Praćenje stanja vozača pomoću kamere u vozilu

Mašanović, Luka

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:255470

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

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

Sveučilišni studij

PRAĆENJE STANJA VOZAČA POMOĆU KAMERE U VOZILU

Diplomski rad

Luka Mašanović

Osijek, 2018.

SADRŽAJ

1. UVOD 1
2. PREGLED POSTOJEĆIH RJEŠENJA ZA PRAĆENJE STANJA VOZAČA 2
3. RAZVOJ VLASTITOG RJEŠENJA ZA PRAĆENJE STANJA VOZAČA 6
3.1. Viola Jones algoritam 6
3.1.1 Haarove značajke 6
3.1.2. Integralna slika 7
3.1.3. *AdaBoost* 8
3.1.4. Kaskada klasifikatora 9
3.2. *OpenCV* 10
3.3. Algoritam za praćenje stanja vozača 10
3.3.1. Detekcija lica i očiju 12
3.3.2. Provjera očiju 12
3.3.3. Provjera spuštenosti glave 13
3.3.4. Provjera položaja glave 14
3.3.5. Detekcija i provjera profila lica 14
3.4. Treniranje vlastitog detektora 15
3.5. Implementacija algoritma na realnoj platformi za pomoć vozaču pri vožnji 16
4. TESTIRANJE UČINKOVITOSTI NOVOG ALGORITMA ZA PRAĆENJE STANJA VOZAČA 20
5. ZAKLJUČAK 27
LITERATURA 28
SAŽETAK 30
ŽIVOTOPIS 31
PRILOZI 32

1. UVOD

Vožnja je mnogim ljudima postala sastavni dio svakodnevnog života te zahtjeva njihovu potpunu pozornost. U današnjem užurbanom svijetu gdje su ljudi pod stalnim pritiskom, stresom i drugim iscrpljujućim situacijama, njihova sposobnost upravljanja vozilom može biti smanjenja. Prema statističkim podacima Svjetske zdravstvene organizacije, ukupan broj smrtno stradalih osoba godišnje u cestovnom prometu iznosi 1.3 milijuna dok 20 do 50 milijuna bude ozlijeđeno [1]. Nacionalna zaklada za spavanje (engl. *Nacional Sleep Foundation*) izvijestila je da je 2009. godine 54% odraslih vozača upravljalo vozilom dok se osjećalo pospano, a njih 28% zaspalo je prilikom vožnje [2]. Postoji nekoliko čimbenika koji mogu pridonijeti pospanosti. Među najvažnijima su umor (mentalni ili fizički), nedostatak sna te konzumiranje droga [3]. Kako bi se ove brojke smanjile i ovakva stanja vozača na vrijeme otkrila, potreban je stalan nadzor u vozilu na temelju kojeg bi samo vozilo pravovremeno reagiralo i izbjeglo potencijalnu nesreću.

Kao rješenje ovog problema sve se češće koriste različiti sustavi koji vožnju automobilom trebaju učiniti udobnijom i sigurnijom. Jedan od takvih sustava je i sustav koji prati stanje vozača, a temelji se na analizi video signala dobivenog s kamere koja se nalazi na instrument ploči i okrenuta je prema glavi vozača. Neke od glavnih funkcionalnosti ovakvog sustava su identifikacija vozača, praćenje budnosti vozača uz zvučna upozorenja, nadzor pozornosti vozača u kritičnim situacijama, prilagodba grafičkog sučelja instrument ploče i sl.

Zadatak ovog rada je razvoj algoritma koji će na temelju video signala dobivenog s kamere detektirati vozačevu glavu u svakom okviru, detektirati gleda li vozač ispred vozila ili ne te detektirati ima li vozač neuobičajeno dugo zaklopljene oči. Algoritam je potrebno napisati u programskom jeziku C, izvršiti njegovu optimizaciju te ga prilagoditi za dostupnu ADAS (engl. *advanced driver-assistance systems*) ploču kako bi se testirala njegova učinkovitost na realnoj platformi.

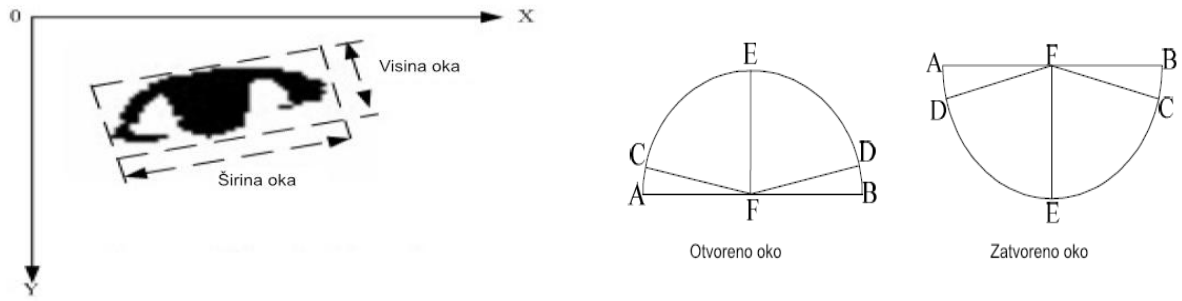
U drugom poglavlju rada dan je pregled postojećih rješenja za praćenje stanja vozača. U trećem poglavlju objašnjena je izrada vlastitog algoritma korištenjem *OpenCV* biblioteke. Četvrto poglavlje opisuje rezultate testiranja izrađenog algoritma. Zadnje poglavlje daje zaključna razmišljanja vezana za cjelokupni rad te navodi moguća poboljšanja.

2. PREGLED POSTOJEĆIH RJEŠENJA ZA PRAĆENJE STANJA VOZAČA

U ovom poglavlju dan je pregled postojećih rješenja za nadzor vozača. Sustav za nadzor stanja vozača je zamišljen tako da vozilo detektira kada je vozač umoran ili pospan te u tom trenutku poduzme potrebnu akciju. Ukoliko vozilo primijeti pospanost vozača, može ga upozoriti zvučnim signalom (npr. puštanjem glazbe), porukom da mora stati i odmoriti ili vibriranjem zaštitnog pojasa [4].

Postupak detekcije umora vozača nije jednostavan, zbog toga su korišteni različiti pristupi. Sustavi za praćenje stanja vozača mogu koristiti četiri različita pristupa, a to su: pristup zasnovan na biološkim (fiziološkim) pokazateljima, pristup zasnovan na ponašanju vozila, pristup zasnovan na analizi lica te hibridni pristup. Kod prvog pristupa mjere se biološke promjene kao što su otkucaji srca, moždani valova, krvni tlak i sl. Ovakvi uređaji zahtijevaju da vozač nosi senzor kako bi se mogli mjeriti određeni parametri. Zbog toga navedeni sustavi spadaju pod nametljive jer smetaju vozaču prilikom vožnje. Takav sustav bi se mogao lako zaobići jednostavnim ne korištenjem senzora kako bi se izbjegla potencijalna upozorenja, što nije poželjan rezultat. Kod pristupa zasnovanog na ponašanju vozila, koriste se podaci prikupljeni iz vozila, kao što su odstupanje iz trake, brzina vozila, pokreti volana i sl. U pristupu zasnovanog na analizi lica koristi se računalni vid kako bi odredili stanje vozača. Detektiranjem i praćenjem značajki vozačevog lica, očiju, usta, treptaja očiju i otvorenosti kapaka može se odrediti potencijalni umor vozača. Ovakvi sustavi su nenametljivi, ne zahtijevaju nošenje senzora te ne odvlače pažnju prilikom vožnje. Zadnja skupina su hibridna rješenja koja kombiniraju različite metode kako bi što točnije odredili je li vozač umoran. Ovakvi sustavi koriste infracrvena zračenja koja omogućuju snimanje vozača po noći bez ometanja. U nastavku je dan pregled radova koji koriste neke od ovih pristupa.

U radu [5] opisan je sustav za praćenje stanja vozača koji je uspješno testiran u stvarnom vremenu i u različitim svjetlosnim uvjetima. Sustav procjenjuje da li je dan ili noć na temelju svjetline s ulazne slike kamere. Ako je svjetlina veća od određenog praga koristi se algoritam za dnevni način rada, ako je manja koristi se algoritam za noćni način rada. Za dnevni način rada sustava koristi se *AdaBoost* metoda koja detektira lice i oči crtajući pravokutnike oko istih. Nakon detekcije oka, računaju se karakteristični parametri kao što su omjer visine i širine očiju te zakrivljenost gornjeg kapka (Sl.2.1.).



(a) (b)

Sl. 2.1. *Karakteristični parametri (a) omjer visine i širine oka (b) zakrivljenost gornjeg kapka kod otvorenog i zatvorenog oka* [5]

Na osnovnu tih podataka određuje se umor vozača koristeći PERCLOS metodu. PERCLOS određuje stanje vozača na temelju zatvorenosti kapka. Oči se smatraju zatvorene kada je stupanj zatvorenosti oka veći od 80%. Ako sustav utvrdi umor vozača, oglasit će se zvučni alarm. Noćni način rada koristi razliku između infracrvenih okvira s kamere kako bi se detektirale oči. Uz oči kao dodatni pokazatelj umora promatra se zijevanje vozača. Nakon detektiranja očiju moguće je locirati položaj usta prema konstrukciji lica. Za testiranje zijevanja vozača korišten je LPB (engl. *local binary pattern*) algoritam. Kod zijevanja mjeri se otvorenost usta, te se na temelju toga može odrediti razina umora prema duljini i učestalosti zijevanja. Primjer detekcije zijevanje prikazan je slikom 2.2.



(a) (b)

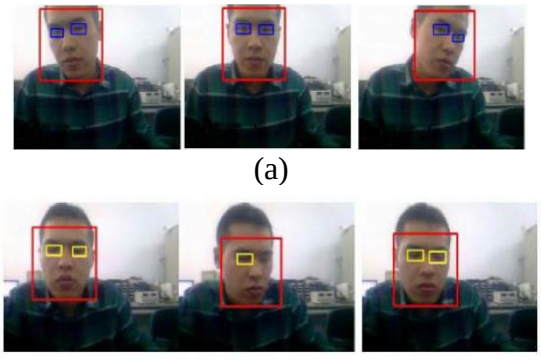
Sl. 2.2. *Uspješna detekcija (a) zijevanja i (b) normalnog stanja usta* [5]

U svom radu Floers i suradnici [6], 2009, predstavili su sustav za pomoć pri vožnji koji automatski detektira umor. Sustav koristi algoritme temeljene na umjetnoj inteligenciji zajedno sa prikupljenim vizualnim podacima s kamere. Sustav identificira, prati lice i oči te određuje pospanost koristeći metodu potpornih vektora SVM (engl. s*upport vector machines*). Sustav je dizajniran za rad u stvarnom vremenu i promjenjivim svjetlosnim uvjetima. Osim što promatra oči

(treptanje) vozača, uzima u obzir i druge faktore kao što su zijevanje, nagib glave i orijentacija lica, što dodatno povećava pouzdanost sustava.

Sahayadhas i suradnici [7] 2013. godine koristili su fiziološke signale kako bi detektirali umor vozača. Korišteni su EKG (mjeri srčane aktivnosti) i EMG (mjeri električnu aktivnost mišića) kao manje nametljivi uređaji za mjerenje fizioloških promjena. Iako nametljiva, fiziološka mjerenja su pouzdana i točna zbog toga što pružaju prave informacije o stanju u kojem se vozač nalazi. Rezultati pokazuju da različite značajke izračunate koristeći rezultate dobivenih iz EKG i EMG pokazuju značajnu razliku između budnog i pospanog stanja. Postoje i druga fiziološka očitanja koja se mogu koristiti kao što su EEG (mjeri električnu aktivnosti mozga) i EOG (mjeri pokrete očiju). Kombinacijom fizioloških signala i sustava temeljenih na računalnom vidu može se postići visoka razina točnosti detekcije umora kod vozača.

