosti 1280 x 720 piksela obavljena je detekcija objekata brzinom dvije slike u sekundi. Testiranjem Fast YOLO modela korištenjem iste grafičke karticu i slika iste razlučivosti, brzina detekcije objekata iznosi deset slika u sekundi. Zbog toga, zaključeno je kako YOLO algoritam obavlja detekciju objekata u stvarnom vremenu jedino uz korištenje grafičke kartice čija je snaga jednaka ili veća od snage grafičke kartice NVIDIA GeForce Titan X ili uz smanjenje točnosti detekcije. Programski kod potreban u procesima treniranja i testiranja YOLO algoritma dostupan je svima, kao i prethodno navedeni modeli [11].

Postoje određena ograničenja i nedostatci YOLO algoritma. Prema [2], neka od ograničenja algoritma su nedovoljno točna detekcija malih objekata i ako su objekti smješteni blizu jedan drugoga, tada ih je teško sve detektirati.

2.3. Metoda detekcije objekata korištenjem TensorFlow programskog aplikacijskog sučelja za detekciju objekata

U ovom potpoglavlju detaljno je opisana metoda detekcije objekata korištenjem TensorFlow programskog aplikacijskog sučelja za detekciju objekata. Prema [12], TensorFlow je biblioteka programske podrške, otvorenog koda, a služi za izračune visokih performansi. Također, podržava strojno i duboko učenje. Prema [13], nastalo je TensorFlow programsko aplikacijsko sučelje za detekciju objekata.

TensorFlow programsko aplikacijsko sučelje za detekciju objekata je okvir otvorenog koda temeljen na TensorFlow biblioteci, a služi za jednostavniju izradu, učenje i razvoj modela za detekciju objekata [3]. Prema [3], moguće je izvršiti instalaciju TensorFlow programskog aplikacijskog sučelja za detekciju objekata.

Nakon izvršene instalacije moguće je testirati dostupne modele. Neki od dostupnih modela su: SSD (engl. *Single Shot Multibox Detector*) način detekcije s MobileNet konvolucijskom neuronskom mrežom, SSD način detekcije s Inception V2 konvolucijskom neuronskom mrežom, Faster RCNN način detekcije s Resnet 101 konvolucijskom neuronskom mrežom. Za provedbu testiranja treba odabrati dostupni model i sliku ili video zapis na kojima treba detektirati objekte. Proces učenja dostupnog modela temeljenog na SSD načinu detekcije s MobileNet konvolucijskom neuronskom mrežom izvršen je na COCO [8] skupu slika te omogućava detekciju 90 različitih objekata. Ako objekt kojega treba detektirati nije unutar tih 90 objekata, tada treba modificirati model, odnosno pretrenirati ga kako bi mogao detektirati traženi objekt. TensorFlow programsko aplikacijsko sučelje za detekciju objekata zahtjeva da je označeni skup slika korišten u procesu učenja spremljen u TFRecord datotečnom formatu. Ako je označeni skup slika spremljen u XML datotečnom formatu kao što je Pascal VOC [9] skup slika, tada postoji Python programski kod koji pretvara označeni skup slika iz XML datotečnog formata u TFRecord datotečni format. Ako je označeni skup slika spremljen u drugačijem datotečnom formatu kao što je YAML, tada treba napisati vlastiti programski kod koji pretvara označeni skup slika u TFRecord datotečni format. Za izradu vlastitog označenog skupa slika treba koristiti alat za označavanje slika. Postoje različiti alati za označavanje slika, a neki od njih su: LabelImg, LabelMe, FastAnnotationTool, itd. Za navedeni slučaj najbolje je koristiti LabelImg alat za označavanje slika zato što sprema označene slike u isti format kao što je spremljen i Pascal VOC skup slika. Kao što je prethodno navedeno, Pascal VOC označeni skup slika spremljen je u XML datotečnom formatu te ga je moguće pretvoriti u TFRecord datotečni format korištenjem dostupnog Python programskog koda. Nakon izrade vlastitog označenog skupa slika treba pretrenirati dostupni model kako bi detektirao

objekte označene u vlastitom skupu slika. Potrebno je ukloniti posljednih 90 neurona klasifikacijskog sloja mreže i zamijeniti ih novim slojem. Neuroni klasifikacijskog sloja predstavljaju objekte koje treba detektirati. Nakon toga treba pokrenuti proces učenja modela. Za prikaz napredovanja procesa učenja treba koristiti TensorBoard. TensorBoard je korišten kao vizualizacijski alat. Nakon završetka procesa treniranja, moguće je koristiti izrađeni model za detekciju objekata na slikama ili video zapisu.

U tablici 2.1. prikazana je brzina detekcije i srednja vrijednost preciznosti odnosno mAP (engl. *mean Average Precision*) odabranih dostupnih modela za detekciju objekata testiranih na COCO [8] testnom skupu slika [14]. Testiranje je provedeno korištenjem grafičke kartice NVIDIA GeForce Titan X [14]. Srednja vrijednost preciznosti odnosno mAP predstavlja prosjek maksimalne preciznosti prilikom različitih vrijednosti odziva. Također, u tablici 2.1. vidljivo je kako od odabranih dostupnih modela najveću brzinu detekcije i najveću srednju vrijednost preciznosti odnosno mAP ima model temeljen na Faster RCNN načinu detekcije s Resnet 101 konvolucijskom neuronskom mrežom.

Tab. 2.1. *Brzina detekcije i srednja vrijednost preciznosti (mAP) odabranih dostupnih modela za detekciju objekata testiranih na COCO [8] testnom skupu slika [14].*

Naziv modela	**Brzina detekcije [ms]**	**mAP**
SSD, MobileNet	30	21
SSD, Inception V2	42	24
Faster RCNN, Resnet 101	106	32
Faster RCNN, Resnet 50	89	30
Faster RCNN, Inception V2	58	28

Testiranje dostupnog modela temeljenog na SSD načinu detekcije s MobileNet konvolucijskom neuronskom mrežom izvršeno je korištenjem procesora Intel Core i5-6200U i slika razlučivosti 800 x 600 piksela te brzina detekcije objekata iznosi deset slika u sekundi. Brzinu detekcije objekata moguće je povećati korištenjem grafičke kartice umjesto procesora. Provedenim testiranjem zaključeno je kako korištenjem navedenog modela i navedenih komponenti nije moguće ostvariti detekciju objekata u stvarnom vremenu. Rad u stvarnom vremenu moguće je ostvariti korištenjem grafičke kartice ili smanjenjem razlučivosti slike. Smanjenjem razlučivosti slike smanjena je točnost detekcije objekata.

3. IZRADA ALGORITMA ZA DETEKCIJU I PRAĆENJE OBJEKATA

U ovom poglavlju opisana je izrada algoritma za detekciju i praćenje objekata. Prilikom izbora metode za detekciju objekata važna je točnost i brzina detekcije, a osim toga i mogućnost pretvorbe algoritma u C programski jezik kako bi bila moguća kasnija implementacija i testiranje učinkovitosti rješenja na odgovarajućoj ugradbenoj platformi. Zbog toga, odabran je Viola-Jones algoritam te su na temelju njega izrađeni odgovarajući detektori. U potpoglavlju 3.1. detaljno je opisano podešavanje i instalacija svih potrebnih biblioteka za izradu algoritma za detekciju i praćenje objekata. U potpoglavlju 3.2. opisano je prikupljane i označavanje slika korištenih prilikom procesa učenja. U potpoglavlju 3.3. detaljno je opisana izrada detektora. U potpoglavlju 3.4. detaljno je opisana izrada računalnog algoritma za detekciju i praćenje objekata u video zapisu u Python programskom jeziku. U potpoglavlju 3.5. detaljno je opisana izrada računalnog algoritma za detekciju i praćenje objekata u video zapisu u C++ programskom jeziku.

3.1. Podešavanje i instalacija potrebnih biblioteka

U ovom potpoglavlju detaljno je opisano podešavanje i instalacija potrebnih biblioteka za izradu algoritma za detekciju i praćenje objekata. Korištene su OpenCV i dlib biblioteke.

3.1.1. OpenCV biblioteka

OpenCV (engl. *Open Source Computer Vision*) je biblioteka otvorenog koda koja sadrži programske funkcije koje su korištene u računalnom vidu i strojnom učenju [15]. Usmjerena je na aplikacije koje obavljaju svoje funkcije u stvarnom vremenu. Napisana je u C i C++ programskim jezicima. Također, podržava korištenje na više operacijskih sustava, a to su: MS Windows, Linux, Mac OS i Android. Sadrži sučelja za C++, Python i MATLAB programske jezike. OpenCV biblioteka također sadrži i veliki broj optimiziranih algoritama, među kojima su algoritmi korišteni u računalnom vidu i strojnom učenju. Navedeni algoritmi korišteni su za detekciju objekata, praćenje detektiranih objekata, praćenje pokreta, spajanje više slika u jednu, itd.

Za korištenje OpenCV biblioteke izvršena je instalacija iste. Prikazane su upute na temelju kojih je izvršena instalacija OpenCV 3.1.0. biblioteke na Ubuntu 16.04 operacijskom sustavu. Navedena inačica OpenCV biblioteke korištena je za izradu vlastitog rješenja. Na početku je izvršena instalacija pkg-config alata. OpenCV biblioteka sadrži funkcije i algoritme koji izvršavaju radnje na različitim formatima slika i zato je izvršena instalacija biblioteka koje omogućavaju učitavanje različitih formata slika (libpng, libjpeg, libtiff, libjasper). Osim različitih formata slika,

funkcije i algoritmi unutar OpenCV biblioteke izvršavaju radnje i na video zapisima i zato je izvršena instalacija biblioteka koje omogućavaju učitavanje video zapisa i obradu slika dobivenih pomoću kamere (libavformat, libavcodec, libswscale, libv4l, libxvidcore, libx264). Nakon toga, izvršena je instalacija GTK biblioteke koja omogućava izradu grafičkog korisničkog sučelja. Potom je izvršena instalacija biblioteke čija je zadaća optimizirati različite funkcionalnosti koje koriste funkcije unutar OpenCV biblioteke (libatlas-base-dev). Nakon toga, preuzet je OpenCV modul dostupan na GitHubu [16]. Osim osnovnog OpenCV modula, preuzet je i dodatni modul opencv_contrib [17]. Za razliku od osnovnog modula, dodatni modul sadrži dodatne funkcije koje su još u razvoju te nisu dovoljno testirane. Potom je postavljeno Python okruženje, a prvi korak postavljanja istog je instalacija pip alata koji omogućava instalaciju svih potrebnih Python paketa. Također, postavljeno je i Python virtualno okruženje. Prilikom postavljanja Python virtualnog okruženja navedena je inačica Python programskog jezika (python3). Nakon toga, korištenjem naredbe *workon naziv_virtualnog_okruženja* pristupljeno je virtualnom okruženju te je izvršena instalacija Python paketa NumPy korištenjem pip alata. Unutar virtualnog okruženja također je izvršena konfiguracija i prevođenje OpenCV biblioteke. Nakon toga, moguće je koristiti OpenCV biblioteku. Ako je OpenCV biblioteka korištena u aplikacijama pisanim u Python programskom jeziku, tada je izvršeno prevođenje programskog koda unutar virtualnog okruženja. Ako je OpenCV biblioteka korištena u aplikacijama pisanim u C++ programskom jeziku, tada je izvršeno prevođenje programskog koda korištenjem naredbe *g++ naziv_aplikacije.cpp `pkg-config –cflags –libs opencv`*. Prethodno navedeni koraci instalacije detaljnije su opisani u [18].

3.1.2. Dlib biblioteka

Dlib je višeplatformska biblioteka, opće namjene, otvorenog koda, napisana u C++ programskom jeziku [19]. Neke od glavnih značajki biblioteke su: dokumentacija, portabilnost izvornog koda, algoritmi strojnog učenja, obrada slike, itd. Za razliku od nekih drugih projekata otvorenog koda, dlib biblioteka sadrži potpunu dokumentaciju svih klasa i funkcija. Također, postoji veliki broj primjera aplikacija napisanih u C++ i Python programskim jezicima koje koriste navedenu biblioteku. Biblioteka je testirana na Linux, MS Windows i Mac OS X operacijskim sustavima. Sadrži algoritme strojnog učenja korištene za detekciju objekata, klasifikaciju, regresiju, itd. Osim toga, sadrži i funkcije za obradu slike (npr. detekciju rubova, morfološke operacije). Izvorni kod dlib biblioteke dostupan je na GitHubu [20]. Za korištenje Python modula dlib biblioteke izvršena je instalacija istog korištenjem pip alata. Naredba za instalaciju je *pip*

install dlib. Ako je dlib biblioteka korištena u aplikacijama pisanim u C++ programskom jeziku, tada je izvršeno prevođenje programskog koda korištenjem g++ prevoditelja i C++11 standarda.