Kako bi postigli ranu detekciju umora u stvarnom vremenu u radu [8] je korištena analiza vozačevog lica, točnije stanje u kojem se nalaze oči i usta. Prvo sustav detektira lice koristeći Viola Jones algoritam za detekciju. Nakon toga u području na kojem je detektirano lice, MB-LPB (engl. *multi-block local binary pattern*) značajka se koristi za brzo pronalaženje i detekciju očiju. Kalmanov filtar se koristi za praćenje usta i očiju. Nakon prilagodbe filtra i izračuna elipse koja ima oblik ljudskog oka, slijedi procjena stanja usta i očiju. Kod procjene stanja usta, postavlja se prag koji određuje otvorenost, zatvorenost usta i zijevanje vozača. Kod očiju se postavlja prag za određivanje stanja u kojem se oči nalaze izračunavanjem omjera između duge i kratke osi elipse koja je prethodno primijenjena. Rezultati testiranja pokazali su da ova metoda može brzo i točno detektirati položaj i stanje u kojem se nalaze oči i usta. Na slici 2.3. prikazani su rezultati kod detekcije otvorenih i zatvorenih očiju, primjenom rješenja iz [8].



(a)

(b)

Sl. 2.3. *Primjeri (a) detekcije kada su oči otvorene (b) detekcije kada su oči zatvorene* [8]

U radu [9] predstavljeno je učinkovito mjerenje treptanja oka koje se koristi za određivanje umora vozača. Problem kod praćenje treptanja vozača prilikom dnevne vožnje je kada je vozačevo lice izloženo različitim razinama svjetlosti. Kao rješenje ovog problema Park i suradnici predstavili su formiranje SVM (engl. *support vector machines*) kaskade s dvije razine kako bi povećali točnost detekcije oka. Za efikasno mjerenje treptanja oka, definirali su funkciju odlučivanja koja se temelji na mjerenju pokreta kapka i težini koju generira klasifikator za otkrivanje stanja zatvorenosti oka. Funkcija odlučivanja mjeri kada je oko zatvoreno, a kada otvoreno. Rezultati su pokazali da u dnevnom načinu rada, ova metoda daje bolje rezultate nego uobičajena metoda koja koristi razliku između svijetle i tamne zjenice koja se dobiva reakcijom oka na infracrveno svjetlo. Primjer rada sustava u različitim svjetlosnim uvjetima prikazan je na slici 2.4.



Sl. 2.4. *Rezultati detekcije očiju pri različitim svjetlosnim uvjetima kada se koristi rješenje iz* [9]

3. RAZVOJ VLASTITOG RJEŠENJA ZA PRAĆENJE STANJA VOZAČA

Za izradu vlastitog rješenja korištena je *OpenCV* biblioteka. Rješenje detektira vozačevu glavu i oči koristeći algoritam za raspoznavanje objekata koji su razvili Viola i Jones. Nakon detekcije prati se žmiri li, gleda li u stranu i drži li glavu u spuštenom položaju. U nastavku slijedi detaljnije objašnjenje Viola i Jones algoritma na kojem se temelji ovaj rad.

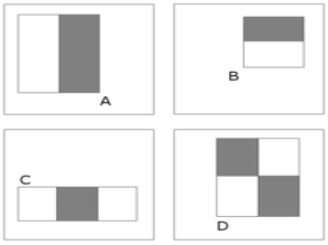
3.1. Viola Jones algoritam

Za detekciju objekata danas se koristit sve popularnija metoda, strojno učenje. Strojno učenje je proces u kojem se računalnim sustavima omogućuje da uče na temelju podataka, bez izričitog programiranja te na temelju tih podataka predvide rezultat.

Kako bi računalo moglo raspoznati nekakav objekt, primjer lice, potrebno mu je dati određeni broj slika lica i „ne-lica“ te ga istrenirati kako bi znalo razliku. Kada je istrenirano, računalo će izvući određene značajke po kojima će raspoznati lice i spremiti ih u datoteku. Pomoću tih značajki bi nakon treniranja na svakoj novoj ulaznoj slici trebalo prepoznati ima li lica ili nema. Na ovom principu radi algoritam kojeg su razvili Viola i Jones. U svom radu [10] su predstavili pristup strojnog učenja za detekciju objekata koji osigurava veliku brzinu i veliku razinu točnosti. Iako je prvenstveno dizajniran za detekciju lica, moguće ga je istrenirati da raspoznaje druge objekte. U radu se spominju četiri bitne stavke za razumijevanje algoritma: Haarove značajke (engl. *Haar Features*), integralna slika, *AdaBoost* i kaskadni klasifikator.

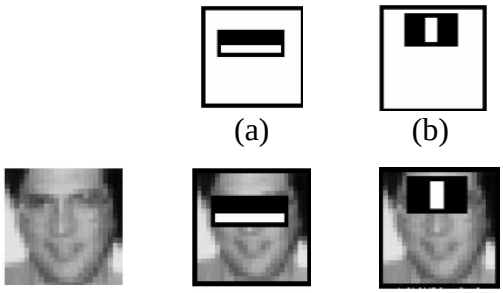
3.1.1 Haarove značajke

Kako bi detektirali objekt, Viola i Jones koriste vrijednosti jednostavnih značajki. Razlog zbog kojeg koriste značajke umjesto direktnog rada s elemntima slike je kao što navode, da sustav temeljen na značajkama radi puno brže od sustava koji se temelji na radu s elementima slike. Točnije koriste tri vrste značajki. Značajku s dva pravokutnika, s tri i s četiri, kao što je prikazano na slici 3.1.



Sl. 3.1. *Haarove značajke* [10]

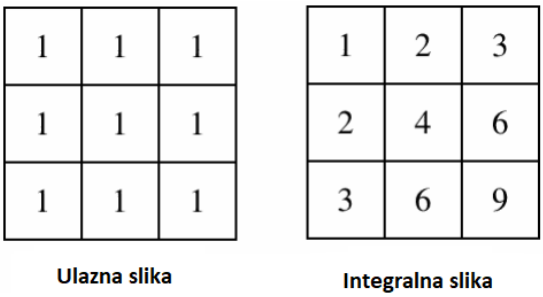
Svaka značajka se primjenjuje na slici pomičući se po elementima slike do kraja slike. Nakon toga postupak se ponavlja. Svakim ponavljanjem mijenjaju se veličina, pozicija i varijacija značajki. Svakim pomakom značajke računa se njena vrijednost, oduzimanja sume elemenata slike koji se nalaze ispod crnog dijela značajke od sume elemenata slike koji se nalaze ispod bijelog dijela značajke. Na mjestima gdje se značajka poklapa s karakteristikom lica dobit će se najveća vrijednost. Primjer je prikazan na slici 3.2. gdje se značajka sa slike 3.2. (a), poklapa s područjem očiju koje je tamnije od gornjeg dijela obraza (d). Također značajka pod (b), poklapa se s područjem nosa koji je svjetlije od očiju (e). Kao što je vidljivo iz primjera, značajke imaju sličnost s karakteristikama lica. Autori su koristili 24x24 elementa slike kao osnovnu rezoluciju detektora te na kraju dobili preko 160 000 značajki. Kako bi ubrzali proces računanja značajki Viola i Jones predstavili su integralnu sliku.



Sl. 3.2. *Primjer (a) Haarova značajka 1 (b) Haarova značajka 2 (c) ulazna slika, (d) primjena Haarove značajke 1 na područje očiju (e) primjena Harrove značajke 2 na područje nosa* [10]

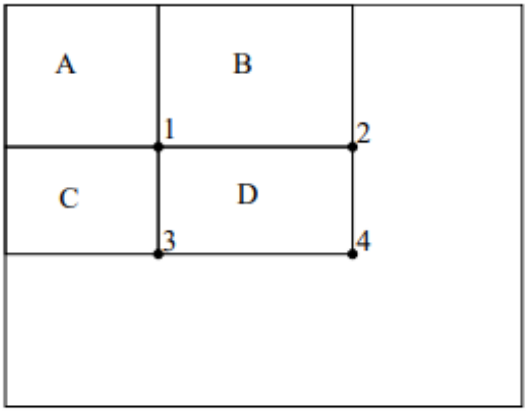
3.1.2. Integralna slika

Svaki puta kada je potrebno izračunati vrijednost značajke, vrijednosti elemenata slike ispod crne strane značajke moraju se zbrojiti, pa oduzeti od sume vrijednosti elemenata slike koji se nalaze ispod bijele strane značajke. Takav postupak je spor. Korištenjem integralne slike značajke mogu biti izračunate velikom brzinom. Na integralnoj slici vrijednost elementa slike na poziciji (x, y) jednaka je zbroju svih elemenata slike lijevo i iznad njegove pozicije uključujući i promatrani element slike, kao što je prikazano na slici 3.3.



Sl. 3.3. *Ulazna slika i njezina pripadna integralna slika* [10]

Korištenjem integralne slike bilo koja pravokutna značajka može biti izračunata koristeći četiri reference (Sl.3.4). Vrijednost integralne slike na lokaciji „1“ predstavlja sumu svih elemenata slike u pravokutniku A. Vrijednost na lokaciji „2“, je A + B, na lokaciji „3“ A + C, te na lokacija „4“ A + B + C + D. Suma svih elemenata slike za pravokutnik D može biti izračunata kao 4 + 1 – (2 + 3). Za razliku između dva pravokutnika potrebno je osam referenci. Zbog toga što su pravokutnici susjedni, razliku između dva susjedna pravokutnika moguće je izračunati koristeći šest referenci, za tri pravokutne značajke potrebno je osam referenci, dok se za četiri pravokutnika koristi devet referenci.



Sl. 3.4. *Prikaz načina računanja integralne slike* [10]

3.1.3. *AdaBoost*

Kao što je ranije spomenuto, u prozoru 24x24 elementa slike može biti i do 160 000+ značajki koje je potrebno izračunati. Iako svaka značajka može biti izračunata velikom brzinom, nisu sve relevantne. Mali broj korisnih značajki može biti formiran u jedan učinkovit klasifikator. *AdaBoost* je algoritam za strojno učenje koji se koristi kako bi se našle najbolje značajke od njih 160 000+. Svaka odabrana značajka od strane *AdaBoosta* smatra se korisnim ako točno klasificira više od 50% uzoraka, odnosno radi bolje od nasumičnog pogađanja. Takve značajke se nazivaju slabi

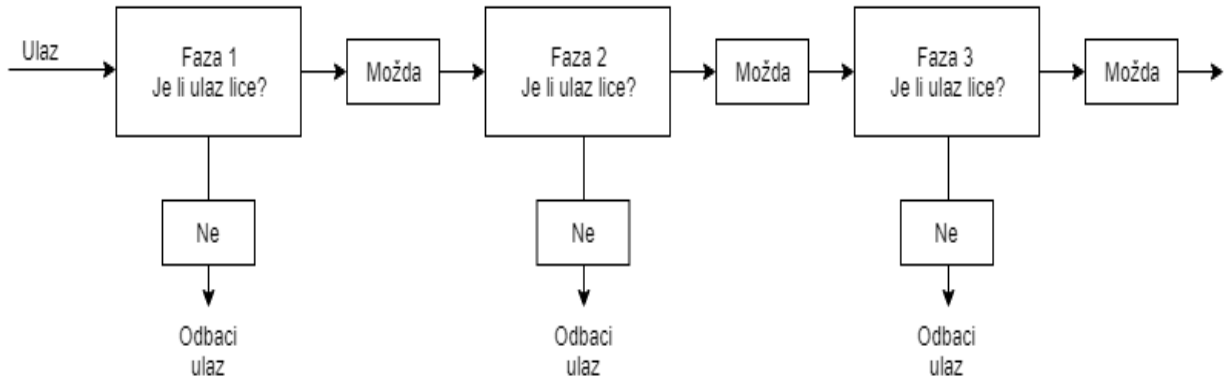
klasifikatori. *AdaBoost* stvara jedan jaki klasifikator kao linearnu kombinaciju slabih klasifikatora, prikazano izrazom (3-1)

$$F(x) = \sum_{n=1}^{N} a_n f_n(x) , \qquad (3\text{-}1)$$

gdje *F(x)* predstavlja jaki klasifikator, *f(x)* slabi klasifikator, a_n težinu pridruženu *n-tom* klasifikatoru i *N* ukupan broj slabih klasifikatora.