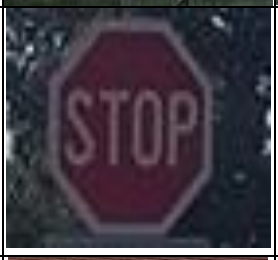
3.2. Prikupljanje i označavanje slika

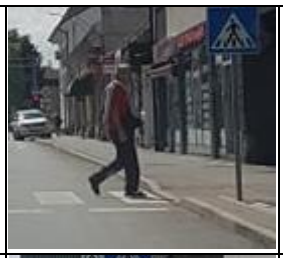
Kao što je prethodno navedeno, izrađeni detektori temeljeni su na Viola-Jones algoritmu. Budući da treba detektirati objekte ispred vozila, treba izraditi detektore vozila, pješaka, prometnih znakova i semafora. Zbog nedostataka Viola-Jones algoritma navedenih u potpoglavlju 2.1., detektor vozila podijeljen je na detektor prednje i stražnje strane automobila, detektor bočne strane automobila i detektor prednje i stražnje strane autobusa i kamiona. Također, detektor prometnih znakova podijeljen je na detektore prometnih znakova opasnosti, izričitih naredbi koje karakterizira plava pozadina, izričitih naredbi koje karakterizira crveni rub, prometnog znaka obveznog zaustavljanja i prometnog znaka raskrižja s cestom koja ima prednost prolaska. Zbog toga, izrađeno je deset detektora.

Za izradu detektora korištene su pozitivne i negativne slike. Pozitivne slike sadrže označene objekte koje treba detektirati, a negativne slike ne sadrže tražene objekte. Prvi korak je prikupljanje slika. Moguće je koristiti vlastite slike te na njima označiti tražene objekte ili dostupne označene skupove slika. Za izradu potrebnih detektora korišteni su: vlastiti skup slika, negativni skup slika dostupan na GitHubu [21] i dijelovi Cars [22], Udacity [23], German Traffic Sign Recognition Benchmark [24] i INRIA Person [25] skupova slika. Ako treba detektirati pješake, tada treba prikupiti slike pješaka različitog spola, različitih visina, itd. Zbog toga, prikupljene su slike traženog objekta iz različitih kutova i različitih dimenzija, odnosno prikupljene su raznolike slike traženog objekta. U tablici 3.1. prikazani su primjeri prikupljenih pozitivnih slika korištenih prilikom procesa učenja.

Tab. 3.1. *Primjeri prikupljenih pozitivnih slika korištenih prilikom procesa učenja.*

Naziv detektora	**Primjeri slika korištenih prilikom procesa učenja**
Detektor prednje i stražnje strane automobila	

Detektor bočne strane automobila			
Detektor prednje i stražnje strane autobusa i kamiona			
Detektor prometnih znakova opasnosti			
Detektor prometnih znakova izričitih naredbi koje karakterizira plava pozadina			
Detektor prometnih znakova izričitih naredbi koje karakterizira crveni rub			
Detektor prometnog znaka obveznog zaustavljanja			
Detektor prometnog znaka raskrižja s cestom koja ima prednost prolaska			

Detektor pješaka			
Detektor semafora			

Nakon prikupljanja slika, na njima su označeni traženi objekti. Osim toga, prikupljene su i slike na kojima je jedino prikazan traženi objekt. Za označavanje slika treba koristiti neki od dostupnih alata, kao što su: LabelImg, LabelMe, OpenCV Annotation Tool, itd. Nakon označavanja slika, oznake objekata spremljene su u tekstualnu datoteku korištenjem sljedećeg zapisa: *naziv_slike broj_objekata x_koordinata y_koordinata širina_objekta duljina_objekta*. Ako je označeno više od jednog objekta na slici, tada je korišten sljedeći zapis: *naziv_slike broj_objekata x_koordinata_objekta_1 y_koordinata_objekta_1 širina_objekta_1 duljina_objekta_1 x_koordinata_objekta_N y_koordinata_objekta_N širina_objekta_N duljina_objekta_N*. Ako alat za označavanje slika koristi drugačiji zapis oznaka, tada treba napisati vlastiti programski kod koji pretvara određeni zapis oznaka u traženi zapis oznaka. Za označavanje slika korišten je OpenCV Annotation Tool čiji je programski kod dostupan na GitHubu [26]. Odabran je navedeni alat zato što koristi traženi zapis oznaka. Negativne slike nisu označene jer ne sadrže traženi objekt. Jedino su nabrojane putanje i nazivi svih negativnih slika te su spremljeni u tekstualnu datoteku. Osim toga, prilikom prikupljanja negativnih slika prikupljene su slike koje sadrže što više detalja, a ne slike koje sadrže samo nebo, more, livadu, itd. Tako je detaljnije prikazano što nije traženi objekt, odnosno što ne treba detektirati na slikama. Slike mogu biti različitih dimenzija, jedini uvjet je da budu jednake ili veće dimenzije od dimenzije prozora za detekciju.

Nakon prikupljanja i označavanja slika treba izraditi uzorak pozitivnih slika. Zbog toga, izrađena je VEC datoteka. Za izradu navedene datoteke korištena je opencv_createsamples aplikacija čiji je programski kod dostupan na GitHubu [27]. Prilikom izrade VEC datoteke navedeni su: putanja i naziv tekstualne datoteke u kojoj su spremljene oznake objekata na slikama, putanja i naziv izlazne VEC datoteke, broj pozitivnih uzoraka koje treba izraditi te širina i duljina izlaznog uzorka u pikselima. Nakon izrade uzoraka pozitivnih slika, pokrenut je proces učenja.

3.3. Izrada detektora

U ovom potpoglavlju detaljno je opisana izrada detektora. Izrada detektora temeljena je na OpenCV biblioteci koja je detaljnije opisana u potpoglavlju 3.1.1. Također, detektor je izrađen na temelju Viola-Jones algoritma koji je detaljnije opisan u potpoglavlju 2.1.

Programski kod aplikacije koja obavlja proces učenja dostupan je na GitHubu [28]. Prilikom pokretanja procesa učenja navedeni su: putanja izlaznog detektora, putanja i naziv prethodno izrađene VEC datoteke, putanja i naziv tekstualne datoteke u kojoj su spremljene putanje i nazivi negativnih slika, broj pozitivnih i negativnih uzoraka korištenih u svakoj razini kaskade, broj razina kaskade, veličina međuspremnika za izračunavanje vrijednosti značajki (što je veća veličina međuspremnika, to je proces učenja kraći), način korištenja Haarovih karakteristika, širina i duljina uzorka koja mora biti identična širini i duljini prilikom izrade VEC datoteke. Nakon završetka procesa učenja, izrađena je XML datoteka koja predstavlja kaskadu klasifikatora detaljnije opisanu u potpoglavlju 2.1. Kaskada klasifikatora sastavljena je od prethodno navedenog broja razina, gdje svaka razina predstavlja skupinu slabih klasifikatora, a zajedno predstavljaju završni klasifikator, odnosno detektor.

U tablici 3.2. prikazane su informacije o procesu učenja za svaki od izrađenih detektora. Proces učenja svih detektora izvršen je korištenjem procesora Intel Core i5-6200U. Prilikom pokretanja procesa učenja svih detektora odabrana je ista veličina međuspremnika za izračunavanje vrijednosti značajki (1024 MB) i isti način korištenja Haarovih karakteristika (ALL). Način korištenja Haarovih karakteristika ALL znači da su korištene sve Haarove karakteristike. Vrijeme trajanja učenja ovisi o broju kaskada i broju pozitivnih i negativnih slika. Što je veći broj kaskada i broj pozitivnih i negativnih slika, to je duže vrijeme trajanja procesa učenja. Broj kaskada za svaki od izrađenih detektora odabran je empirijskim putem. Dimenzija izlaznog uzorka predstavlja najmanju dimenziju kasnije detektiranog objekta, a omjer dimenzija izlaznog uzorka ovisi o omjeru kasnije detektiranog objekta. Pretpostavljeno je da prednja i stražnja strana automobila te svi prometni znakovi najčešće imaju isti omjer širine i duljine. Također, pretpostavljeno je da bočna strana automobila najčešće ima omjer širine i duljine 2 : 1, prednja i stražnja strana autobusa i kamiona 2 : 3, a pješak i semafor 1 : 3. Osim toga, pretpostavljeno je kako je minimalna dimenzija semafora najčešće manja od minimalne dimenzije pješaka te je zbog toga odabrana manja dimenzija izlaznog uzorka semafora od dimenzije izlaznog uzorka pješaka.

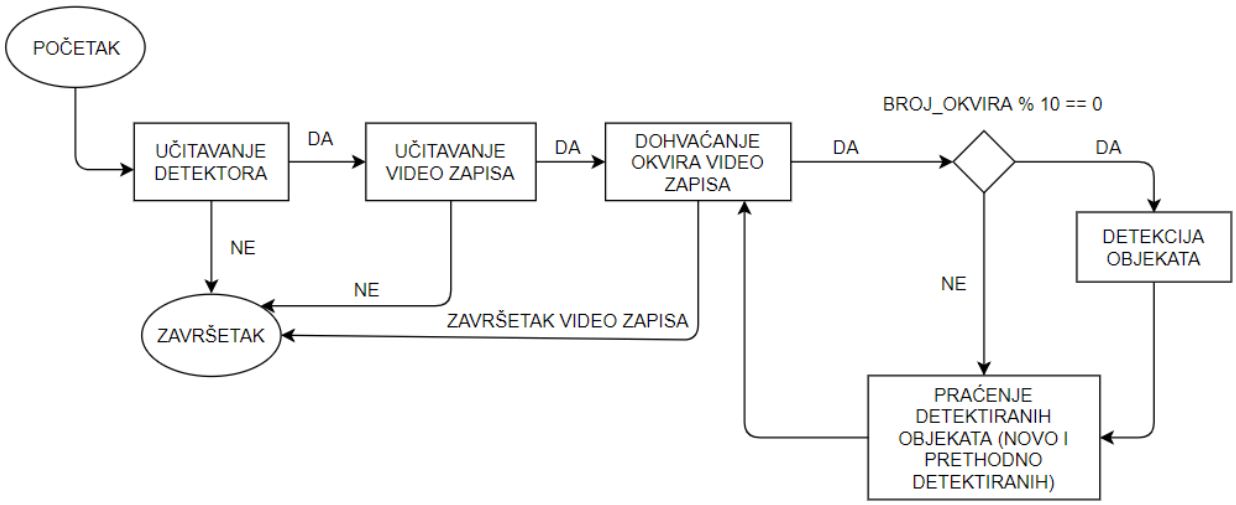
Tab. 3.2. *Informacije o procesu učenja za svaki od izrađenih detektora.*

Naziv detektora	**Broj pozitivnih slika**	**Broj negativnih slika**	**Broj kaskada**	**Dimenzija izlaznog uzorka [piksela]**	**Vrijeme trajanja učenja [h]**
Detektor prednje i stražnje strane automobila	1700	4300	24	24 x 24	24
Detektora bočne strane automobila	1200	4300	24	30 x 15	24
Detektor prednje i stražnje strane autobusa i kamiona	600	4300	24	24 x 32	22
Detektor prometnih znakova opasnosti	2000	3000	15	24 x 24	16
Detektor prometnih znakova izričitih naredbi koje karakterizira plava pozadina	1000	4300	17	24 x 24	16
Detektor prometnih znakova izričitih naredbi koje karakterizira crveni rub	2000	3000	19	24 x 24	20
Detektor prometnog znaka obveznog zaustavljanja	580	4300	10	24 x 24	12
Detektor prometnog znaka raskrižja s cestom koja ima prednost prolaska	1500	3000	19	24 x 24	20
Detektor pješaka	1488	3761	24	12 x 36	24
Detektor semafora	1000	4300	26	5 x 15	36

3.4. Izrada algoritma za detekciju i praćenje objekata u video zapisu u Python programskom jeziku

U ovom potpoglavlju detaljno je opisana izrada računalnog algoritma za detekciju i praćenje objekata u video zapisu u Python programskom jeziku. Za izradu računalnog algoritma korišteni su prethodno izrađeni detektori te OpenCV i dlib biblioteke detaljnije opisane u potpoglavlju 3.1.

Računalni algoritam izrađen u Python programskom jeziku podijeljen je na tri modula, a to su: glavni modul (*main.py*), modul koji obavlja detekciju i praćenje detektiranih objekata (*detection.py*) i modul koji sadrži konstante (*constants.py*). Algoritam radi tako da detektira objekte na svakom desetom okviru video zapisa, a na ostalim okvirima prati detektirane objekte. Detekcija objekata izvršena je na svakom desetom okviru video zapisa jer je pretpostavljeno kako je potrebno obraditi barem deset slijednih okvira video zapisa kako bi se pojavio novi objekt u video zapisu. Za detekciju objekata korišteni su izrađeni detektori i OpenCV biblioteka, a za praćenje detektiranih objekata korištena je dlib biblioteka. Na slici 3.1. prikazan je dijagram toka računalnog algoritma za detekciju i praćenje objekata u video zapisu.



Sl. 3.1. *Dijagram toka računalnog algoritma za detekciju i praćenje objekata u video zapisu*

Svaki detektirani objekt spremljen je u rječnik (engl. *dictionary*) koji sadrži identifikacijski broj objekta i objekt klase *dlib.correlation_tracker*. Za svaki okvir video zapisa rječnik je osvježen te je za svaki praćeni objekt izračunat PSLR (engl. *Peak-to-side Lobe Ratio*). Što je vrijednost PSLR-a veća to je veća vjerojatnost da je unutar trenutnog okvira video zapisa i dalje traženi objekt. Ako je vrijednost PSLR manja od zadane vrijednosti, tada prestaje praćenje objekta.

Modul pod nazivom *constants.py* sadrži konstante. Svaki izrađeni detektor sadrži vlastiti identifikacijski broj (npr. detektor prednje i stražnje strane automobila - 0, detektor bočne strane automobila – 1, itd.). Također, modul sadrži putanje i nazive svih detektora spremljene u dvodimenzionalno polje. Osim toga, sadrži i zadane minimalne vrijednosti PSLR-a pomoću kojih je odlučeno je li unutar trenutnog okvira video zapisa i dalje traženi objekt. Minimalna vrijednost PSLR-a za prednju i stražnju stranu automobila postavljena je na 6,5, za semafore na 8, za bočnu stranu automobila na 8, za pješake na 6, za prednju i stražnju stranu autobusa i kamiona na 8, za prometni znak obveznog zaustavljanja na 9, za prometni znak raskrižja s cestom koja ima prednost prolaska na 8, za prometne znakove opasnosti na 6, za prometne znakove izričitih naredbi koje karakterizira crveni rub na 5, a za prometne znakove izričitih naredbi koje karakterizira plava pozadina na 9. Sadrži i parametre *scaleFactor* i *minNeighbors* korištene prilikom detekcije objekata. U tablici 3.3. prikazane su korištene vrijednosti parametara *scaleFactor* i *minNeighbors* za različite detektore. Parametar *scaleFactor* predstavlja koliko je ulazna slika smanjena svaki puta kada prozor za detekciju prođe kroz cijelu sliku, a *minNeighbors* određuje koliki je minimalan broj susjednih pravokutnika kandidata koji pravokutnik kandidat mora imati kako bi bio označen kao traženi objekt. Ako je parametar *scaleFactor* postavljen na veću vrijednost, tada je veća brzina detekcije, ali je također smanjena točnost detekcije. Vrijednosti navedenih parametara odabrane su empirijskim putem.

Tab. 3.3. *Korištene vrijednosti parametara scaleFactor i minNeighbors za različite detektore.*

Naziv detektora	**scaleFactor**	**minNeighbors**
Detektor prednje i stražnje strane automobila	1,03	5
Detektora bočne strane automobila	1,04	7
Detektor prednje i stražnje strane autobusa i kamiona	1,03	8
Detektor prometnih znakova opasnosti	1,023	3
Detektor prometnih znakova izričitih naredbi koje karakterizira plava pozadina	1,01	5
Detektor prometnih znakova izričitih naredbi koje karakterizira crveni rub	1,015	7
Detektor prometnog znaka obveznog zaustavljanja	1,03	2
Detektor prometnog znaka raskrižja s cestom koja ima prednost prolaska	1,05	9
Detektor pješaka	1,03	6
Detektor semafora	1,03	2

Prethodno navedeni modul sadrži i razlučivost prikazane slike koja iznosi 640 x 480 piksela. Osim toga, prilikom detekcije različitih objekata, različiti objekti označeni su različitim bojama te zbog toga navedeni modul sadrži i boje kojima su označeni različiti objekata.

Glavni modul sadrži glavnu funkciju (*main*) koja je prva pozvana prilikom pokretanja aplikacije. Unutar glavne funkcije učitani su izrađeni detektori. Za učitavanje izrađenih detektora izrađeni su objekti pomoću konstruktora klase *CascadeClassifier* koji koristi jedan parametar, a to je *filename* (naziv XML datoteke koja predstavlja izrađeni detektor). Nakon toga, učitan je video zapis. Za učitavanje video zapisa izrađen je objekt pomoću konstruktora klase *VideoCapture* koji koristi jedan parametar, a to je *filename* (naziv video zapisa). Naziv video zapisa unesen je kao drugi argument komandne linije te je spremljen u odgovarajuću varijablu. Nakon toga, inicijalizirana je varijabla *frameCounter* na vrijednost 0. Navedena varijabla predstavlja trenutni broj okvira video zapisa. Potom je izrađena petlja koja izvršava dio programskog koda do završetka video zapisa. Unutar tog dijela programskog koda dohvaća se svaki okvir video zapisa korištenjem funkcije *read* klase *VideoCapture*. Svakom dohvaćenom okviru video zapisa promijenjena je razlučivost korištenjem funkcije *resize* koja prima dva parametra, a to su: *src* (ulazna slika) i *dsize* (razlučivost na koju je promijenjena ulazna slika). Korištena razlučivost navedena je u modulu *constants.py*, a iznosi 640 x 480 piksela. Potom je povećana vrijednost varijable *frameCounter* za 1. Osim toga, deset puta pozvana je funkcija *trackObjects* definirana u modulu *detection.py*. Funkcija je pozvana navedeni broj puta zato što je izrađeno deset detektora te svaki detektor koristi različite parametre spremljene u različitim varijablama. Nakon toga, prikazan je okvir video zapisa s označenim detektiranim objektima. Sve navedene klase i funkcije pripadaju OpenCV biblioteci.

Modul pod nazivom *detection.py* sadrži funkciju *trackObjects* čija je glavna zadaća detektirati tražene objekte te ih pratiti. Praćenje detektiranih objekata izvršeno je korištenjem klase *dlib.correlation_tracker* iz dlib biblioteke. Funkcija *trackObjects* prima osam parametara, a to su: *frameCounter* (broj trenutnog okvira video zapisa), *image* (ulazna slika), *resultImage* (rezultantna slika), *objectCascade* (objekt klase *CascadeClassifier* izrađen u glavnom modulu), *currentObjectID* (trenutni broj određenog tipa objekta), *objectIDtoDelete* (polje koje sadrži identifikacijske brojeve objekata koje više ne treba pratiti), *objectTracker* (rječnik koji sadrži identifikacijske brojeve objekata i objekte klase *dlib.correlation_tracker*) i *objectID* (identifikacijski broj detektora). Prvo je izračunata vrijednost PSLR-a za svaki praćeni objekt korištenjem funkcije *update* klase *dlib.correlation_tracker*. Funkcija prima jedan parametar, a to je *image* (ulazna slika, odnosno trenutni okvir video zapisa). Nakon toga, vrijednost PSLR-a praćenog objekta uspoređena je s minimalnom vrijednosti PSLR-a za taj objekt. Minimalne

vrijednosti su prethodno navedene. Ako je izračunata vrijednost PSLR-a praćenog objekta manja od minimalne vrijednosti, tada je identifikacijski broj objekta dodan u polje *objectIDtoDelete* koje sadrži identifikacijske brojeve objekata koje više ne treba pratiti. Potom su izbrisani svi objekti iz rječnika čiji su identifikacijski brojevi navedeni u polju *objectIDtoDelete*. Nakon toga, navedena je *if* grana koja je izvršena ako je broj trenutnog okvira video zapisa djeljiv brojem deset. Unutar *if* grane ulazna slika pretvorena je u sliku u nijansama sive boje. To je ostvareno korištenjem funkcije *cvtColor* iz OpenCV biblioteke koja prima dva parametra, a to su: *src* (ulazna slika) i *code* (model boja u koji je pretvorena ulazna slika). Nakon toga, pokrenut je proces detekcije objekata korištenjem funkcije *detectMultiScale* klase *CascadeClassifier*. Funkcija prima šest parametara, a to su: *image* (slika u nijansama sive boje), *scaleFactor*, *minNeighbors*, *flags*, *minSize* (minimalna dimenzija detektiranog objekta) i *maxSize* (maksimalna dimenzija detektiranog objekta). Svi potrebni parametri funkcije *detectMultiScale* za svaki detektor su konstante te su navedeni u modulu *constants.py*. Funkcija kao povratnu vrijednost vraća lokacije detektiranih objekata na okviru video zapisa. Nakon toga, uspoređene su lokacije novo detektiranih objekata s lokacijama prethodno praćenih objekata. Ako je lokacija novo detektiranog objekta identična ili je u neposrednoj blizini prethodno praćenog objekta, odnosno granice su im preklopljene, tada prestaje praćenje prethodno detektiranog objekta jer je pretpostavljeno da je identičan novo detektiranom objektu. Potom je izrađen objekt klase *dlib.correlation_tracker* i počinje praćenje novo detektiranog objekta korištenjem funkcije *start_track* prethodno navedene klase. Funkcija prima dva parametra, a to su: *image* (ulazna slika) i *rectangle* (lokacija novo detektiranog objekta). Potom je u rječnik upisan identifikacijski broj objekta i objekt klase *dlib.correlation_tracker* koji je prethodno izrađen te je povećana vrijednost varijable *currentObjectID* za 1. Nakon toga, završeno je izvođenje prethodno navedene *if* grane te je svakom objektu upisanom u rječnik osvježena lokacija. To je izvršeno korištenjem funkcije *get_position* klase *dlib.correlation_tracker*. Potom su označeni traženi objekti na trenutnom okviru video zapisa. Traženi objekti označeni su pravokutnikom. To je učinjeno korištenjem funkcije *rectangle* iz OpenCV biblioteke koja prima pet parametara, a to su: *image* (ulazna slika), *pt1* (lokacija gornje lijeve točke), *pt2* (lokacija donje desne točke), *color* (boja stranica pravokutnika) i *thickness* (debljina stranica pravokutnika). Osim toga, ispisane su lokacije i dimenzije traženih objekata u komandnom sučelju. Funkcija *trackObjects* vraća tri povratne vrijednosti, a to su: cijeli broj *currentObjectID*, polje *objectIDtoDelete* i rječnik *objectTracker*.

3.5. Izrada algoritma za detekciju i praćenje objekata u video zapisu u C++ programskom jeziku

U ovom potpoglavlju detaljno je opisana izrada računalnog algoritma za detekciju i praćenje objekata u video zapisu u C++ programskom jeziku. Za razliku od Python programskog jezika, u C++ programskom jeziku treba obratiti veću pozornost na tipove podataka, odnosno treba navesti tip podatka svake varijable. Računalni algoritam izrađen je u C++ programskom jeziku radi poboljšanja performansi, a osim toga lakše ga je implementirati na računalnu ugradbenu platformu. Samim time, takvo rješenje je bliže stvarnoj primjeni. Dijagram toka računalnog algoritma izrađenog u Python programskom jeziku isti je kao i u C++ programskom jeziku (Sl. 3.1.). Za razliku od algoritma izrađenog u Python programskom jeziku, algoritam izrađen u C++ programskom jeziku koristi višenitnost, odnosno obavlja više operacija istovremeno. Osim toga, nije korištena dlib biblioteka. Zbog toga je korištena OpenCV biblioteka za praćenje detektiranih objekata.