3.1.4. Kaskada klasifikatora

Osnovni princip rada Viola i Jones algoritma je takav da detektor prolazi kroz sliku mnogo puta. Čak i nakon konstrukcije jakog klasifikatora, svakim pomakom bi se morale provjeravati dijelovi slike na kojima nema traženog objekta i samim time događalo bi se bespotrebno računanje. Iz tog razloga koristi se kaskada klasifikatora koja je sastavljena od nekoliko faza (engl. *stages*). Sve značajke su grupirane u fazama. Svaka faza sadrži nekoliko značajki. Zadatak svake faze je odrediti nalazi li se u svakom pod-prozoru lice ili ne. Ako se ne nalazi, pod-prozor se automatski odbacuje. S ovim postupkom početni klasifikator prolazi veliki dio slike gdje se ne nalazi lice s vrlo malo obrade. Jednostavniji klasifikatori koriste se kako bi odbacili većinu pod prozora, nakon njih evaluiraju se sve složenije kaskade koje imaju sve manji postotak pogreške. Nakon pozitivnog rezultata prvog klasifikatora pokreće se drugi, pozitivan rezultate drugog pokreće treći itd. Na slici 3.5. nalazi se grafički prikaz kaskade klasifikatora.



Sl. 3.5. *Primjer kaskade klasifikatora koji odlučuje nalazi li se na promatranom pod-prozoru lice* [10]

3.2. *OpenCV*

Kao pomoć za izradu algoritma korištena je *OpenCV* biblioteka koju je razvio Intel, no danas je to biblioteka otvorenog koda, koja sadrži algoritme za strojno učenje i računalni vid. *OpenCV* je izgrađen kako bi omogućio zajedničku infrastrukturu za aplikacije računalnog vida. Biblioteka sadrži više od 2500 optimiziranih algoritama koji uključuju skup klasičnih i najsuvremenijih algoritama strojnog učenja i računalnog vida [11]. Jedan od tih algoritama je i gotova implementacija Viola i Jones algoritma koja je korištena u ovom radu, uz neke druge funkcije koje će biti objašnjenje u nastavku.

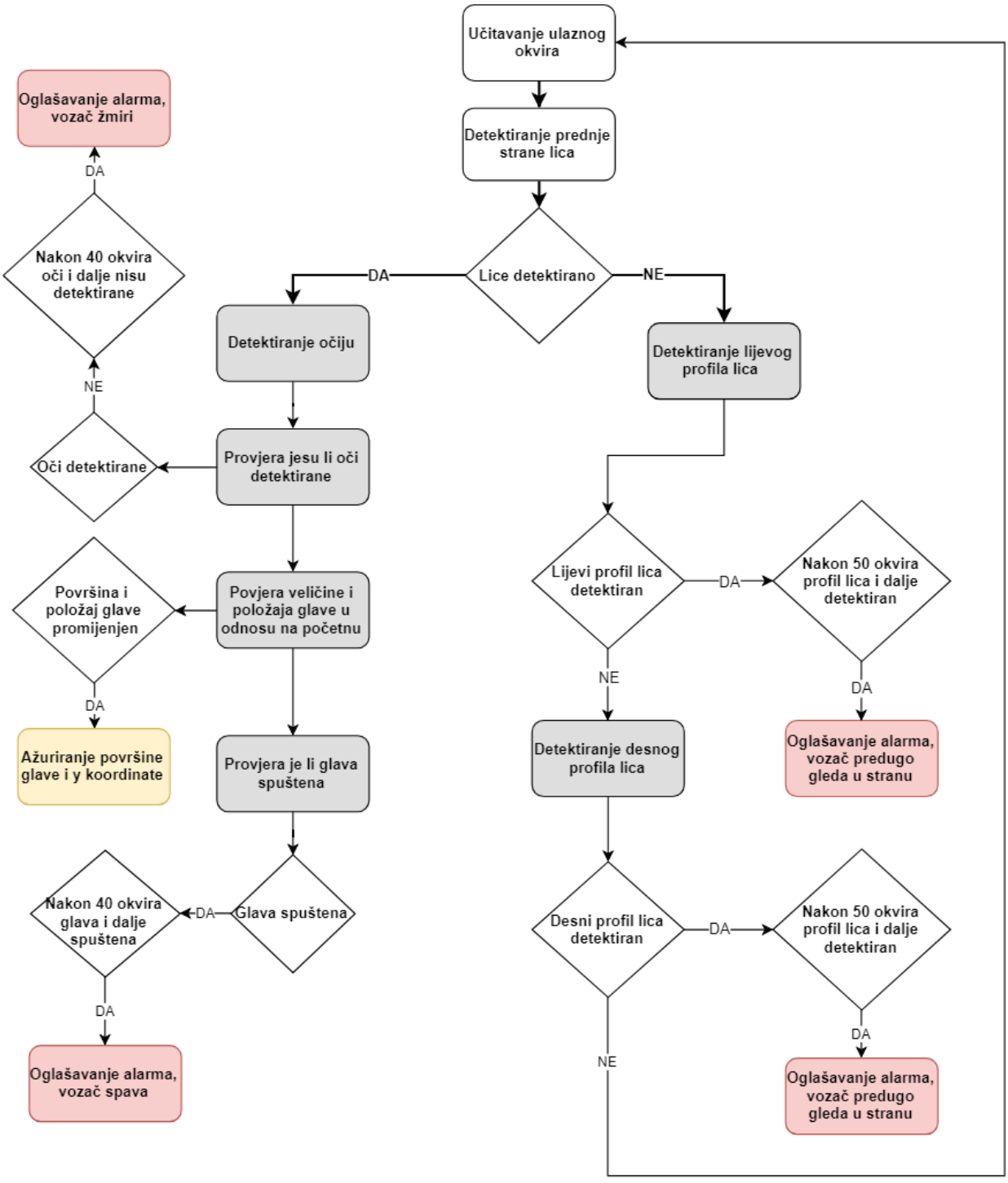
3.3. Algoritam za praćenje stanja vozača

Zadatak diplomskog rada bio je razviti algoritam koji će detektirati kada vozač ima spuštenu glavu prema dolje, gleda li u stranu te jesu li mu oči zatvorene. Rješenje je razvijeno u C++ programskom jeziku. Programski kod razvijenog rješenja nalazi se u P.3.1.

U sustavu za praćenje stanja vozača korištena su tri detektora. Detektori se temelje na ranije objašnjenom Viola i Jones algoritmu. Korišteni detektori su već prethodno istrenirani i pohranjeni u *xml* datoteke. *OpenCV* pruža mogućnost korištenja više gotovih detektora, u ovom radu korišten je detektor za prednju stranu i profil lica te za oči [12]. Također moguće je istrenirati i vlastiti detektor što će biti objašnjeno u kasnijem dijelu rada. *OpenCV* sadrži puno gotovih funkcija. Korištene funkcije pri izradi algoritma su sljedeće:

- *cascadeClassifier()* – učitava klasifikator iz datoteke.
- *detectMultiScale()* – glavna funkcija koja detektira objekte ovisne o učitanoj *xml* datoteci. Bitni parametri ove funkcije su:
    - *Faktor skaliranja* (engl. *scale factor*) – određuje koliko se dimenzija ulazne slike smanji pri svakoj iteraciji. Ovaj parametar omogućava detekciju velikih i malih lica korištenjem istog detekcijskog prozora.
    - *Minimalni broj susjeda* (engl. *min neighbors*) – parametar koji određuje koliko susjeda svaki pravokutnik treba imati da bi bio označen kao traženi objekt. U poglavlju testiranje ovaj parametar će biti detaljnije objašnjen.
- *cvtColor()* – funkcija za pretvaranje ulazne slike u crno bijelu sliku.
- *rectangle()* – funkcija za crtanje pravokutnika.
- *flip()* – funkcija koja okreće sliku oko y-osi, postiže se efekt zrcala.
- *putText()* – funkcija korištena za ispis poruka upozorenja na ekran.

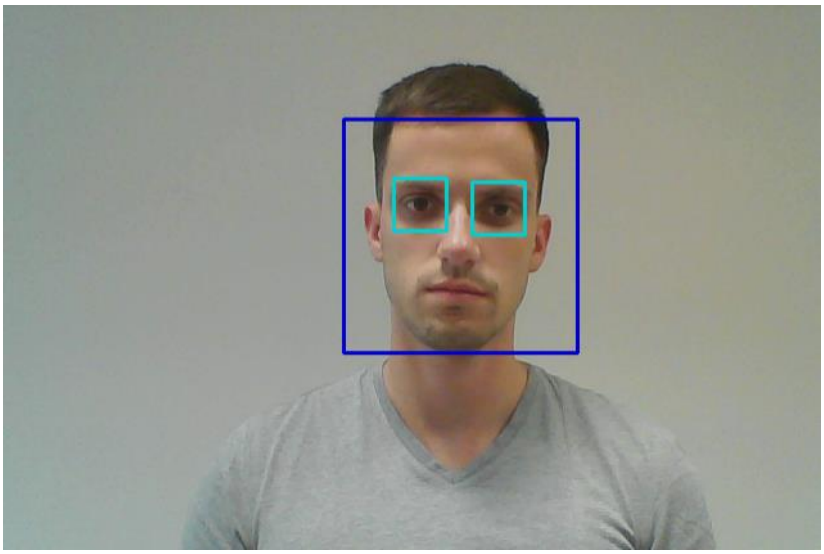
Algoritam se sastoji od dva dijela. Na početku pokušava detektirati prednju stranu lica vozača. Ako ne uspije, prelazi na drugi dio koji pokušava detektirati profil lica ukoliko vozač gleda u stranu. Razlog korištenja dva detektora je što karakteristike profila i prednje strane lica nisu iste. Korištenjem samo detektora za prednju stranu lica ne bi se mogli pratiti vozačevi pogledi u stranu. Dijagram toka algoritma prikazan je na slici 3.6.



Sl. 3.6. *Dijagram toka algoritma za praćenje stanja vozača izrađenog u ovom diplomskom radu*

3.3.1. Detekcija lica i očiju

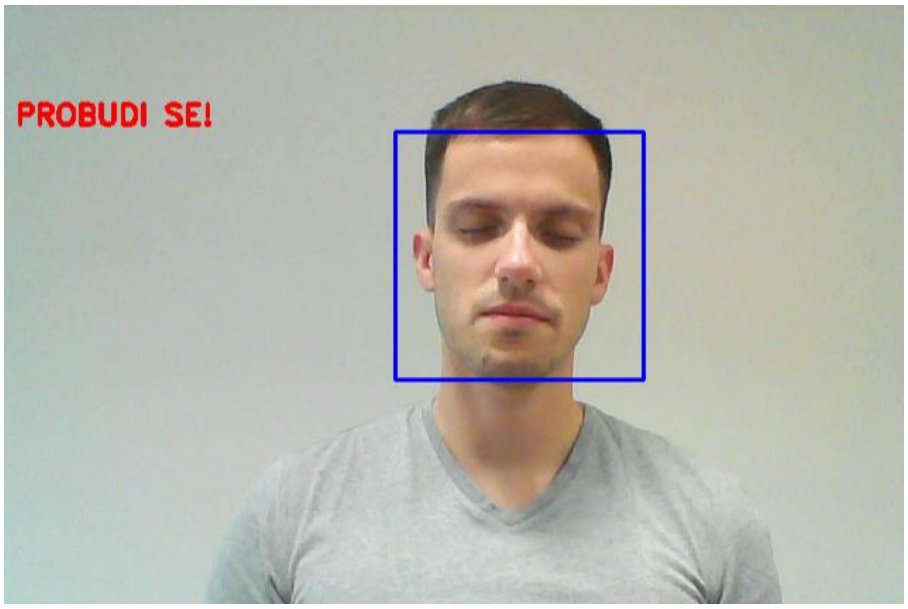
Prije početka rada algoritama učitavaju se potrebni klasifikatori koji se nalaze u *xml* datotekama. Algoritam radi tako da provjerava svaki okvir te na temelju njega poduzima određene radnje. Nakon prvog okvira u kojem se pojavi lice, poziva se funkcija za detekciju lica. Ako je ta detekcija bila uspješna, poziva se funkcija koja se izvršava samo jednom. Ta funkcija uzima početne podatke s detektiranog lica, a to su veličina glave i koordinate položaja glave. Veličina glave se računa koristeći visinu i širinu pravokutnika koji je označio lice. Početne točke će kasnije služiti za usporedbu prilikom računanja spuštenosti glave i udaljenosti vozačeve glave od kamere. Algoritam prvo pokušava detektirati lice, ako uspije prelazi se na idući korak. Ukoliko se u kadru nalazi više ljudi, što je lako moguće, jer kamera koja snima vozača može snimiti i putnike iza vozača, algoritam će detektirati najveće lice, odnosno lice koje se nalazi najbliže kameri, a to je vozač. Oči se detektiraju samo ako je prednja strana lica uspješno detektirana. Algoritam pretražuje dio unutar kojeg se nalazi detektirano lice jer je to jedino mjesto na kojem se oči mogu nalaziti. Slika 3.7. prikazuje uspješno detektirano lice i oči.