Unutar OpenCV biblioteke postoje različite metode praćenja detektiranih objekata. Neke od tih metoda su: Meanshift algoritam, Camshift algoritam, Median Flow metoda za praćenje objekata, itd.

Meanshift algoritam služi za praćenje objekata. Potrebno je pretpostaviti da postoji skup točaka i područje interesa. Područje interesa pomaknuto je do područja s maksimalnim brojem točaka, odnosno maksimalnom gustoćom piksela. Pomicanje se izvršava iterativno. Algoritam radi tako da upotrebljava histogram unatrag projicirane slike i početnu lokaciju i dimenziju objekta. Prilikom pomicanja objekta, pokret je reflektiran na histogram unatrag projicirane slike. Kao rezultat, početna lokacija objekta pomaknuta je na novu lokaciju, a to je lokacija s maksimalnim brojem točaka, odnosno maksimalnom gustoćom piksela. Unutar OpenCV biblioteke postoji funkcija *meanShift* koja koristi Meanshift algoritam za praćenje objekata. Funkcija radi tako da pronalazi objekt na unatrag projiciranoj slici. Prima tri parametra, a to su: *probImage* (unatrag projicirana slika traženog objekta), *window* (početna lokacija i dimenzija traženog objekta) i *criteria* (kriterij koji treba ispuniti). Testiranjem navedenog algoritma zaključeno je kako je dimenzija objekta uvijek identična njegovoj početnoj dimenziji bez obzira na promjenu lokacije objekta. Bez obzira na približavanje ili udaljavanje traženog objekta, njegova dimenzija je identična njegovoj početnoj dimenziji. Također, ako je objekt izvan slike odnosno okvira video zapisa, njegovo praćenje neće prestati. Ako je promijenjeno osvjetljenje između slijednih okvira video zapisa, tada najčešće dolazi do pogrešnog praćenja traženih objekata. Zaključeno je kako

navedeni algoritam nije dovoljno pouzdan za praćenje objekata detektiranih korištenjem izrađenih detektora.

Kao i Meanshift algoritam, tako i Camshift (engl. *Continuously Adaptive Meanshift*) algoritam služi za praćenje objekata, ali uz određene modifikacije. Prvi korak je primjena Meanshift algoritma. Nakon toga, osvježena je dimenzija područja interesa. Također, određena je najbolje odgovarajuća elipsa za područje interesa. Potom je ponovno primijenjen MeanShift algoritam s novom dimenzijom i prethodnom lokacijom područja interesa. Proces je nastavljen dok nije ispunjena određena točnost. Unutar OpenCV biblioteke postoji funkcija *CamShift* koja koristi Camshift algoritam za praćenje objekata. Funkcija radi tako da pronalazi središte i dimenziju objekta. Prima tri identična parametra kao i *meanShift* funkcija. Testiranjem navedenog algoritma zaključeno je kako je promijenjena dimenzija objekta ovisno o njegovu približavanju ili udaljavanju. Osim toga, prisutni su isti problemi prilikom praćenja objekata kao i kod Meanshift algoritma te je zaključeno kako navedeni algoritam nije dovoljno pouzdan za praćenje objekata detektiranih korištenjem izrađenih detektora.

Osim prethodno navedenih algoritama za praćenje objekata, unutar OpenCV biblioteke postoji i Median Flow metoda za praćenje objekata. Prema [29], Median Flow metoda za praćenje objekata je temeljena na FB (engl. *Forward Backward*) pogrešci. Smanjivanjem FB pogreške omogućeno je pouzdano otkrivanje pogrešaka prilikom praćenja objekata i odabir kretanja praćenog objekta u slijednim okvirima video zapisa. Prema [30], Median Flow metoda izvještava kada praćeni objekt izađe izvan okvira video zapisa i točno prati tražene objekte ako se ne pomiču prebrzo između slijednih okvira video zapisa. Prema [30], nedostatak Median Flow metode je pogrešno praćenje objekta prilikom brzog kretanja između slijednih okvira video zapisa.

Između prethodno navedenih metoda praćenja objekata dostupnih unutar OpenCV biblioteke odabrana je Median Flow metoda zbog prethodno navedenih objašnjenja.

Algoritam za detekciju i praćenje objekata u C++ programskom jeziku sastavljen je od glavnog modula (*main.cpp*) i modula za detekciju i praćenje detektiranih objekata. Modul za detekciju i praćenje detektiranih objekata sastavljen je od datoteke zaglavlja (.h datoteke) i .cpp datoteke.

Kao i u Python programskom jeziku, tako i u C++ programskom jeziku glavni modul sadrži glavnu funkciju (*main*) koja je prva pozvana prilikom pokretanja aplikacije. Na početku su deklarirane sve potrebne varijable, a opisane su samo najvažnije. Deklarirano je polje *thread* koje sadrži deset niti tipa *pthread_t*. Potom je deklarirano polje *arguments* koje sadrži deset struktura *functionArguments*. Struktura *functionArguments* sastavljena je od 12 članova, a to su:

- *uint32_t ID* - identifikacijski broj detektora
- *int minNeighbors* - parametar korišten prilikom detekcije objekata detaljnije objašnjen u poglavlju 3.4.
- *double scale* - parametar korišten prilikom detekcije objekata detaljnije objašnjen u poglavlju 3.4.
- *uint32_t frameCounter* - trenutni broj okvira video zapisa
- *uint32_t* currentObjectID* - pokazivač koji pokazuje na varijablu tipa *uint32_t* koja predstavlja trenutni broj objekta određenog detektora
- *char const* objectType* - pokazivač koji pokazuje na varijablu tipa *char const* koja predstavlja naziv tipa objekta
- *int* objectColor* - pokazivač koji pokazuje na varijablu tipa *int* koja predstavlja boju kojom je označen određeni objekt
- *vector<uint32_t> *objectIDtoDelete* - pokazivač koji pokazuje na dinamičko polje koje sadrži elemente tipa *uint32_t* koji predstavljaju objekte koje treba prestati pratiti
- *map<uint32_t, Ptr<Tracker> > *objectTracker* - pokazivač koji pokazuje na rječnik koji sadrži identifikacijske brojeve objekata i pokazivače koji pokazuju na objekt klase *Tracker*
- *CascadeClassifier objectCascade* - objekt klase *CascadeClassifier*
- *Mat img* - ulazna slika
- *Mat* resultImage* - pokazivač koji pokazuje na objekt klase *Mat* koji predstavlja sliku na kojoj su označeni detektirani objekti.

Svaki detektor posjeduje vlastitu strukturu. Unutar glavne funkcije pozvana je funkcija *loadCascadeClassifiers* koja je implementirana u modulu za detekciju i praćenje detektiranih objekata. Nakon toga, inicijalizirane su sve strukture unutar polja *arguments*. Potom je pozvana funkcija *videoOpen* koja je također implementirana u prethodno navedenom modulu. Nakon toga, preostali dio programskog koda naveden je unutar *while* petlje čiji je uvjet izvršavanja postojanje okvira video zapisa. Unutar navedene *while* petlje promijenjena je razlučivost ulazne slike korištenjem funkcije *resize* iz OpenCV biblioteke koja prima tri parametra, a to su: *src* (ulazna slika), *dst* (izlazna slika) i *dsize* (razlučivost na koju je promijenjena ulazna slika). Razlučivost na koju je promijenjena ulazna slika iznosi 640 x 480 piksela, isto kao i prilikom izrade algoritma u Python programskom jeziku. Nakon toga, povećana je vrijednost varijable *frameCounter* za 1. Potom je izrađeno deset niti koje izvršavaju funkciju *trackObjects* istovremeno. Svaka nit izvršava navedenu funkciju korištenjem različitih parametara jer svaki detektor sadrži vlastite parametre, a

oni su navedeni unutar struktura *functionArguments*. Funkcija *trackObjects* implementirana je u modulu za detekciju i praćenje detektiranih objekata. Također, sve niti su sinkronizirane. To znači da sve niti trebaju završiti s radom na trenutnom okviru video zapisa, a tek nakon toga prikazan je okvir video zapisa s označenim detektiranim objektima i učitan je sljedeći okvir video zapisa.

Datoteka zaglavlja modula za detekciju i praćenje detektiranih objekata ekvivalentna je modulu *constants.py* korištenom prilikom izrade algoritma u Python programskom jeziku. Modul *constants.py* detaljno je opisan u potpoglavlju 3.4. U prethodno navedenoj datoteci zaglavlja definirane su vrijednosti svih potrebnih konstanti te deklaracije funkcija. Unutar .cpp datoteke prethodno navedenog modula definirane su tri funkcije, a to su: *videoOpen*, *loadCascadeClassifiers* i *trackObjects*. Funkcija *videoOpen* služi za učitavanje video zapisa i prima dva parametra, a to su: pokazivač *capture* koji pokazuje na objekt klase *VideoCapture* koji predstavlja video zapis i pokazivač *videoPath* na pokazivač koji pokazuje na varijablu tipa *char* koja predstavlja putanju i naziv video zapisa. Unutar funkcije korištena je funkcija *open* klase *VideoCapture* za učitavanje video zapisa. Navedena funkcija prima jedan parametar, a to je *filename* (naziv video zapisa). Funkcija *loadCascadeClassifiers* služi za učitavanje detektora i prima dva parametra, a to su: pokazivač *objectCascadeClassifier* koji pokazuje na objekt klase *CascadeClassifier* i pokazivač *objectPath* koji pokazuje na varijablu tipa *char* koja predstavlja putanju i naziv XML datoteke, odnosno detektora. Unutar navedene funkcije učitan je detektor korištenjem funkcije *load* klase *CascadeClassifier* koja prima jedan parametar, a to je *filename* (naziv XML datoteke, odnosno detektora). Funkcija *trackObjects* služi za detekciju traženih objekata i za praćenje detektiranih objekata. Navedena funkcija prima jedan parametar, a to je pokazivač *arguments* na varijablu tipa *void*. Unutar funkcije izvršena je pretvorba ulaznog tipa podatka u pokazivač na strukturu *functionArguments*. Nakon toga, provjeren je uvjet je li trenutni okvir video zapisa djeljiv brojem 50. Ako je djeljiv, tada su svi identifikacijski brojevi objekata dodani u polje *objectIDtoDelete* te su svi objekti čiji su identifikacijski brojevi navedeni u polju izbrisani iz rječnika. Tako je izvršeno poništenje praćenja objekata svakih 50 okvira video zapisa. Kao i u algoritmu izrađenom u Python programskom jeziku, tako i u ovome algoritmu detekcija objekata obavljena je na svakom desetom okviru video zapisa. Navedena je *if* grana koja je izvršena u slučaju da je broj trenutnog okvira video zapisa djeljiv brojem deset. Unutar *if* grane ulazna slika pretvorena je u sliku u nijansama sive boje korištenjem funkcije *cvtColor* koja prima tri parametra, a to su: *src* (ulazna slika), *dst* (izlazna slika) i *code* (model boja u koji je pretvorena ulazna slika). Nakon toga, detektirani su objekti na trenutnom okviru video zapisa korištenjem funkcije *detectMultiScale* klase *CascadeClassifier* koja prima sedam parametara, a to su: *image* (slika u nijansama sive boje), *objects* (dinamičko polje u kojemu su spremljene lokacije objekata),

scaleFactor, *minNeighbors*, *flags*, *minSize* (minimalna dimenzija detektiranog objekta) i *maxSize* (maksimalna dimenzija detektiranog objekta). Potom su osvježene lokacije prethodno praćenih objekata korištenjem funkcije *update* klase *Tracker*. Funkcija prima dva parametra, a to su: *image* (trenutni okvir video zapisa) i *boundingBox* (područje interesa, odnosno lokacija i dimenzija praćenog objekta). Ako je lokacija novo detektiranog objekta unutar lokacije prethodno detektiranog objekta ili obratno, tada prestaje praćenje prethodno detektiranog objekta jer je pretpostavljeno da je novo detektirani objekt identičan prethodno praćenom objektu. Također, isto je izvršeno ako su im granice preklopljene. Nakon toga, svi novo detektirani objekti su praćeni. Za praćenje detektiranih objekata korištena je Median Flow metoda koja je prethodno objašnjena. Izrađen je objekt klase *Tracker* korištenjem funkcije *create*. Funkcija prima jedan parametar, a to je *trackerType* (naziv metode koja je korištena za praćenje objekata). Nakon toga, prethodno navedeni objekt je inicijaliziran korištenjem funkcije *init* klase *Tracker* koja prima dva parametra, a to su: *image* (trenutni okvir video zapisa) i *boundingBox* (područje interesa, odnosno lokacija i dimenzija objekta kojega treba pratiti). Potom je u rječnik *objectTracker* upisan identifikacijski broj objekta i pokazivač koji pokazuje na objekt klase *Tracker* koji je prethodno izrađen te je povećana vrijednost varijable *currentObjectID* za 1. Također, ako je objekt u neposrednoj blizini ruba okvira video zapisa, tada je pretpostavljeno da objekt izlazi izvan okvira video zapisa i više ga ne treba pratiti. Posljednji korak je označavanje detektiranih objekata pravokutnikom i ispis lokacija i dimenzija detektiranih objekata. Za označavanje detektiranih objekata korištena je funkcija *rectangle*. Funkcija prima iste parametre kao i istoimena funkcija korištena prilikom izrade algoritma u Python programskom jeziku.