Sl. 3.7. *Uspješna detekcija prednje strane lica i očiju pomoću stvorenog algoritma*

3.3.2. Provjera očiju

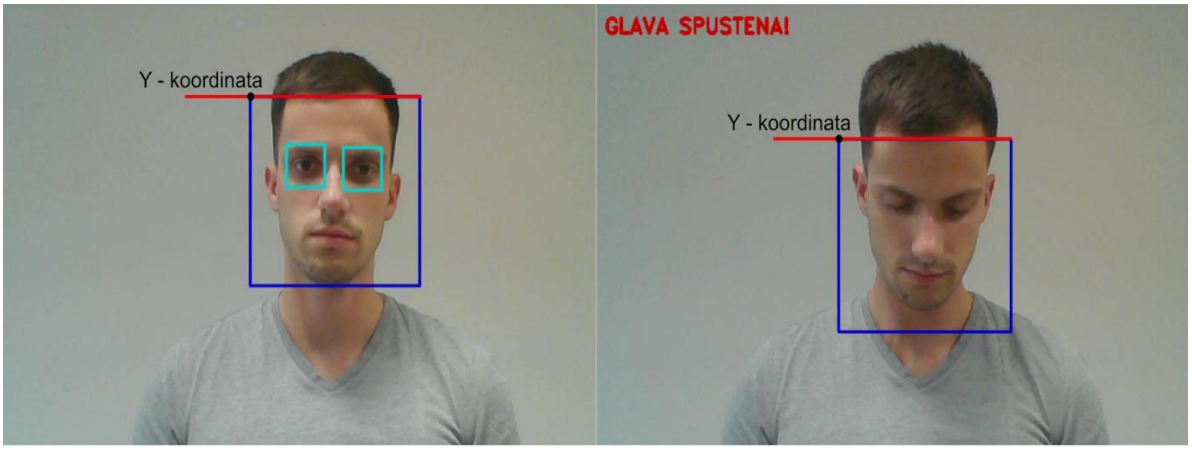
Svaki puta kada je lice detektirano, a oči nisu, poziva se funkcija koja mjeri zatvorenost očiju. Brojač prati koliko dugo oči nisu detektirane. Ukoliko brojač pređe određeni vremenski prag, oglasit će se alarm koji implicira da vozač najvjerojatnije žmiri. Slika 3.8. prikazuje ispis poruke kada vozač žmiri duže od navedenog praga.



Sl. 3.8. *Ispis poruke kada osoba žmiri dulje od prethodno definiranog perioda*

3.3.3. Provjera spuštenosti glave

Iduća provjera koja se radi je spuštenost glave. Provjerava se *y* koordinata okvira detektiranog lica. Ova provjera uspoređuje početnu koordinatu *y* osi koja se nalazi u gornjem lijevom kutu, s trenutnom *y* koordinatom gornjeg lijevog kuta pravokutnika. Ukoliko vozač spusti glavu prema dolje, *y* koordinata se mijenja. Ako vozač glavu drži spuštenu određeni broj sekundi aktivira se alarm i prikazuje se poruka na ekranu. Normalno i spušteno stanje glave prikazano je na slici 3.9.

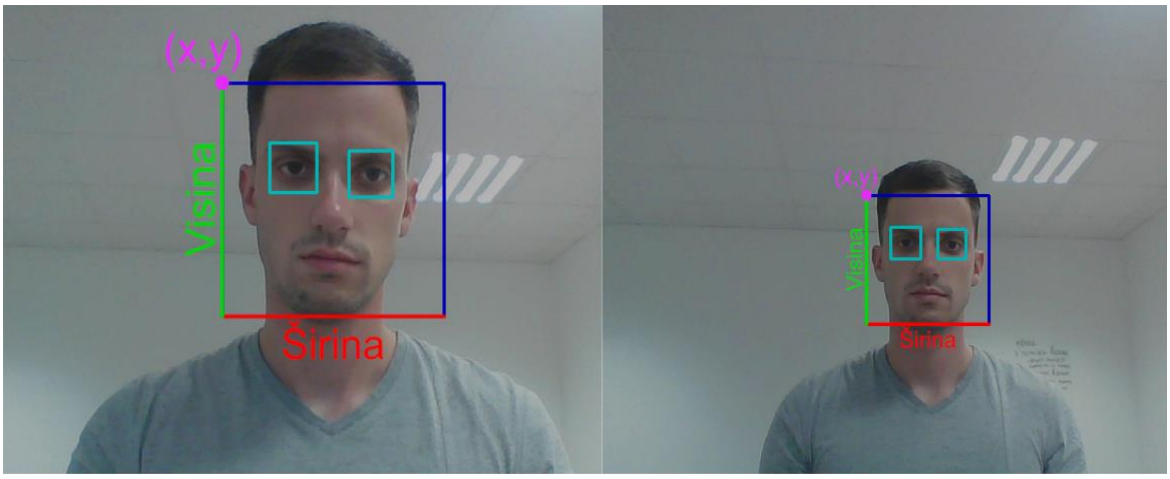


(a) (b)

Sl. 3.9. *Prikaz glave u (a) normalnom položaju (b) spuštenom položaju*

3.3.4. Provjera položaja glave

Kako je moguće da se vozač udalji ili približi kameri tijekom vožnje, pomicanjem sjedala ili promjenom položaja sjedenja pomaknut će se i položaj promatrane *y* koordinate. Takva radnja se može prepoznati promjenom površine označenog okvira, što je vidljivo sa slike 3.10. gdje se vozač nalazi blizu kamere te je površina označenog okvira veća. Ako se vozač udalji površina se smanji, a samim time se promjeni i položaj *y* koordinate. Površina okvira se računa množenjem visine i širine. Ovakvom promjenom položaja i udaljenosti od kamere dobili bi se krivi rezultati prilikom provjere spuštenosti glave. Sustav bi detektirao spuštenost glave, a zapravo se vozač samo udaljio od kamere. Kako bi se izbjegla ovakva greška, radi se dodatna provjera koja uspoređuje početnu površinu s trenutnom. Ukoliko je površina veća ili manja od određenog praga usporedno s početnom, površina glave i *y* koordinate se ažuriraju na trenutnu veličinu i položaj.



(a) (b)

Sl. 3.10. *Prikaz kako se mijenja veličina glave i položaj koordinata: (a) vozač se nalazi bliže kameri, veća površina označenog lica (b) vozač se udaljio od kamere, manja površina označenog lica.*

3.3.5. Detekcija i provjera profila lica

Ako algoritam ne može detektirati prednju stranu licu, pokušat će detektirati profil lica. Ovaj dio algoritma radi tako da kada se detektira profil lica, pali se brojač i broji koliko dugo vozač gleda u stranu. Ako brojač pređe određeni prag, aktivira se alarm za upozorenje koji govori vozaču da vrati pogled prema naprijed. Budući da detektor može označiti samo jedan profil lica, za svaku stranu potreban je po jedan detektor. Kako bi se smanjilo korištenje više nepotrebnih detektora, korišten je detektor samo za detekciju lijevog profil lica. Problem za desnu stranu je riješen tako da ukoliko algoritam nije detektirao lijevi profil, pozvat će ranije spomenutu funkciju koja će

okrenuti sliku oko y osi, odnosno postići će se efekt zrcala. S ovim pristupim moguće je detektirati i lijevu i desnu stranu lica korištenjem samo jednog detektora. Na slici 3.11. prikazana je detekcija lica okrenutog u lijevu i desnu stranu.



(a) (b)

Sl. 3.11. *Prikaz detekcije kada je glava okrenuta (a) u desnu stranu (b) u lijevu stranu*

3.4. Treniranje vlastitog detektora

OpenCV daje mogućnost treniranja vlastitog detektora. Ukoliko se želi detektirati određeni dio lica ili neki drugi objekt, moguće je istrenirati vlastiti detektor. U nastavku je dan kratki opis kako to učiniti.

Treniranje se sastoji od nekoliko faza: prikupljanje podataka za treniranje, priprema podataka i izvođenje stvarnog modela treniranja. Kod prikupljanja podataka potrebno je skupiti set pozitivnih slika i set negativnih slika. Pozitivne slike su one koje sadrže objekt koji se želi detektirati, dok negativne slike sadrže bilo koje druge objekte koji nisu traženi objekt. Pozitivni uzorci kreiraju se pomoću posebne *OpenCV* aplikacije. Aplikacija omogućava dva načina kreiranja pozitivnih uzoraka. Prvi način je generiranje pozitivnih slika na temelju jedne slike. Ovakav pristup radi tako da uzme jedan objekt i na temelju njega generira velik set pozitivnih uzoraka traženog objekta. To radi tako da nasumično rotira objekt, mijenja intenzitet slike te ga postavlja na proizvoljnu pozadinu. Količinu i raspon nasumičnosti moguće je kontrolirati. Drugi način je da korisnik sam skupi pozitivne uzorke te koristeći alat za rezanje označi željeni objekt, promjeni veličinu i spremi ih u binarni format. Ako se želi istrenirati jak model, potrebno je uzeti uzorke koji pokrivaju različite varijacije u kojima se objekt može nalaziti. Na primjer, kod kreiranja detektora za lice, bilo bi dobro razmotriti različite rase, dobne skupine, emocije itd. Također pozitivni uzorci se

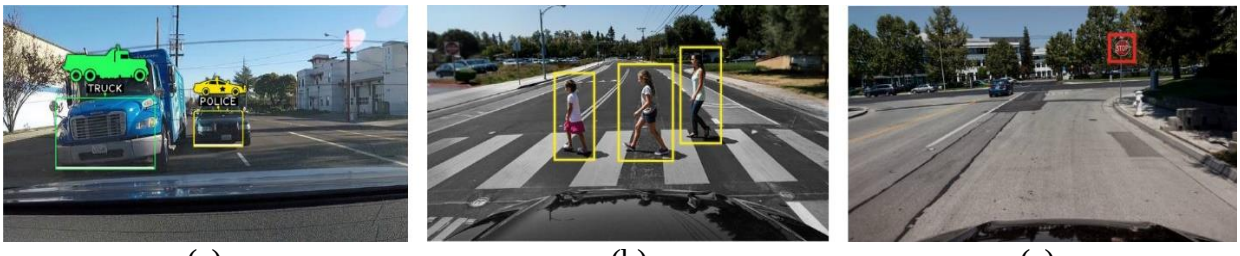
mogu dobiti i iz kolekcije prethodno označenih slika. Gdje se imena slika, anotacije i brojevi koji opisuju koordinate objekata nalaze u tekstualnoj datoteci. Svaka linija predstavlja jednu sliku.

Idući korak je treniranje kaskade klasifikatora koja je bazirana na prethodno priređenim pozitivnim i negativnim slikama. Nakon završetka treniranja kaskada će biti spremljena u datoteku cascade.*xml* i biti spremna za korištenje [13] .

3.5. Implementacija algoritma na realnoj platformi za pomoć vozaču pri vožnji

Napredni sustavi za pomoć u vožnji ili ADAS (engl. *advanced driver assistance systems*), su inteligentni sigurnosni sustavi koji pomažu vozaču u vožnji. Razvijeni su kako bi smanjili ljudsku pogrešku, automatizirali, prilagodili i poboljšali sustave vozila za sigurnost i bolju vožnju. Napredni sustavi mogu detektirati objekte, upozoriti vozača na opasne uvjete ili prijeteće opasnosti i automatski zaustaviti vozilo [14]. ADAS se oslanja na podatke iz više izvora kao što su: senzori, radar, LiDAR, kamere i računalni vid. Neke od ADAS aplikacija uključuju: pomoć pri noćnoj vožnji, otkrivanje pospanosti vozača, prepoznavanje znakova i pješaka, upozorenje prilikom prelaska u drugu traku, otkrivanje mrtvog kuta itd [15]. Neki od vodećih poduzeća koje se bave razvojem ADAS platformi su: Continental AG, Bosch, NVIDIA, Texas Instruments, BlackBerry QNX, Green Hills Software, Mobileye i dr.

NVIDIA je predstavila svoju *The NVIDIA DRIVE* platformu koja koristi duboko učenje i softverske biblioteke uz pomoć kojih programeri mogu izgraditi aplikacije koje iskorištavaju računalno intenzivne algoritme za otkrivanje objekata, lokalizaciju karata i planiranje puta. S *NVIDIA AI* rješenjem, neke od stvari koje ADAS u vozilu može razlikovati su kamion od policijskog auta, može identificirat biciklista na biciklističkoj stazi te prisutne pješake [16]. Primjer nekih mogućnosti je prikazan na slici 3.12.



(a) (b) (c)

Sl. 3.12. *NVIDIA nudi mogućnost (a) razlikovanja vozila, (b) detekciju pješaka (c) prepoznavanje znakova za ograničenje brzine* [16]

QNX platforma za ADAS je dobitnik nagrade za najbolju aktivnu sigurnost 2017 [17]. QNX platforma može se koristiti u širokom rasponu ADAS i automatiziranih aplikacija za vožnju, od četiri kamere za pogled iz ptičje perspektive, do aktivnih sigurnosnih sustava kao što je naglo kočenje te do potpuno autonomnog sustava vožnje. Neke od značajki platforme:

- Implementacija za 4 kamere za pogled odozgo, 1 kamera za ADAS, središnji senzor za ulaz s više kamera
- Prikupljanja podataka sa senzora: kamera, radar, LiDAR, GPS
- Vizualizacija podataka
- Mrežni dodaci za pružanje podataka sa senzora preko automobilske mreže
- Integrirane biblioteke otvornog koda kao što su: *OpenCV*, SOME/IP, Ceres i dr.

Za potrebe ovog diplomskog rada korištena je ADAS ploča ALPHA [18]. ALPHA ploča se temelji na TDA2x *Texas Instruments* sustavu u čipu (engl. *system on chip*). Implementira osnovne i napredne sustave upozorenja, aktivne upravljačke sustave i polu-autonomne operacije. Skup ciljanih aplikacija koje pokrivaju različite algoritme za pomoć vozaču su:

- procjena brzine vozila sa kamere iz retrovizora
- prepoznavanje prometnih znakova
- povezani pogled odozgo
- praćenje stanja vozača
- kamera za noćni vid
- parkiranje i pomoć u prometu
- različite značajke za povećanje udobnosti i iskustva u vožnji

Slika 3.13. prikazuje aplikacije u primjeni.



(a) (b)

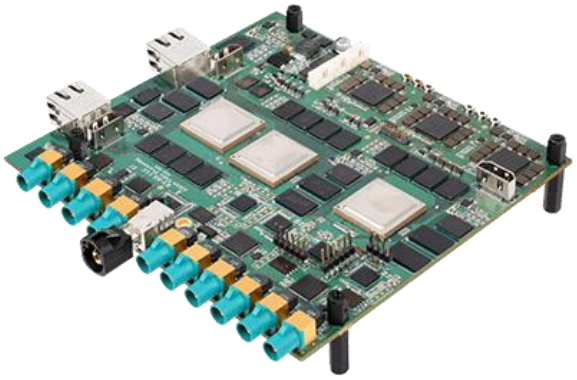
Sl. 3.13. *Primjer (a) procjene brzine iz retrovizora (b) povezani pogled odozgo za pomoć pri parkiranju* [18]

ALPHA koristi *VisionSDK* koji predstavlja više-procesorski komplet za razvoj softvera na ploči. Uz njega dolaze i neki postojeći slučajevi korištenja kao što su: detekcija i klasifikacija prometnih znakova, povezani pogled odozgo i dr.

Slijede hardverske specifikacije ALPHA ploče:

- Procesori: 3 x TDA2x SoC (2x ARM Cortex A15, 2x ARM Cortex M4, 2x C66x (DSP), 4x (EVE))
- Memorija: 3 x 1.5 GB DDR
- Podrška do 10 kamera
- Svaki SoC ima HDMI izlaz, DCAN, Ethernet, JTAG, UART sučelje i utor za Micro SD karticu
- Podrška za AVB Ethernet

Na slici 3.14. prikazan je izgled ALPHA ploče.



Sl. 3.14. *ALPHA ploča* [18]

Nakon dovršetka algoritma bilo je potrebno implementirati ga na ALPHA ploču i testirati njegovu učinkovitost. Ključni dio za rad algoritma su istrenirani detektori koji se nalaze u *xml* datotekama i pomoćna biblioteka *OpenCV*. ALPHA ploča ima podršku *OpenCV* verzije 3.1. Nakon pokretanja postojećeg rješenja za detekciju rubova i testiranja osnovnih *OpenCV* funkcija, rezultati rada u toj verziji nisu bili zadovoljavajući jer i jednostavna funkcija za zrcaljenje slike je uzimala previše procesorske snage. Iz tog razloga je odlučeno da se koristi *OpenCV* verzija 1.0., koja je napisana u C jeziku i da se koriste samo funkcije koje su potrebne za rad algoritma. Nakon spuštanja dijela *OpenCV* koda na ploču, koji učitava i nakon toga parsira *xml* datoteku, došlo je do problema.

Problem koji se događao tijekom parsiranja je da bi program napunio stog i zamrznuo rad ploče. Za rješavanje problema kontaktirana je *Texas Instruments* podrška, koja nije znala dati odgovor na dobiveni problem. Nakon više-mjesečnog testiranja i neuspješnosti otklanjanja problema u funkciji koja je ključna za učitavanje detektora te zbog ograničenja ploče, rad algoritma nije testiran na ALPHA ploči, zbog samih ograničenja hardvera i softvera ploče opisanih ranije.

4. TESTIRANJE UČINKOVITOSTI NOVOG ALGORITMA ZA PRAĆENJE STANJA VOZAČA

Za provjeru učinkovitosti algoritma, testiran je u okruženju u kojem bi i trebao raditi, a to je automobil. Kako bi se ispitali svi slučajevi kao što je spuštenost glave, detektiranja kada vozač žmiri te gleda li u stranu predugo, bilo je potrebno snimiti vlastite video zapise. Pri snimanju kamera je postavljena ispred vozača. Snimanje je izvršeno s prednjom kamerom Samsung Galaxy S6 uređaja, čija kamera ima razlučivost od 5 MP. Video zapisi su snimani u rezoluciji 1280x720 elementa slike, nakon toga su skalirani na 640x480 elementa slike. Video zapisi su snimani tijekom dana u različitim svjetlosnim uvjetima. Algoritam nije testiran po noći, zbog samog načina rada algoritma za detekciju koji ne bi radio u takvim uvjetima.

Algoritam je testiran na ukupno osam video zapisa. Tablica 4.1. prikazuje koliko je puta algoritam uspješno detektirao određenu radnju u odnosu na koliko se puta ta radnja dogodila. U drugom stupcu nalaze se rezultati mjerenja koji se odnose na pogled vozača u stranu, stupac sadrži mjerenja okrenutosti glave u lijevu i desnu stranu zajedno. Četvrti stupac prikazuje koliko puta je vozač držao oči zatvorene duže od određenog praga. Predzadnji stupac sadrži rezultate mjerenja koliko je puta vozač spustio glavu u odnosu na normalan položaj glave.

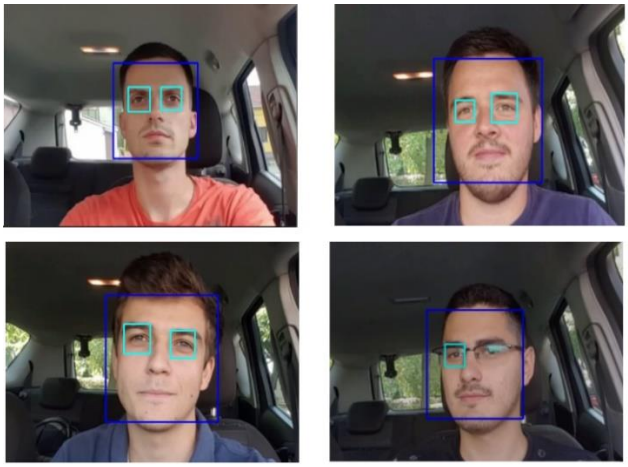
Tab. 4.1. *Rezultati testiranja algoritma za praćenje stanja vozača*

#VIDEO	Točno detektiran pogled u stranu / ukupan broj pogleda u stranu (lijevo ili desno)	Uspješnost pri detekciji pogleda u stranu	Točno detektirano da vozač žmiri / ukupan broj puta vozač zažmirio	Uspješnost pri detekciji žmirenja vozača	Točno detektirano spuštanje glave / ukupan broj puta glava spuštena	Uspješnost pri detekciji spuštenosti glave
Video 1	**4 / 4**	100,0%	**2 / 3**	66,7%	**1 / 2**	50,0%
Video 2	**4 / 4**	100,0%	**2 / 2**	100,0%	**3 / 3**	100,0%
Video 3	**3 / 3**	100,0%	**1 / 1**	100,0%	**1 / 1**	100,0%
Video 4	**1 / 2**	50,0%	**1 / 1**	100,0%	**1 / 1**	100,0%
Video 5	**1 / 2**	50,0%	**0 / 2**	0,0%	**0 / 2**	0,0%
Video 6	**2 / 4**	50,0%	**1 / 2**	50,0%	**0 / 2**	0,0%
Video 7	**2 / 2**	100,0%	**1 / 1**	100,0%	**1 / 1**	100,0%
Video 8	**0 / 2**	0,00%	**2 / 2**	100,0%	**1 / 2**	50,0%
Ukupno:	**17 / 23**		**10 / 14**		**8 / 14**	
Uspješnost:	**73.91%**		**71.43%**		**57.14%**	

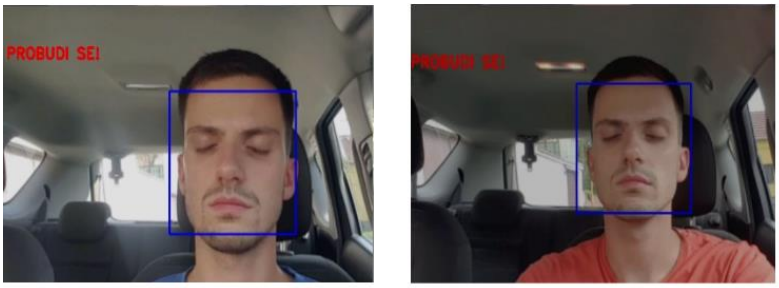
Na slikama 4.1., 4.2. i 4.3. prikazani su neki od uspješnih slučajeva rada algoritma pri testiranju u različitim svjetlosnim uvjetima.