Za prevođenje programskog rješenja izrađena je .mk datoteka (engl. *Makefile*) koja je sastavljena od skupa naredbi koje prevode programski kod napisan u C++ programskom jeziku i izrađuju izvršnu datoteku. Za pokretanje .mk datoteke izvršena je naredba *make* unutar komandnog sučelja. Nakon toga, izrađena je izvršna datoteka (.out datoteka). Izvršna datoteka je pokrenuta unutar komandnog sučelja korištenjem naredbe *naziv_izvršne_datoteke naziv_video_zapisa*.

4. EVALUACIJA IZRAĐENOG ALGORITMA ZA DETEKCIJU I PRAĆENJE OBJEKATA U VIDEO ZAPISU

U ovom poglavlju prikazana je evaluacija izrađenog algoritma za detekciju i praćenje objekata u video zapisu. U potpoglavlju 4.1. prikazani su rezultati testiranja izrađenih detektora na testnom skupu slika. U potpoglavlju 4.2. prikazana je evaluacija izrađenog algoritma za detekciju i praćenje objekata u video zapisu u Python programskom jeziku. U potpoglavlju 4.3. prikazana je evaluacija izrađenog algoritma za detekciju i praćenje objekata u video zapisu u C++ programskom jeziku.

4.1. Testiranje izrađenih detektora

U ovom potpoglavlju prikazani su rezultati testiranja izrađenih detektora. Za testiranje izrađenih detektora korišten je skup slika različit od skupa slika korištenog u procesu učenja. Također je moguće koristiti dostupne ili vlastite skupove slike. Za testiranje izrađenih detektora korišteni su vlastiti skup slika i dijelovi Cars [22], Udacity [23], German Traffic Sign Recognition Benchmark [24] i INRIA Person [25] skupova slika. Također, korištene su pozitivne i negativne slike, odnosno slike na kojima je prikazan traženi objekt i slike na kojima nije prikazan traženi objekt. Rezultati testiranja prikazani su tablicama te su na temelju njih izračunate dvije mjere, a to su preciznost i odziv. Za izračun preciznosti i odziva, kao i ostalih mjera u strojnom učenju, treba koristiti broj točnih pozitivnih (engl. *True Positive - TP*) detekcija, lažnih negativnih (engl. *False Negative - FN*) detekcija, lažnih pozitivnih (engl. *False Positive - FP*) detekcija i točnih negativnih (engl. *True Negative - TN*) detekcija. Ako je na slici prikazan automobil i treba ga detektirati te je isti detektiran, tada to predstavlja točnu pozitivnu detekciju. Ako je na slici prikazan automobil i treba ga detektirati, a nije detektiran, tada to predstavlja lažnu negativnu detekciju. Ako je na slici na kojoj nije prikazan automobil detektiran isti, tada to predstavlja lažnu pozitivnu detekciju. Ako na slici na kojoj nije prikazan automobil nije detektiran isti, tada to predstavlja točnu negativnu detekciju. S obzirom na to da je svaki dio slike na kojemu nije prikazan objekt te isti nije detektiran smatran točnom negativnom detekcijom, nije potrebno koristiti točne negativne detekcije u evaluaciji rješenja. U tablici 4.1. prikazani su rezultati testiranja izrađenih detektora. U tablici 4.1. vidljivo je kako detektor bočne strane automobila nema nijednu FP detekciju na testnom skupu slika. Osim toga, detektor prometnog znaka raskrižja s cestom koja ima prednost prolaska ima najmanji broj FN detekcija na testnom skupu slika. Navedeni rezultati ovise o broju testnih slika i o broju traženih objekata na tim testnim slikama.

Tab 4.1. *Rezultati testiranja izrađenih detektora.*

Naziv detektora	Broj testnih slika	Broj traženih objekata	Broj TP detekcija	Broj FN detekcija	Broj FP detekcija
Detektor prednje i stražnje strane automobila	423	258	233	25	13
Detektor bočne strane automobila	196	81	68	13	0
Detektor prednje i stražnje strane autobusa i kamiona	257	76	61	15	14
Detektor prometnih znakova opasnosti	267	91	85	6	14
Detektor prometnih znakova izričitih naredbi koje karakterizira plava pozadina	234	117	112	5	14
Detektor prometnih znakova izričitih naredbi koje karakterizira crveni rub	288	119	109	10	12
Detektor prometnog znaka obveznog zaustavljanja	230	77	67	10	2
Detektor prometnog znaka raskrižja s cestom koja ima prednost prolaska	313	89	87	2	4
Detektor pješaka	300	100	78	22	16
Detektor semafora	100	240	191	49	9

Na slici 4.1. prikazan je primjer točne pozitivne detekcije detektora prednje i stražnje strane automobila.

Sl. 4.1. *Primjer točne pozitivne detekcije detektora prednje i stražnje strane automobila*

Na slici 4.2. prikazan je primjer točne pozitivne detekcije detektora bočne strane automobila.

Sl. 4.2. *Primjer točne pozitivne detekcije detektora bočne strane automobila*

Na slici 4.3. prikazan je primjer točne pozitivne detekcije detektora prednje i stražnje strane autobusa i kamiona.

Sl. 4.3. *Primjer točne pozitivne detekcije detektora prednje i stražnje stranu autobusa i kamiona*

Na slici 4.4. prikazan je primjer točne pozitivne detekcije detektora semafora.

Sl. 4.4. *Primjer točne pozitivne detekcije detektora semafora*

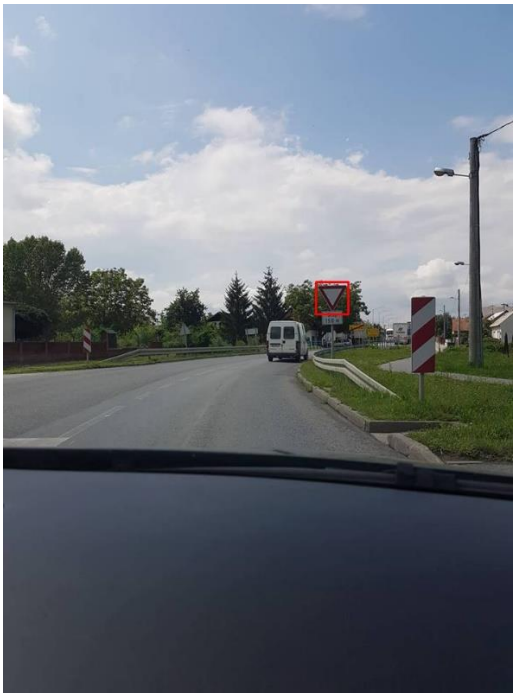
Na slici 4.5. prikazan je primjer točne pozitivne detekcije detektora pješaka.

Sl. 4.5. *Primjer točne pozitivne detekcije detektora pješaka*

Na slici 4.6. prikazan je primjer točne pozitivne detekcije detektora prometnog znaka obveznog zaustavljanja.

Sl. 4.6. *Primjer točne pozitivne detekcije detektora prometnog znaka obveznog zaustavljanja*

Na slici 4.7. prikazan je primjer točne pozitivne detekcije detektora prometnog znaka raskrižja s cestom koja ima prednost prolaska.



Sl. 4.7. *Primjer točne pozitivne detekcije detektora prometnog znaka raskrižja s cestom koja ima prednost prolaska*

Na slici 4.8. prikazan je primjer točne pozitivne detekcije detektora prometnih znakova opasnosti.

Sl. 4.8. *Primjer točne pozitivne detekcije detektora prometnih znakova opasnosti*

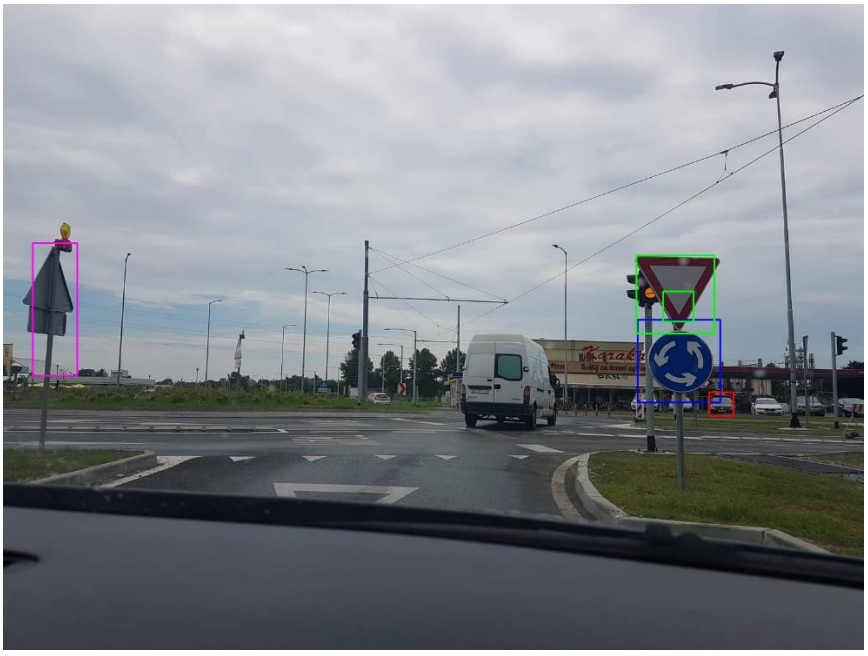
Na slici 4.9. prikazan je primjer točne pozitivne detekcije detektora prometnih znakova izričitih naredbi koje karakterizira crveni rub.

Sl. 4.9. *Primjer točne detekcije detektora prometnih znakova izričitih naredbi koje karakterizira crveni rub*

Na slici 4.10. prikazan je primjer točne pozitivne detekcije detektora prometnih znakova izričitih naredbi koje karakterizira plava pozadina.

Sl. 4.10. *Primjer točne pozitivne detekcije detektora prometnih znakova izričitih naredbi koje karakterizira plava pozadina*

Na slici 4.11. prikazan je primjer rada svih detektora na jednoj slici.



Sl. 4.11. *Primjer rada svih detektora na slici*

Na slici 4.11. plavom bojom su označene detekcije detektora prometnih znakova izričitih naredbi koje karakterizira plava pozadina, crvenom bojom detekcije detektora prednje i stražnje strane automobila, rozom bojom detekcije detektora pješaka, a zelenom bojom detekcije detektora prometnog znaka raskrižja s cestom koja ima prednost prolaska. Na slici 4.11. prikazana je prednja i stražnja strana automobila, a točno je detektirana samo prednja strana automobila te ona predstavlja točnu pozitivnu detekciju detektora prednje i stražnje strane automobila. Stražnja strana automobila koja nije detektirana predstavlja lažnu negativnu detekciju detektora prednje i stražnje strane automobila. Stražnja strana automobila nije detektirana zbog nagnutosti i omjera širine i duljine navedenog objekta. Na slici 4.11. prikazan je i prometni znak raskrižja s cestom koja ima prednost prolaska te je vidljivo kako je točno označen navedeni prometni znak, no označen je i dio navedenog prometnog znaka te on predstavlja lažnu pozitivnu detekciju detektora prometnog znaka raskrižja s cestom koja ima prednost prolaska. Na slici 4.11. detektiran je pješak i označen je pravokutnikom roze boje, no on se ne nalazi na navedenoj slici te zbog toga navedena detekcija predstavlja lažnu pozitivnu detekciju detektora pješaka. Objekt je označen kao pješak zbog sličnog oblika. Osim toga, na slici 4.11. točno je detektiran prometni znak izričitih naredbi kojeg karakterizira plava pozadina te zbog toga navedena detekcija predstavlja točnu pozitivnu detekciju detektora prometnih znakova izričitih naredbi koje karakterizira plava pozadina.