Sl. 4.1. *Uspješna detekcija pri okretanju glave u stranu*



Sl. 4.2. *Uspješna detekcija lica i očiju*



Sl. 4.3. *Uspješna detekcija vozača koji ima predugo zatvorene oči*

Nakon provedenog testiranja i dobivenih rezultata zaključeno je da na točnost rezultata najviše utječu parametri same funkcije koja obavlja detekciju. Drugi važan faktor je razina svjetlosti. Kao što je vidljivo, video zapisi 2, 3 i 7 postigli su najbolje rezultate. Algoritam je uspješno detektirao odrađene radnje vozača. Razlog tomu je uz parametre detektora i „idealna“ razina svjetlosti, koja nije bila ni prejaka ni preslaba. Svjetlost je utjecala i na samu detekciju, ukoliko bi jaka svjetlost

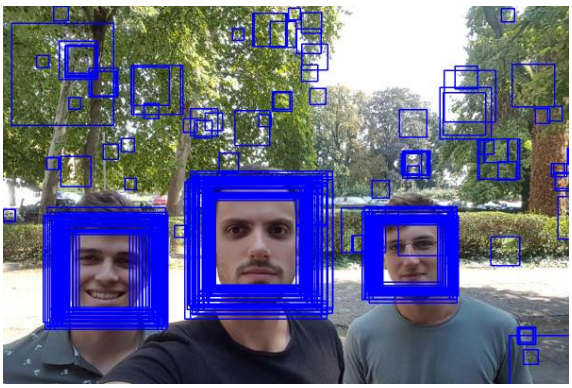
obasjala vozačevu glavu, zbog svog načina rada, detektor ga ne bi uspješno označio. Zbog velike važnosti za rad algoritma, korišteni parametri za svaki detektor navedeni su u nastavku.

Korišteni parametri u funkciji *detectMultiScale*:

- Lice
  - *minimalni broj susjeda*: 15
  - *faktor skaliranja*: 1.2
- Profil lica
  - *minimalni broj susjeda*: 1.3
  - *faktor skaliranja*: 3
- Oči
  - *minimalni broj susjeda*: 20
  - *faktor skaliranja*: 1.3

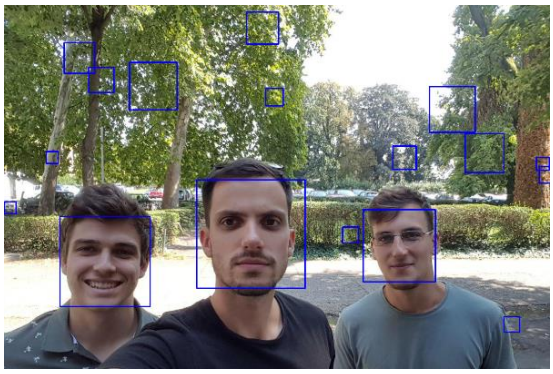
Kako bi dobili bolje razumijevanje rezultata, prvo je potrebno razumjeti kako to parametri utječu na rad algoritma. Najveći utjecaj na detekciju ima parametar minimalni broj susjeda (engl. *minNeighbors*), koji je objašnjen u nastavku.

Kaskada klasifikatora radi na principu klizećeg pod-prozora. Primjenjivanjem klizećeg pod-prozora po slici, pod-prozor traži objekt te prolazi kroz cijelu sliku. Nakon toga promjeni veličinu te ponovno traži željeni objekt i prolazi kroz sliku. Taj postupak se ponavlja dokle god se veličina prozora više ne može promijeniti. Svakom iteracijom pozitivni izlazi odnosno detekcije s Haarove kaskade su spremljeni. Nakon takve primjene gdje se svakom iteracijom promjeni veličina i ponovno prođe kroz sliku, detektira se puno lažno pozitivnih lica. Na slici 4.4. je prikazana detekcija kada je parametar *minimalni broj susjeda* postavljen na 0.



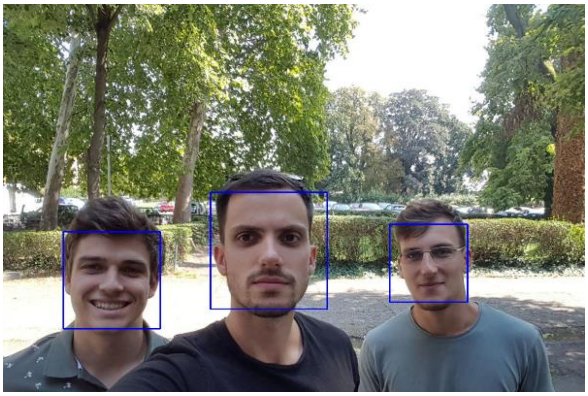
Sl. 4.4. *Detekcija uz parametar minimalni broj susjeda = 0*

Kako bi se uklonili lažni pozitivni rezultati i dobila točna detekcija lica, primjenjuje se koncept susjeda. Kao što sama riječ govori, ako je detektirani objekt u susjedstvu drugih prozora, onda je u redu. S toga ovaj broj određuje koliko je potrebno susjeda da bi se detektiralo lice. Slika 4.5. prikazuje kada je *minimalni broj susjeda* 1.



Sl. 4.5. *Detekcija uz parametar minimalni broj susjeda = 1*

Povećavanjem broja susjeda moguće je eliminirati lažne pozitive, ali prevelikim povećanjem tog broja mogu se izgubiti i točno detektirana lica. U slučaju ove slike, stavljanjem *minimalnog broja susjeda* na tri dobiva se točan rezultat.

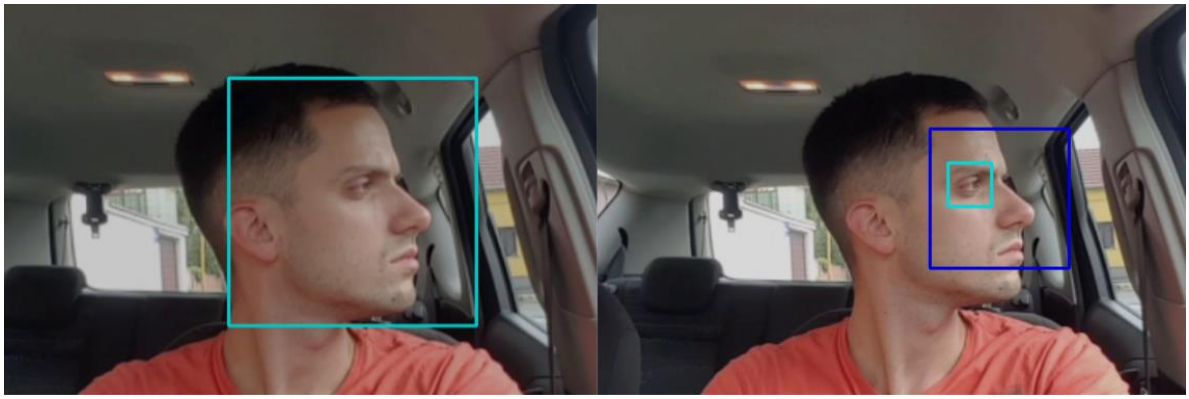


Sl. 4.6. *Detekcija uz parametar minimalni broj susjeda = 3*

Razlog zbog kojeg je *minimalni broj susjeda* kod detekcije lica i očiju u algoritmu postavljen na tako veliki broj je da bi se osigurala veća preciznost te izbjeglo pogrešno označavanje. Čim vozač okrene glavu u stranu, prednja detekcija lica se ne bi trebala događati.

Zbog toga pogled vozača u stranu ima najveću točnost jer ako *bi minimalni broj susjeda* bio manji, kod detekcije lica dolazilo bi do detekcije koja se ne bi trebala događati. Ovakav primjer se događao i kada je broj susjeda bio postavljen na veće vrijednosti, ali takvi slučajevi su bili rijetki. Primjer je slika 4.7. koja prikazuje opisanu situaciju. U ovoj situaciji svijetlo plavi pravokutnik je

ispravna detekcija. On se aktivira kada vozač gleda u stranu, ali već idući okvir algoritam aktivira detektor za prednju stranu lica, te ga označuje. Kako bi se izbjegle takve situacije „pogrešne detekcije“ broj susjeda za detekciju prednjeg dijela lica je postavljen na 15.



(a) (b)

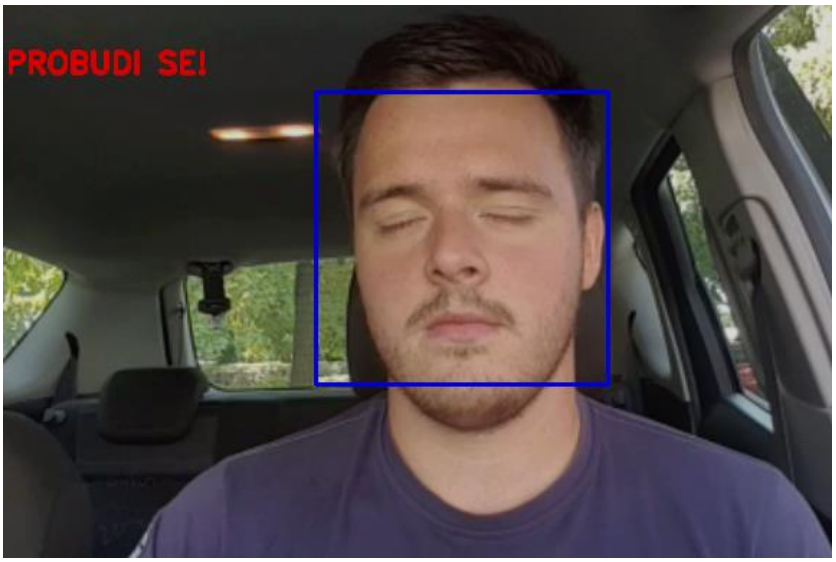
Sl. 4.7. *Prikaz (a) točne i (b) pogrešne detekcije kod okrenute glave (aktivan detektor za lice i oči)*

Kao što je ranije objašnjeno, postavljanjem *minimalnog broja susjeda* na velike vrijednosti događa se da algoritam u nekim situacijama neće detektirati lice. Najveći problem ovakve pogrešne detekcije je što svaki puta kada je lice krivo detektirano, poništava se brojač koji broji koliko dugo vozač gleda u stranu. Samim time neće upozoriti vozača ako predugo gleda u stranu.

Ovim pristupom dobivaju se točniji rezultati rada algoritma, ali slabija detekcija pri situacijama kada osvjetljenije nije idealno. Ovaj problem bi se mogao riješiti predobradom slike u takvim situacijama.

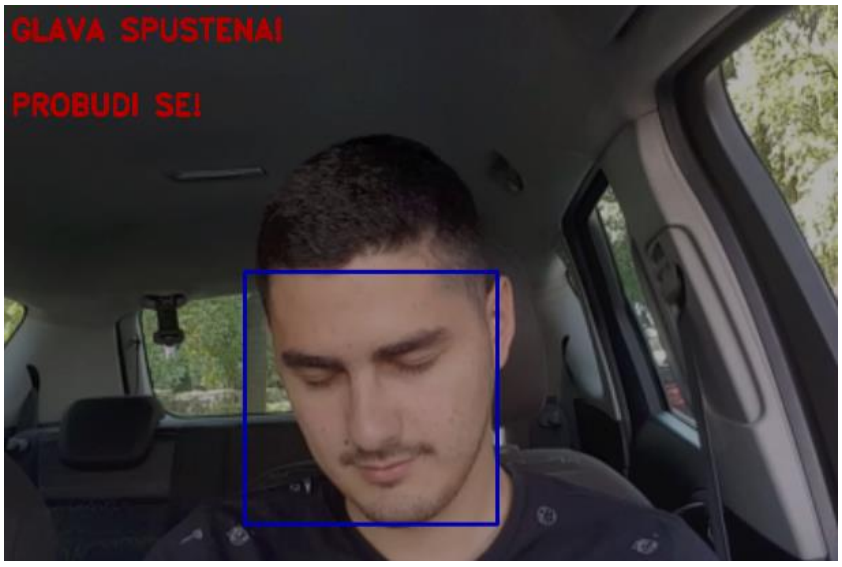
Detekcija spavanja ovisi o detekciji očiju koja ovisi o tome je li lice detektirano ili ne. Detektor korišten za traženje očiju je treniran da označi samo otvoreno oko. Korištenjem takvog detektora kada vozač žmiri, oči ne bi trebale biti detektirane. Prilikom testiranja otkriveno je da detektor označi oči vozača i kada žmiri, samim time poništava brojač koji broji kako bi upozorio vozača da predugo žmiri. Kako bi se izbjegle takve situacije kao i kod detekcije lica, *minimalni broj susjeda* je postavljen na veću vrijednost.