Na temelju rezultata iz tablice 4.1., za svaki detektor izračunate su dvije mjere, a to su preciznost i odziv. Preciznost je definirana kao omjer broja točnih pozitivnih detekcija i ukupnog broja detekcija u skupu slika te je izračunata korištenjem izraza (4-1). Prema [31], izraz za izračun preciznosti je sljedeći:

$$PRECIZNOST = \frac{TP}{TP+FP} \tag{4-1}$$

gdje je:

- *TP* - broj točnih pozitivnih detekcija,
- *FP* - broj lažnih pozitivnih detekcija.

Zbroj točnih pozitivnih detekcija i lažnih negativnih detekcija predstavlja ukupan broj detekcija u skupu slika.

Odziv je definiran kao omjer broja točnih pozitivnih detekcija i ukupnog broja traženih objekata u skupu slika te je izračunat korištenjem izraza (4-2). Prema [31], izraz za izračun odziva je sljedeći:

$$ODZIV = \frac{TP}{TP+FN} \tag{4-2}$$

gdje je:

- *TP* - broj točnih pozitivnih detekcija,
- *FN* - broj lažnih negativnih detekcija.

Zbroj točnih pozitivnih detekcija i lažnih negativnih detekcija predstavlja ukupan broj traženih objekata u skupu slika.

Ako je niska preciznost, a visok odziv, tada je većina traženih objekata točno detektirana, ali je također i veliki broj lažnih pozitivnih detekcija. Sa stajališta autonomne vožnje to znači kako bi autonomno vozilo detektiralo objekte tamo gdje nisu te bi kočilo bez potrebe. Ako je visoka preciznost, a nizak odziv, tada većina traženih objekata nije detektirana, ali one detekcije koje su pozitivne zaista su točno pozitivne. Sa stajališta autonomne vožnje to znači kako autonomno vozilo ne bi detektiralo sve potrebne objekte i samim time ih ne bi izbjeglo te bi došlo do sudara. Ako je niska preciznost i nizak odziv, tada većina traženih objekata nije detektirana, a također postoji i veliki broj lažnih pozitivnih detekcija. Ako je visoka preciznost i visok odziv, tada je većina traženih objekata točno detektirana te ne postoji veliki broj lažnih pozitivnih detekcija.

U tablici 4.2. prikazane su izračunate mjere preciznosti i odziva svakog od izrađenih detektora na temelju tablice 4.1. te brzina detekcije izrađenih detektora. Prilikom izračuna brzine detekcije svakog detektora, korištene su slike razlučivosti 640 x 480 piksela te je korišten procesor Intel i5-6200U. Također, korištene su vrijednosti parametara *scaleFactor* i *minNeighbors* navedene u tablici 3.3.

Tab. 4.2. *Preciznost i odziv svakog od izrađenih detektora na temelju tablice 4.1. te brzina detekcije izrađenih detektora.*

Naziv detektora	**Preciznost**	**Odziv**	**Brzina detekcije [broj obrađenih slika u sekundi]**
Detektor prednje i stražnje strane automobila	0,9472	0,9031	3
Detektor bočne strane automobila	1	0,8395	4
Detektor prednje i stražnje strane autobusa i kamiona	0,8133	0,8026	3
Detektor prometnih znakova opasnosti	0,8586	0,9341	6
Detektor prometnih znakova izričitih naredbi koje karakterizira plava pozadina	0,8889	0,9573	2
Detektor prometnih znakova izričitih naredbi koje karakterizira crveni rub	0,9008	0,916	3

Detektor prometnog znaka obveznog zaustavljanja	0,971	0,8701	14
Detektor prometnog znaka raskrižja s cestom koja ima prednost prolaska	0,956	0,9775	18
Detektor pješaka	0,8298	0,78	2
Detektor semafora	0,955	0,7958	3

Prema tablici 4.2., zaključeno je kako detektor bočne strane automobila ima najveću preciznost, a detektor prometnog znaka raskrižja s cestom koja ima prednost prolaska ima najveći odziv i brzinu detekcije. Preciznost i odziv ovise o odabranog testnom skupu slika i vrijednostima parametara *scaleFactor* i *minNeighbors* koji su detaljnije opisani u potpoglavlju 3.4. U slučaju odabira drugačijeg testnog skupa slika ili drugačijih vrijednosti parametara *scaleFactor* i *minNeighbors*, rezultati bi se razlikovali od prikazanih rezultata. Brzina detekcije objekata ovisi o postavljenim parametrima *scaleFactor* i *minNeighbors* i korištenom procesoru.

4.2. Evaluacija algoritma za detekciju i praćenje objekata izrađenog u Python programskom jeziku

U ovom potpoglavlju prikazana je evaluacija algoritma za detekciju i praćenje objekata izrađenog u Python programskom jeziku. Najveći nedostatak izrađenog računalnog algoritma je spora detekcija objekata. Provedenim testiranjem, korištenjem procesora Intel Core i5 – 6200U, potrebno je 2,5 sekunde za detekciju svih traženih objekata na okviru video zapisa, dok prilikom praćenja detektiranih objekata brzina praćenja ovisi o broju praćenih objekata. Testiranje brzine praćenja detektiranih objekata provedeno je na tri testna video zapisa. Sva tri testna video zapisa snimljena su prilikom gradske vožnje i sunčanog razdoblja dana. Prvi testni video zapis traje 2 minute i 37 sekundi, drugi 1 minutu i 32 sekunde, a treći 1 minutu i 1 sekundu. U tablici 4.3. prikazana je brzina praćenja detektiranih objekata ovisno o broju praćenih objekata u testnim video zapisima.

Tab. 4.3. *Brzina praćenja detektiranih objekata ovisno o broju praćenih objekata u testnim video zapisima.*

Naziv testnog video zapisa	Broj praćenih objekata	Minimalna brzina praćenja [broj slika u sekundi]	Maksimalna brzina praćenja [broj slika u sekundi]	Srednja brzina praćenja [broj slika u sekundi]
Prvi testni video zapis	1	42	70	51
	2	35	42	40
	3	26	28	27
Drugi testni video zapis	1	47	60	53
	2	37	40	38
	3	28	31	29
	4	24	26	25
Treći testni video zapis	1	51	70	55
	2	32	40	35
	3	26	29	28
	4	22	24	23

Prema tablici 4.3. zaključeno je da što je manji broj praćenih objekata, to je brzina praćenja detektiranih objekata veća.

Praćenje detektiranih objekata također ovisi i o postavljenoj minimalnoj vrijednosti PSLR-a koja je detaljnije opisana u potpoglavlju 3.4. U tablici 4.4. prikazan je broj pogrešnih praćenja točno pozitivnih detekcija u testnim video zapisima. Promjenom minimalne vrijednosti PSLR-a rezultati prethodno navedene tablice bi se promijenili.

Tab. 4.4. *Broj pogrešnih praćenja točno pozitivnih detekcija u testnim video zapisima.*

Video zapis	Broj TP detekcija	Broj pogrešnih praćenja TP detekcija
Prvi video zapis	30	9
Drugi video zapis	30	8
Treći video zapis	17	6

Provedenim testiranjem zaključeno je da navedeni računalni algoritam ne obavlja detekciju i praćenje detektiranih objekata u stvarnom vremenu i samim time nije dovoljno pouzdan za

korištenje u autonomnim vozilima. Osim toga, zaključeno je kako je najčešća pogreška praćenja točno pozitivnih detekcija objekata prebrzo kretanje praćenog objekta između slijednih okvira video zapisa. Također, navedeni algoritam nije dovoljno pouzdan za korištenje u autonomnim vozilima zbog prethodno navedenih pogrešaka praćenja detektiranih objekata. Za brže obavljanje procesa detekcije i praćenja detektiranih objekata treba koristiti višenitnost, odnosno obavljanje više operacija istovremeno.

4.3. Evaluacija algoritma za detekciju i praćenje objekata izrađenog u C++ programskom jeziku

U ovom potpoglavlju prikazana je evaluacija algoritma za detekciju i praćenje objekata izrađenog u C++ programskom jeziku. Testiranje algoritma provedeno je korištenjem procesora Intel Core i5-6200 te je potrebno 2,5 sekunde za detekciju svih traženih objekata na okviru video zapisa, a brzina praćenja prethodno detektiranih objekata ovisi o broju praćenih objekata. Testiranje brzine praćenja detektiranih objekata provedeno je na identična tri testna video zapisa kao i u prethodnom potpoglavlju. U tablici 4.5. prikazana je brzina praćenja detektiranih objekata ovisno o broju praćenih objekata u testnim video zapisima.

Tab. 4.5. *Brzina praćenja detektiranih objekata ovisno o broju praćenih objekata u testnim video zapisima.*

Naziv testnog video zapisa	Broj praćenih objekata	Minimalna brzina praćenja [broj slika u sekundi]	Maksimalna brzina praćenja [broj slika u sekundi]	Srednja brzina praćenja [broj slika u sekundi]
Prvi testni video zapis	1	86	102	96
	2	54	87	67
	3	37	54	47
Drugi testni video zapis	1	74	102	95
	2	58	82	65
	3	47	57	50
	4	46	52	47
Treći testni video zapis	1	89	101	98
	2	73	83	77
	3	42	70	55
	4	29	47	36

Prema tablici 4.5. zaključeno je da što je manji broj praćenih objekata, to je brzina praćenja detektiranih objekata veća.

U tablici 4.6. prikazan je broj pogrešnih praćenja točno pozitivnih detekcija u testnim video zapisima.

Tab. 4.6. *Broj pogrešnih praćenja točno pozitivnih detekcija u testnim video zapisima.*

Video zapis	**Broj TP detekcija**	**Broj pogrešnih praćenja TP detekcija**
Prvi video zapis	30	2
Drugi video zapis	30	7
Treći video zapis	17	4

Provedenim testiranjem zaključeno je kako algoritam za detekciju i praćenje objekata izrađen u C++ programskom jeziku obavlja brže praćenje detektiranih objekata od algoritma izrađenog u Python programskom jeziku na korištenim testnim video zapisima. Osim toga, zaključeno je da algoritam za detekciju i praćenje objekata izrađen u C++ programskom jeziku radi manji broj pogrešaka prilikom praćenja točnih pozitivnih detekcija na korištenim testnim video zapisima. Najčešća pogreška praćenja točno pozitivnih detekcija objekata također je prebrzo kretanje praćenog objekta između slijednih okvira video zapisa. Zaključeno je da računalni algoritam za detekciju i praćenje objekata izrađen u C++ programskom jeziku ne obavlja detekciju objekata u stvarnom vremenu, no obavlja praćenje istih. Također, zaključeno je kako oba prethodno navedena algoritma nisu dovoljno pouzdana za korištenje u autonomnim vozilima. Navedeni algoritmi nisu dovoljno pouzdani za korištenje u autonomnim vozilima zbog prethodno navedenih pogrešaka praćenja detektiranih objekata i brzine detekcije objekata.

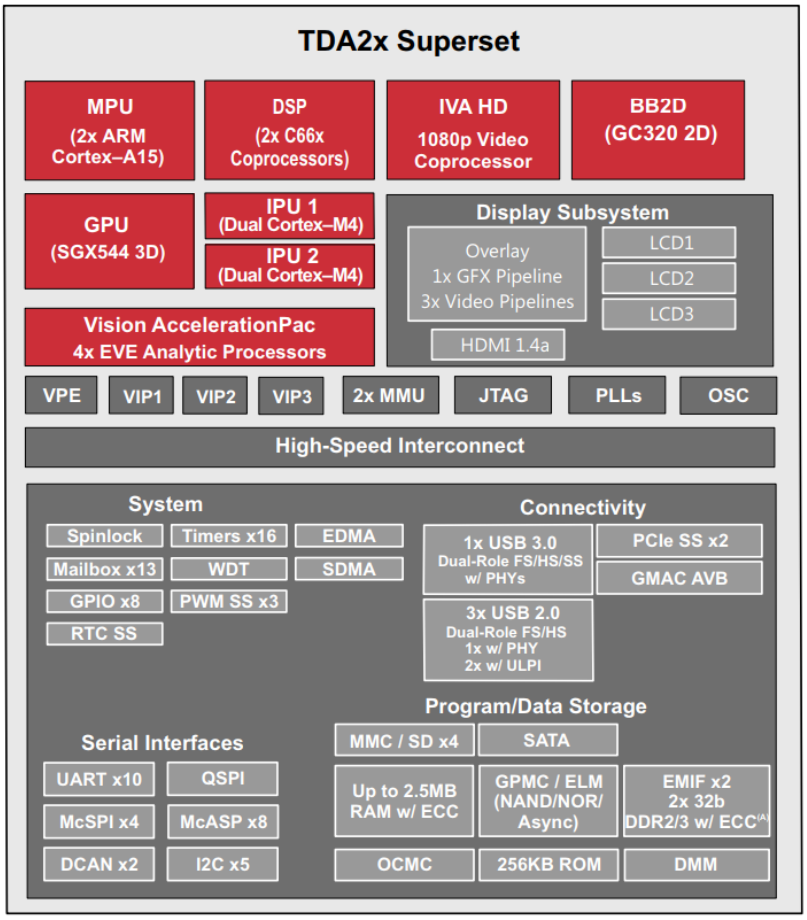
5. IMPLEMENTACIJA ALGORITMA ZA DETEKCIJU I PRAĆENJE OBJEKATA NA ALPHA PLOČU

U ovom poglavlju detaljno je opisana implementacija vlastitog algoritma za detekciju i praćenje objekata na Alpha ploču. U potpoglavlju 5.1. detaljno je opisana Alpha ploča i njezina upotreba [32]. U potpoglavlju 5.2. detaljno je opisan Vision SDK, paket za razvoj programske podrške za TDAx SoC-ove koji se nalaze na Alpha ploči [33]. U potpoglavlju 5.3. opisani su problemi prilikom implementacije vlastitog algoritma za detekciju i praćenje objekata na Alpha ploču.

5.1. Alpha ploča

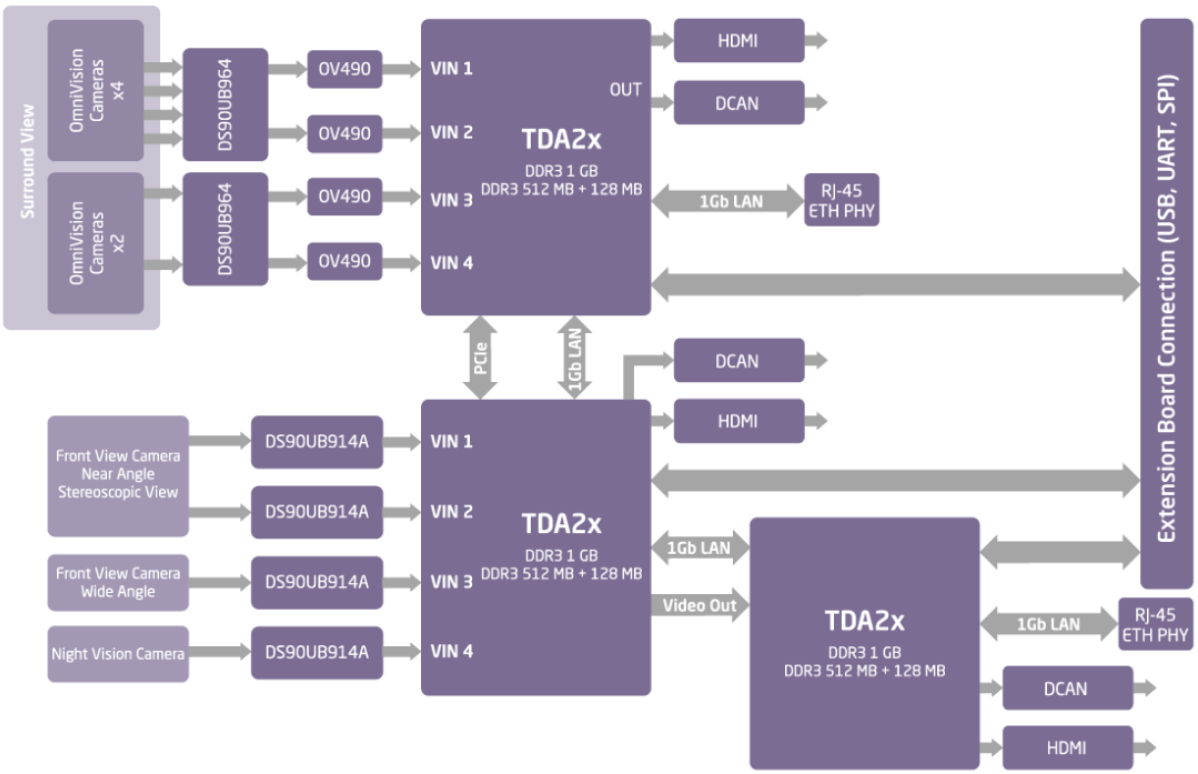
U ovom potpoglavlju detaljnije je opisana Alpha ploča i njezina upotreba. Alpha ploča predstavlja realnu platformu koja služi za testiranje ADAS rješenja. Služi za testiranje osnovnih i naprednih sustava upozorenja, polu-autonomnih operacija, itd. Također, služi za testiranje algoritama koji detektiraju objekte, klasificiraju ih, nadziru stanje vozača, itd. Tako su smanjene ljudske pogreške prilikom vožnje, prometne nesreće i samim time ljudima je omogućena veća sigurnost.

Alpha ploča sadrži tri TDA2x SoC-a (engl. *System on Chip*) te svaki od njih sadrži sljedeće procesore: dva ARM Cortex A15, dva ARM Cortex M4, dva C66x Digital Signal Processor (DSP) i četiri Embedded vision Engine (EVE) procesora. SoC predstavlja integrirani krug koji sadrži različite elektroničke komponente koje funkcioniraju zajedno kako bi ostvarile zajednički cilj. Također, svaki TDA2x SoC sadrži HDMI (engl. *High-Definition Multimedia Interface*) izlaz, Ethernet, UART (engl. *Universal Asynchronous Receiver/Transmitter*), JTAG i DCAN sučelja te utor za Micro SD karticu. Na slici 5.1. prikazan je blok dijagram TDA2x SoC-a [34].



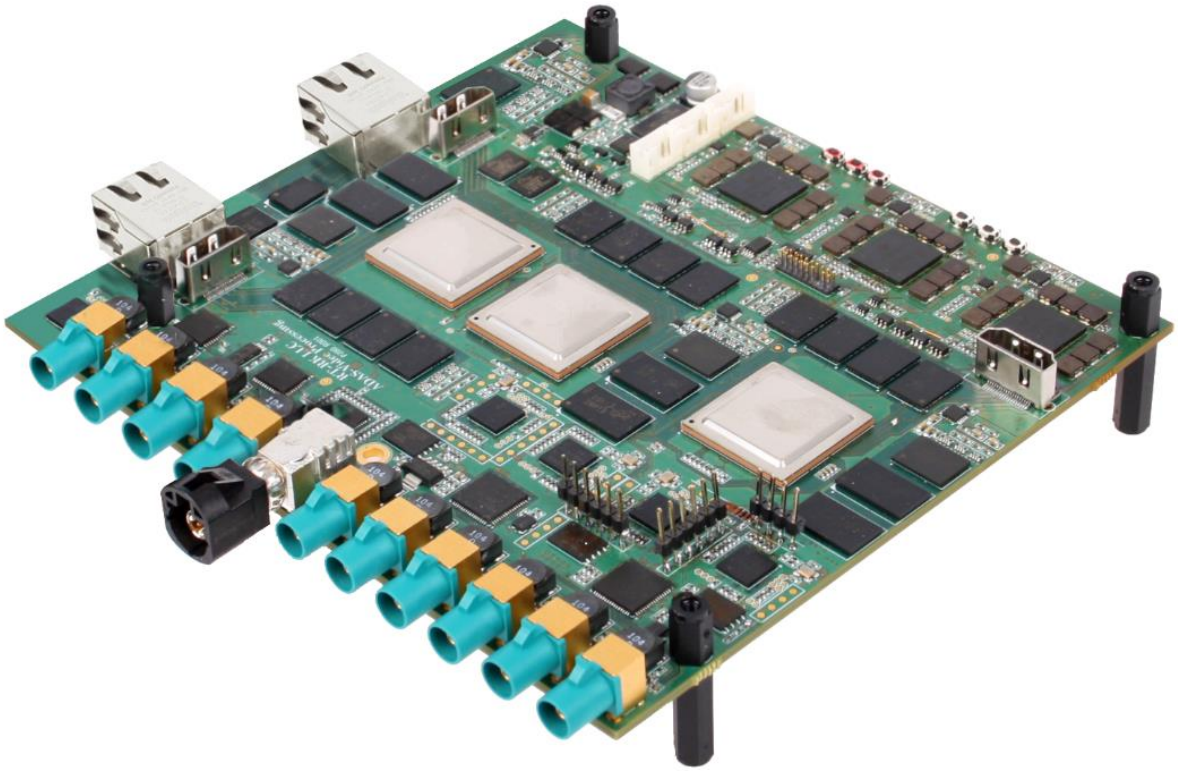
Sl. 5.1. *Blok dijagram TDA2x SoC-a [34]*

Prvi SoC nazvan je SCV (engl. *Surround Camera View*) te podržava šest kanala video prikaza i služi za prikaz okoline vozila. Drugi SoC nazvan je FFN (engl. *Front View Camera Near Angle Stereoscopic View, Front View Camera Wide Angle, Night Vision Camera*) te podržava četiri kanala video prikaza i služi za prikaz slike ispred vozila. Treći SoC nazvan je FUS (engl. *Fusion*) te služi za sintezu informacija dobivenih pomoću senzora priključenih na dva prethodno navedena SoC-a. Prilikom testiranja algoritama omogućeno je balansiranje opterećenja između TDA2x SoC-ova korištenjem PCIe-a (engl. *Peripheral Component Interconnect Express*). Tako je omogućeno optimalno opterećenje. Za komunikaciju s računalom korištena su UART i Ethernet sučelja. Izrađena aplikacija učitana je korištenjem Micro SD kartice. Korištenjem HDMI izlaza prikazana je izlazna slika na zaslonu. Alpha ploča podržava korištenje deset kamera, a osim toga sadrži i podršku za AVB (engl. *Audio Video Bridging*) Ethernet. Na slici 5.2. prikazan je blok dijagram Alpha ploče [32].



Sl. 5.2. *Blok dijagram Alpha ploče [32]*

Na slici 5.3. prikazana je Alpha ploča [32].



Sl. 5.3. *Alpha ploča [32]*

5.2. Vision SDK

U ovom potpoglavlju opisan je Vision SDK, paket za razvoj programske podrške za TDAx SoC-ove koji se nalaze na Alpha ploči. Prilikom implementacije i testiranja vlastitog algoritma na Alpha ploči korišten je Vision SDK. Vision SDK je višeprocesorski paket za razvoj programske podrške korišten za TDAx SoC-ove [33]. Omogućuje korisnicima izradu ADAS rješenja koja koriste video zapise, njihovu obradu, njihov prikaz, itd. Osim toga, sadrži i slučajeve uporabe (engl. *use cases*) i algoritme koji korisnicima prikazuju način korištenja TDAx SoC-ova. Korištena je 3.0.3. inačica Vision SDK-a koja je dostupna na službenoj stranici Texas Instrumentsa [33].