Na jednom od video zapisa vozač je nosio dioptrijske naočale te je detektor mogao pratiti njegove oči, što ne mora biti slučaj za ostale slučajeve u kojima osoba nosi naočale, a koristi se isti detektor za oči. Za takve slučajeve postoji posebni detektor koji je istreniran da detektira oči i kada osoba nosi naočale. Slika 4.8. prikazuje uspješnu detekciju vozača koji žmiri.



Sl. 4.8. *Prikaz uspješne detekcije vozača koji žmiri*

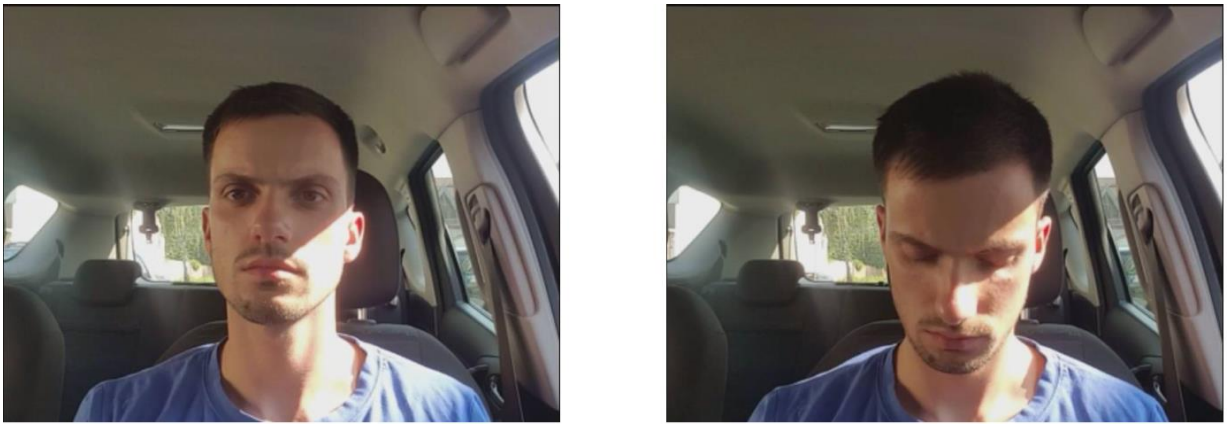
Najlošiji rezultati su dobiveni kod djela gdje je potrebno detektirati spuštenost glave. Jedan od razloga je taj, da u koliko vozač previše spusti glavu izgubi se fokus na lice i detektor jednostavno neće ništa detektirati. Kao rješenje ovog problema moguće je istrenirati novi detektor koji detektira tjeme glave odnosno kada je glava u spuštenom položaju. Drugi problem je kao i kod prethodnih slučajeva, zbog prevelikog *broja minimalnih susjeda*. Detekcija se gubi čim vozač malo spusti glavu ako uvjeti nisu idealni. Slika 4.9. prikazuje uspješnu detekciju i spuštenost glave.



Sl. 4.9. *Prikaz uspješne detekcije spuštenosti glave vozača*

Na slici 4.10. prikazani su rezultati iz video zapisa 5, za kojeg je algoritam postigao rezultate. Razlog ovako loših rezultata, kao što je i ranije spomenuto, su svjetlosni uvjeti, gdje svjetlost

obasjava dio glave vozača. Zbog toga i velikog *broja minimalnih susjeda* algoritam nije detektirao ono što je trebao.



(a) (b)

Sl. 4.10. *Prikaz neuspješne detekcije (a) lica i očiju (b) spuštenosti glave vozača*

Kada bi detektore zasebno testirali, njihova točnost bila bi puno veća od ovdje prikazanih rezultata. No zbog konstrukcije algoritma i postavljenih parametara dobiveni su navedeni rezultati. Također algoritam neće raditi u noćnim uvjetima zbog načina na koji Haarove značajke prepoznaju objekt. Za noćni rad mogla bi se koristiti infracrvena svjetlost, uz pomoć koje bi se pratile oči i na osnovu toga određivala razina umora vozača. Također moguće je i dodati predobradu slike u kojoj bi se posvijetlilo lice vozača pri noćnoj vožnji. Sam algoritam bi se mogao poboljšati dodatnom regulacijom parametra. Najlošiji rezultati koji su dobiveni prilikom određivanja spuštenosti glave vozača mogli bi se poboljšati dodavanjem još jednog detektora koji bi detektirao glavu u spuštenom položaju.

5. ZAKLJUČAK

U ovom radu predstavljen je algoritam za praćenje stanja vozača pomoću kamere. Rad se temelji na algoritmu za raspoznavanje objekata koji su razvili Viola i Jones. Rješenje je uspješno razvijeno uz pomoć *OpenCV* biblioteke, koja nudi mogućnost treniranja vlastitog detektora. Zbog ograničenja ALPHA ploče, implementiranje algoritma na realnoj platformi nije bilo moguće. Nakon provedenog testiranja algoritma, rezultati su zadovoljavajući, ali ne dovoljno dobri za komercijalnu upotrebu. Algoritam je napravljen da radi samo po danu, dok bi se za noćni način mogla koristiti infracrvena svjetlost, uz pomoć koje bi se pratile oči vozača. Algoritam je testiran na malom uzorku i mogao bi se poboljšati dodavanjem još jednog detektora pri detekcije spuštanja glave. Kako bi se dobili točniji rezultati, potrebno je odraditi testiranje na većem uzorku i u različitim uvjetima.

LITERATURA

[1] World Health Organization, Global status report on road safety, 2009, dostupno na: http://www.who.int/violence_injury_prevention/road_safety_status/2009/en/ [10.8.2018]

[2] The National Sleep Foundation, Sleep in America™ Poll, 2009, dostupno na: https://sleepfoundation.org/sites/default/files/2009%20POLL%20HIGHLIGHTS.pdf [10.8.2018]

[3] W. Vanlaar, H. M. Simpson, D. Mayhew, and R. Robertson, „Fatigued and drowsy driving: attitudes, concerns and practices of Ontario drivers“, Traffic Injury Research Foundation, 2007, dostupno na: http://tirf.ca/publications/fatigued-drowsy-driving-attitudes-concerns-practices-ontario-drivers/ [15.8.2018]

[4] S. Arimitsu, K. Sasaki, H. Hosaka, M. Itoh, K. Ishida, and A. Ito, „Vibration stimulus of seat belt motor retractor for keeping drivers awake,” in Proc. 2nd IEEE/ASME Int. Conf. Mechatronic and Embedded Systems and Applications, kolovoz, 2006, pp. 1–6. [15.8.2018]

[5] X. Luo, R. Hu, T. Fan, „The driver fatigue monitoring system based on face recognition technology“, IEEE, 9 – 11 lipanj, 2013 [15.8.2018]

[6] M. J. Flores, J. M. Armingol and A. Escalera, „Real-time warning system for driver drowsiness detection Using Visual Information”, Journal of Intelligent & Robotic Systems, kolovoz, 2010 [15.8.2018]

[7] A. Sahayadhas, K. Sundaraj and M. Murugappan, „Drowsiness detection during different times of day using multiple feature”, Australasian Physical & Engineering Sciences in Medicine, Lipanj 2013. [15.8.2018]

[8] X. Tang, P. Zhou, P. Wang, „Real-Time Image-based Driver Fatigue Detection and Monitoring System for Monitoring Driver Vigilance“, IEEE, 27 – 29 srpanj, 2016 [15.8.2018]

[9] I. Park, J. Ahn, H. Byun, „Efficient Measurement of the Eye Blinking by Using Decision Function for Intelligent Vehicles“ , Computational Science – ICCS, 2007 [15.8.2018]

[10] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features”, In Conference on Computer Vision and Pattern Recognition, 2001 [15.8.2018]

[11] OpenCV, dostupno na: https://opencv.org/about.html [25.8.2018]

[12] Github, openCV detektori, dostupno na: https://github.com/opencv/opencv/tree/master/data/haarcascades [25.8.2018]

[13] OpenCV, treniranje vlastitog detektora, dostupno na: https://docs.opencv.org/3.4.1/dc/d88/tutorial_traincascade.html [30.8.2018]

[14] Cypress, Advanced Driver Assistance Systems (ADAS), dostupno na: http://www.cypress.com/solutions/advanced-driver-assistance-systems-adas [31.8.2018]

[15] Grand View Research, Advanced Driver Assistance Systems (ADAS) Market Size, Share & Trend Analysis Report By Solution (Adaptive Cruise Control, Blind Spot Detection), By Component, By Vehicle, And Segment Forecasts, 2018 – 2025, veljača 2018, dostupno na: https://www.grandviewresearch.com/press-release/global-advanced-driver-assistance-systems-adas-market [31.8.2018]

[16] NVIDIA, ADVANCED DRIVER ASSISTANCE SYSTEMS (ADAS), dostupno na: https://www.nvidia.com/en-us/self-driving-cars/adas/ [31.8.2018]

[17] BlackBerry QNX, Advanced driver assistance systems, dostupno na: http://blackberry.qnx.com/en/products/adas/index#benefits [31.8.2018]

[18] Automotive Machine Vision ALPHA reference bord, dostupno na: http://www.rt-rk.com/download/rt-rk_ALPHA_ADAS_board.pdf [31.8.2018]

SAŽETAK

U diplomskom radu napravljen je algoritam za praćenje stanja vozača pomoću kamere u vozilu. Pri izradi korištena je *OpenCV* biblioteka te Viola - Jones algoritam za detekciju objekata. Algoritam detektira vozačevu glavu u svakom okviru i koristi tri detektora; detektor za oči, profil i prednju stranu lica. Nakon toga algoritam prati da li vozač ima zatvorene oči, gleda li u stranu te ima li glavu spuštenu predugo. Rješenje je uspješno testirano u različitim svjetlosnim uvjetima. Rezultati su pokazali da za određene svjetlosne uvjete algoritam postiže visoke performanse, dok bi za neke svjetlosne uvjete bila potrebna nadogradnja kako bi se dobili točniji rezultati.

Ključne riječi: detekcija umora, openCV, kompjuterski vid, Haarove kaskade, detekcija objekata

DRIVER MONITORING USING THE CAMERA IN THE VEHICLE

ABSTRACT

In the graduate thesis, a driver tracking algorithm was created using the camera in the vehicle. In development process *OpenCV* library and Viola - Jones algorithm was used for object detection. Algorithm detects the driver's head in each frame and uses three detectors; eye, profile and front face detector. After that, the algorithm tracks whether the driver has closed eyes, looking sideways, and whether the head has been down for too long. The solution has been successfully tested under different light conditions. The results show that for some light conditions the algorithm achieves high efficiency, while some light conditions require upgrading to get more accurate results.

Keywords: fatigue detection, OpenCV, computer vision, Haar cascades, object detection

ŽIVOTOPIS

Luka Mašanović rođen je 15.4.1994. g. u Osijeku. Nakon završene osnovne škole Frana Krste Frankopana u Osijeku, upisuje Elektrotehničku i prometnu školu Osijek, smjer tehničar za računalstvo u Osijeku te maturira 2013.g. Iste godine upisuje sveučilišni studij na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija, smjer računarstvo. Nakon završenog preddiplomskog studija računarstva 2016. godine upisuje diplomski studij - Informacijske i podatkovne znanosti.

Potpis:

_____

PRILOZI

Prilog P.3.1. - Programski kod algoritma za detekciju stanja vozača.

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <unistd.h>
#include <iostream>
#include <stdio.h>
#include <thread>

using namespace std;
using namespace cv;

#define EYE_CLOSED_FREAMES 40
#define HEAD_DOWN_FRAMES 40
#define HEAD_TURNED_FRAMES 45

#define FACE_SCALE_PARAM 1.2
#define FACE_MIN_NEIGHBOURS_PARAM 15

#define EYE_SCALE_PARAM 1.3
#define EYE_MIN_NEIGHBOURS_PARAM 20

#define PROFILE_SCALE_PARAM 1.3
#define PROFILE_MIN_NEIGHBOURS_PARAM 3

//Function Headers
void detectAndDisplay(Mat frame);
void detectEyes(vector<Rect> eyes, Mat faceROIgray, Mat faceROIcolor);
void initStartingPoints();
void checkEyes(Mat frame);
void checkHeadSize();
void checkHeadDown(Mat frame);
void detectAndCheckProfileFace(vector<Rect> profileFace, Mat frameGray, Mat frame);

//Global variables
int initalCounter = 0;
int x, y, w, h, area = 0;

//for Eyes
int wakeUpFlag = 1;
int eyeClosedCounter = 0;

//for Head size
int headSize = 0;
int headSizeFrameCounter = 0;
int updatedHeadSize = 0;
int currentHeadPosition = 0;
int headPoistionCounter = 0;

//for Head profile
int headTurnedRightCounter = 0;
int headTurnedLeftCounter = 0;


String faceCascadeName = "haarcascade_frontalface_default.xml";
String eyesCascadeName = "haarcascade_eye.xml";
String profileCascadeName = "haarcascade_profileface.xml";

```

```
59. CascadeClassifier faceCascade;
60. CascadeClassifier eyesCascade;
61. CascadeClassifier profileCascade;
62.
63. string windowName = "Capture";
64.
65. /** main */
66. int main()
67. {
68.     VideoCapture capture(0);
69.
70.     Mat frame;
71.
72.     //-- 1. Load the cascades
73.     if (!faceCascade.load(faceCascadeName)) { printf("Error loading face cascade\n");
     return -1; };
74.     if (!eyesCascade.load(eyesCascadeName)) { printf("Error loading eyes cascade\n");
     return -1; };
75.     if (!profileCascade.load(profileCascadeName))   { printf("Error loading profile f
    ace cascade\n"); return -1; };
76.
77.    //-- 2. Read the video stream
78.     if (capture.isOpened())
79.     {
80.         while (true)
81.         {
82.             capture.read(frame);
83.             //-- 3. Apply the classifier to the frame
84.             if (!frame.empty())
85.             {
86.                 detectAndDisplay(frame);
87.             }
88.             else
89.             {
90.                 printf(" --(!) No captured frame -- Break!");
91.                 break;
92.             }
93.             int c = waitKey(10);
94.
95.             if ((char)c == 'c') { break; }
96.         }
97.    }
98.
99.    return 0;
100.       }
101.
102.        // Main function for detection
103.        void detectAndDisplay(Mat frame)
104.       {
105.           std::vector<Rect> faces;
106.           std::vector<Rect> profileFace;
107.           cv::Rect maxRect;
108.           Mat frameGray;
109.
110.           cvtColor(frame, frameGray, CV_BGR2GRAY);
111.
112.           // -- Detect faces
113.           faceCascade.detectMultiScale(frameGray, faces, FACE_SCALE_PARAM, FACE_MIN_
    NEIGHBOURS_PARAM);
114.
115.
116.
117.           // -- If face is detected if there is more than 0 faces
118.           if (faces.size() > 0)
119.           {
120.               headTurnedRightCounter = 0;
```

```
121.                headTurnedLeftCounter = 0;
122.
123.                wakeUpFlag = 1;
124.                //For all the faces detected find the biggest face.
125.                for (size_t i = 0; i < faces.size(); i++)
126.                {
127.                    x = faces[i].x;
128.                    y = faces[i].y;
129.                    w = faces[i].width;
130.                    h = faces[i].height;
131.
132.                    if (faces[i].area() > maxRect.area())
133.                    {
134.                        maxRect = faces[i];
135.                    }
136.                }
137.                // -
    - IF there is something in maxRect(face) draw rectangle around it.
138.                if (maxRect.x != 0)
139.                {
140.                    x = maxRect.x;
141.                    y = maxRect.y;
142.                    w = maxRect.width;
143.                    h = maxRect.height;
144.
145.                    rectangle(frame, Point(x, y), Point(x + w, y + h), Scalar(255, 0,
    0), 2);
146.
147.                    Mat faceROIgray = frameGray(maxRect);
148.                    Mat faceROIcolor = frame(maxRect);
149.
150.                    std::vector<Rect> eyes;
151.
152.                    //-
    - IF face is detected set starting points for head position and head size.
153.                    initStartingPoints();
154.
155.                    //-- In each face, detect eyes
156.                    detectEyes(eyes, faceROIgray, faceROIcolor);
157.                }
158.                //-
    - After inital detecting for face and eyes, check if head size is changed. If driver
    moves back or forward.
159.                updatedHeadSize = w * h;
160.
161.                // -- Check if eyes are closed. And give warning to user.
162.                checkEyes(frame);
163.
164.                // -
    - Checking Head size. IF updated size if bigger or smaller than inital position. Chan
    ge head size treshold.
165.                checkHeadSize();
166.
167.                // -- Checking if y - axis is changed. IS head down?
168.                checkHeadDown(frame);
169.            }
170.            else
171.            {
172.                // -
    - IF head is turned to the right or left side. Detect and check profile face.
173.                detectAndCheckProfileFace(profileFace, frameGray, frame);
174.            }
175.          imshow(windowName, frame);
176.        }
177.
178.        void initStartingPoints()
```

```
179.        {
180.            if (initalCounter == 0)
181.            {
182.                headSize = w * h;
183.                currentHeadPosition = y;
184.                initalCounter = 1;
185.                printf("Starting head pose y axis: %d\n", y);
186.                printf("Pocetna povrsina: %d\n", headSize);
187.                sleep(5);
188.            }
189.        }
190.
191.        void detectEyes(vector<Rect> eyes, Mat faceROIgray, Mat faceROIcolor)
192.        {
193.            eyesCascade.detectMultiScale(faceROIgray, eyes, EYE_SCALE_PARAM, EYE_MIN_N
    EIGHBOURS_PARAM);
194.
195.            for (size_t j = 0; j < eyes.size(); j++)
196.            {
197.                wakeUpFlag = 0;
198.
199.                int ex = eyes[j].x;
200.                int ey = eyes[j].y;
201.                int ew = eyes[j].width;
202.                int eh = eyes[j].height;
203.
204.                rectangle(faceROIcolor, Point(ex, ey), Point(ex + ew, ey + eh), Scalar
    (255, 255, 0), 2);
205.            }
206.        }
207.
208.        void checkEyes(Mat frame)
209.        {
210.            if (wakeUpFlag == 1)
211.            {
212.                eyeClosedCounter += 1;
213.
214.                if (eyeClosedCounter >= EYE_CLOSED_FREAMES)
215.                {
216.                    putText(frame, "PROBUDI SE!", Point(10,85), 2, 0.7, Scalar(0, 0, 2
    55), 2);
217.                }
218.            }
219.            else
220.            {
221.                eyeClosedCounter = 0;
222.            }
223.        }
224.
225.
226.        void checkHeadSize()
227.        {
228.            if (updatedHeadSize > headSize * 1.3 || updatedHeadSize < headSize * 0.70

229.            {
230.                headSizeFrameCounter += 1;
231.                printf("Head size: %d\n", headSize);
232.                printf("Updated head size: %d\n", updatedHeadSize);
233.                if (headSizeFrameCounter > 10)
234.                {
235.                    if (updatedHeadSize > headSize * 1.3 || updatedHeadSize < headSize
     * 0.70)
236.                    {
237.
238.
239.                        headSize = w * h;
```

```
240.                         currentHeadPosition = y;
241.
242.                     }
243.                 }
244.             }
245.         }
246.
247.         void checkHeadDown(Mat frame)
248.         {
249.             if (currentHeadPosition * 1.2 < y)
250.             {
251.                 headPoistionCounter += 1;
252.
253.                 if (headPoistionCounter > HEAD_DOWN_FRAMES)
254.                 {
255.                     printf("Head pose y: %d\n", currentHeadPosition);
256.
257.                     if (currentHeadPosition * 1.2 < y)
258.                     {
259.                         putText(frame,"GLAVA SPUSTENA!", Point(10,30), 2, 0.7, Scalar(
     0, 0, 255 ), 2);
260.                     }
261.                 }
262.             }
263.             else
264.             {
265.                 headPoistionCounter = 0;
266.             }
267.         }
268.
269.         void detectAndCheckProfileFace(vector<Rect> profileFace, Mat frameGray, Mat fr
    ame)
270.         {
271.             profileCascade.detectMultiScale(frameGray, profileFace, PROFILE_SCALE_PARA
    M, PROFILE_MIN_NEIGHBOURS_PARAM);
272.
273.             // -
    - Checking for left side, if detecor works increment the counter and warn the driver
    if he is looking too long to one side.
274.             if (profileFace.size() > 0)
275.             {
276.                 headTurnedRightCounter = 0;
277.                 headTurnedLeftCounter += 1;
278.
279.                 for (size_t i = 0; i < profileFace.size(); i++)
280.                 {
281.                     int ex = profileFace[i].x;
282.                     int ey = profileFace[i].y;
283.                     int ew = profileFace[i].width;
284.                     int eh = profileFace[i].height;
285.
286.                     rectangle(frame, Point(ex, ey), Point(ex + ew, ey + eh), Scalar(25
    5, 255, 0), 2);
287.                 }
288.
289.                 if (headTurnedLeftCounter > HEAD_TURNED_FRAMES)
290.                 {
291.                     putText(frame, "OKRENI GLAVU U LIJEVO!", Point(300, 30), 2, 0.7, S
    calar(0, 0, 255), 2);
292.                 }
293.             }
294.             // -
    - Checking for the right side, same thing. Using same detector just flipping image.

295.             else
296.             {
```

```
297.                headTurnedLeftCounter = 0;
298.                headTurnedRightCounter += 1;
299.
300.                //Flip frame to use same detector but another side of face
301.                flip(frame, frame, 1);
302.                cvtColor(frame, frameGray, CV_BGR2GRAY);
303.
304.                profileCascade.detectMultiScale(frameGray, profileFace, PROFILE_SCALE_
    PARAM, PROFILE_MIN_NEIGHBOURS_PARAM);
305.
306.                for (size_t i = 0; i < profileFace.size(); i++)
307.                {
308.                    int ex = profileFace[i].x;
309.                    int ey = profileFace[i].y;
310.                    int ew = profileFace[i].width;
311.                    int eh = profileFace[i].height;
312.
313.                    rectangle(frame, Point(ex, ey), Point(ex + ew, ey + eh), Scalar(25
    5, 255, 0), 2);
314.                }
315.
316.                flip(frame, frame, 1);
317.
318.                if(headTurnedRightCounter > HEAD_TURNED_FRAMES && profileFace.size() >
     0)
319.                {
320.                    putText(frame, "OKRENI GLAVU U DESNO!", Point(300, 30), 2, 0.7, Sc
    alar(0, 0, 255), 2);
321.                }
322.            }
323.        }
```