5.3. Problemi prilikom implementacije algoritma za detekciju i praćenje objekata na Alpha ploču

U ovom potpoglavlju detaljno su opisani svi problemi prilikom implementacije vlastitog algoritma za detekciju i praćenje objekata na Alpha ploču. Korištenjem dostupnog Vision SDK također je moguće koristiti dostupne slučajeve uporabe i algoritme. Postoji slučaj uporabe koji koristi OpenCV biblioteku i algoritam za detekciju rubova te je na temelju toga moguće implementirati vlastiti slučaj uporabe i algoritam koji koristi funkcije iz OpenCV biblioteke. OpenCV biblioteku i C++ programski jezik jedino je moguće koristiti na ARM Cortex A15 procesoru Alpha ploče. Implementirana je 3.1.0. inačica OpenCV biblioteke. Implementiran je jedino običan OpenCV modul, ne i dodatni. Korištenje OpenCV funkcija za praćenje detektiranih objekata nije moguće jer su one implementirane u dodatnom OpenCV modulu. To je prvi problem prilikom implementacije vlastitog algoritma za detekciju i praćenje objekata na Alpha ploču, no ne i jedini. Navedeni problem moguće je riješiti implementiranjem svih potrebnih funkcija za praćenje detektiranih objekata korištenjem OpenCV izvornog koda dodatnog modula [17].

Drugi problem prilikom implementacije vlastitog algoritma za detekciju i praćenje objekata na Alpha ploču je nemogućnost korištenja funkcija za rukovanje datotekama iz OpenCV biblioteke. Zbog toga nije moguće korištenje funkcije *load* klase *CascadeClassifier* koja služi za učitavanje detektora. Korištene su dostupne funkcije za rukovanje datotekama (*File_open*, *File_close*, itd.). Osim toga, implementirana je vlastita funkcija za učitavanje detektora. Funkcija je implementirana na računalo korištenjem izvornog koda inačice 1.0. OpenCV biblioteke [35]. Navedenu funkciju nije moguće implementirati na Alpha ploču zbog ograničene količine memorije stoga. Prilikom pozivanja rekurzivne funkcije (funkcije koja poziva samu sebe) nekoliko puta dolazi do pojave pogreške i završetka rada zbog nedovoljne količine memorije stoga. Zbog

toga, zaključeno je kako vlastiti algoritam za detekciju i praćenje objekata trenutno nije moguće implementirati na Alpha ploču.

6. ZAKLJUČAK

Razina autonomije vozila svakoga dana sve više raste. Autonomna vozila trebaju obavljati svoje funkcije bez potrebe za intervencijom vozača, a jedna od osnovnih funkcija koju trebaju obavljati je detekcija objekata ispred vozila. U ovom diplomskom radu prikazano je rješenje za detekciju objekata ispred vozila pomoću kamere na prednjoj strani vozila. Odabran je Viola-Jones algoritam za detekciju objekata kao temelj za izradu vlastitih detektora. Prikupljeni su skupovi slika i označeni su traženi objekti na njima te su uspješno izrađeni detektori vozila, pješaka, semafora i prometnih znakova te su provedena njihova testiranja. Provedenim testiranjem zaključeno je kako detektor bočne strane automobila ima najveću preciznost (1), a detektor prometnog znaka raskrižja s cestom koja ima prednost prolaska ima najveći odziv (0,9775) na testnom skupu slika. Također, najveću brzinu detekcije ima detektor prometnog znaka raskrižja s cestom koja ima prednost prolaska. Nadalje, uspješno je izrađen algoritam za detekciju i praćenje detektiranih objekata u Python i C++ programskom jeziku. Prilikom izrade algoritma u Python programskom jeziku korištena je OpenCV biblioteka za detekciju objekata, a za praćenje detektiranih objekata dlib biblioteka. Prilikom izrade algoritma u C++ programskom jeziku korištena je jedino OpenCV biblioteka. Korištena je i za detekciju objekata i za njihovo praćenje. Unutar OpenCV biblioteke odabrana je Median Flow metoda za praćenje objekata. Prilikom izrade algoritma u C++ programskom jeziku korištena je višenitnost. Provedenim testiranjem prethodno navedenih algoritama zaključeno je kako algoritam za detekciju i praćenje objekata izrađen u C++ programskom jeziku obavlja brže praćenje detektiranih objekata od algoritma izrađenog u Python programskom jeziku. Također, radi i manji broj pogrešaka prilikom praćenja točnih pozitivnih detekcija na korištenim testnim video zapisima. Najčešća pogreška praćenja točno pozitivnih detekcija objekata je prebrzo kretanje praćenog objekta između slijednih okvira video zapisa. Također je zaključeno kako izrađeni algoritmi ne obavljaju detekciju objekata u stvarnom vremenu na računalu. Algoritam izrađen u Python programskom jeziku ne obavlja ni praćenje detektiranih objekata u stvarnom vremenu, dok algoritam izrađen u C++ programskom jeziku obavlja praćenje detektiranih objekata u stvarnom vremenu. Prilikom implementacije vlastitog algoritma za detekciju i praćenje objekata na Alpha ploču zaključeno je kako nije moguća implementacija istog zbog ograničene količine memorije stoga. Prilikom pozivanja rekurzivne funkcije nekoliko puta dolazi do pojave pogreške i završetka rada zbog nedovoljne količine memorije stoga.

LITERATURA

[1] P. Viola, M. Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*, Computer Vision and Pattern Recognition, USA, 2001.

[2] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, *You Only Look Once: Unified, Real Time Object Detection*, The IEEE Conference on Computer Vision and Pattern Recognition, str. 779-788, USA, 2016.

[3] Tensorflow Object Detection API, https://github.com/tensorflow/models/tree/master/research/object_detection, 27.6.2018.

[4] Computer Vision – The Integral Image, https://computersciencesource.wordpress.com/2010/09/03/computer-vision-the-integral-image/, 25.6.2018.

[5] H. Rowley, S. Baluja, T. Kanade, *Neural network-based face detection*, In IEEE Patt. Anal. Mach. Intell., volume 20, str. 22–38, 1998.

[6] OpenCV, https://github.com/opencv/opencv/tree/master/data/haarcascades, 26.6.2018.

[7] Yi-Qing Wang, *An Analysis of the Viola-Jones Face Detection Algorithm*, Image Processing On Line, str. 128-144, Francuska, 2014.

[8] COCO dataset, http://cocodataset.org/#home, 26.6.2018.

[9] Pascal VOC dataset, http://host.robots.ox.ac.uk/pascal/VOC/, 26.6.2018.

[10] S. Ginosar, D. Haas, T. Brown, J. Malik, *Detecting people in cubist art*, In Computer Vision-ECCV 2014 Workshops, str. 101–116, Springer, 2014.

[11] YOLO: Real-Time Object Detection, https://pjreddie.com/darknet/yolov2/, 23.6.2018.

[12] TensorFlow, https://www.tensorflow.org/, 27.6.2018.

[13] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korratikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, K. Murphy, *Speed/accuracy trade-offs for modern convolutional object detectors*, Computer Vision and Pattern Recognition, USA, 2017.

[14] Tensorflow Object Detection API Evaluation, https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md, 15.8.2018.

[15] About OpenCV, https://opencv.org/about.html, 3.7.2018.

[16] OpenCV: Open Source Computer Vision Library, https://github.com/opencv/opencv, 4.7.2018.

[17] Repository for OpenCV's extra modules, https://github.com/opencv/opencv_contrib, 4.7.2018.

[18] Ubuntu 16.04: How to install OpenCV, https://www.pyimagesearch.com/2016/10/24/ubuntu-16-04-how-to-install-opencv/, 3.7.2018.

[19] D. E. King, *Dlib-ml: A Machine Learning Toolkit*, Journal of Machine Learning Research, str. 1755-1758, USA, 2009.

[20] Dlib, https://github.com/davisking/dlib, 18.7.2018.

[21] Haar cascade negative sample images, https://github.com/JoakimSoderberg/haarcascade-negatives, 13.7.2018.

[22] J. Krause, M. Stark, J. Deng, L. Fei-Fei, *3D Object Representation for Fine-Grained Categorization*, IEEE International Conference on Computer Vision Workshops, Australija, 2013.

[23] Udacity, Self-driving car dataset, https://github.com/udacity/self-driving-car/tree/master/datasets, 13.7.2018.

[24] J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, *Man vs. Computer: Benchmarking machine learning algorithms for traffic sign recognition*, Neural Networks, str. 323-333, 2012.

[25] INRIA Person Dataset, http://pascal.inrialpes.fr/data/human/, 13.7.2018.

[26] OpenCV Annotation tool, https://github.com/opencv/opencv/blob/master/apps/annotation/opencv_annotation.cpp, 7.7.2018.

[27] OpenCV Create Samples, https://github.com/opencv/opencv/blob/master/apps/createsamples/createsamples.cpp, 7.7.2018.

[28] OpenCV Train Cascade, https://github.com/opencv/opencv/tree/master/apps/traincascade, 7.7.2018.

[29] Z. Kalal, K. Mikolajczyk, J. Matas, *Forward-Backward Error: Automatic Detection of Tracking Failures*, International Conference on Pattern Recognition, Turska, 2010.

[30] Object Tracking Using OpenCV (C++/Python), https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/#disqus_thread, 18.8.2018.

[31] Confusion Matrix in Machine Learning, https://www.geeksforgeeks.org/confusion-matrix-machine-learning/, 10.7.2018.

[32] Alpha board, http://www.rt-rk.com/services/automotive, 27.7.2018.

[33] Vision SDK, http://www.ti.com/tool/PROCESSOR-SDK-TDAX, 27.7.2018.

[34] TDA2x ADAS Application Processor 23mm Package (ABC) Silicon Revision 2.0 datasheet (Rev. D), http://www.ti.com/lit/ds/symlink/tda2.pdf, 6.8.2018.

[35] Releases – OpenCV library, https://sourceforge.net/projects/opencvlibrary/files/, 5.8.2018.

SAŽETAK

Tema diplomskog rada je „Detekcija objekata ispred vozila pomoću kamere na prednjoj strani vozila“. Detekcija objekata jedna je od funkcija koju autonomna vozila trebaju obavljati. Naziv sustava koji obavljaju tu funkciju je ADAS. U ovom radu odabran je Viola-Jones algoritam za detekciju objekata kao temelj za izradu vlastitih detektora. Izrađeni su detektori vozila, pješaka, semafora i prometnih znakova. Detektori su izrađeni korištenjem OpenCV biblioteke. Provedena su testiranja svih izrađenih detektora te su prikazani njihovi rezultati. Algoritam za detekciju i praćenje objekata u video zapisu izrađen je u C++ i Python programskim jezicima. Testiranjem izrađenih algoritama na računalu zaključeno je kako izrađeni algoritmi ne obavljaju detekciju objekata u stvarnom vremenu. Algoritam izrađen u Python programskom jeziku ne obavlja ni praćenje detektiranih objekata u stvarnom vremenu, dok algoritam izrađen u C++ programskom jeziku obavlja praćenje detektiranih objekata u stvarnom vremenu. Opisana je Alpha ploča i njezina upotreba. Prikazani su svi problemi prilikom implementacije vlastitog algoritma za detekciju i praćenje objekata na Alpha ploču.

Ključne riječi: *detekcija objekata*, *ADAS*, *TDA2x*, *detektor vozila*

ABSTRACT

The topic of the master work is “Object detection in front of autonomous vehicle using dashboard camera”. Object detection is a process of finding objects on an image or video. It is a function that autonomous vehicles need to do. Systems that detect objects and help driver in the driving process are called ADAS. In this paper Viola-Jones algorithm is used to create detectors. Four types of detectors are made (vehicle, pedestrian, traffic light and traffic sign). Detectors are made using OpenCV library. All detectors are tested and their results are shown. Algorithm for object detection and tracking is written in C++ and Python programming languages. After testing the algorithms on computer, it is concluded that neither of them detect objects in real time. Algorithm written in Python programming language does not track objects in real time, while algorithm written in C++ programming language does. Also, Alpha board and its usage are described. All problems are described while algorithm for object detection and tracking is implementing on Alpha board.

Keywords: *object detection, ADAS, TDA2x, vehicle detector*

ŽIVOTOPIS

Juraj Ciberlin rođen je 20. veljače 1995. godine u Našicama. Godine 2009. upisuje srednju školu u Našicama, smjer opća gimnazija te sve razrede završava s odličnim uspjehom. Nakon završene srednje škole, godine 2013. upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Godine 2016. stječe akademski naziv sveučilišni prvostupnik (baccalaureus) inženjer računarstva. Iste godine upisuje diplomski sveučilišni studij računarstva, smjer programsko inženjerstvo na istom fakultetu. Odlično poznaje engleski jezik u govoru i pismu.

Potpis: