

SVEUČILIŠTE U ZAGREBU  
FAKULTET ORGANIZACIJE I INFORMATIKE  
VARAŽDIN

Stjepan Petrović

**PRILAGODLJIV SUSTAV ZA SMANJENJE  
SVJETLOSNOG ZASLJEPLJIVANJA  
VOZAČA**

**ZAVRŠNI RAD**

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Stjepan Petrović

Matični broj: 0016150314

Studij: Informacijski i poslovni sustavi

**PRILAGODLJIV SUSTAV ZA SMANJENJE SVJETLOSNOG  
ZASLJEPLJIVANJA VOZAČA**

**ZAVRŠNI RAD**

**Mentor :**

Doc. dr. sc. Boris Tomaš

Varaždin, rujan 2023.

*Stjepan Petrović*

**Izjava o izvornosti**

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrđio prihvaćanjem odredbi u sustavu FOI-radovi*

---

## Sažetak

Tema rada je izrada prilagodljivog sustava za smanjenje svjetlosnog zasljepljivanja vozača što predstavlja fizički koncept koji čini LCD matrica, dvije web kamere i laptop kao procesna jedinica. Izazov je bio spojiti četiri komponente (komponenta za pozicioniranje očiju vozača, komponenta za pozicioniranje zasljepljujućeg svjetla, komponenta za zaštitu od zasljepljujućeg svjetla i komponenta procesne jedinice zajedno sa ostalim hardverom), od kojih svaka ima svoju važnost i način pristupa, u jedan funkcionalan sustav čiji je koncept realiziran u ovome radu i koji odgovara na pitanje: kako preko kamera prepoznati izvor zasljepljujućeg svjetla i zaštiti oči vozača na način da se preko LCD matrice sprijeći prolazak zasljepljujućeg svjetla do očiju vozača. Kako bi se izradio odgovarajući sustav korištena je biblioteka OpenCV (engl. *Open Source Computer Vision Library*) koja je kao projekt pokrenuta od strane Intel korporacije, a pruža softver za strojno učenje i računalni vid u realnom vremenu i korišten je programski jezik Python. Programski kod i  $\text{\LaTeX}$  dokumentacija je verzionizirana na GitHub repozitoriju, kojem se može pristupiti preko poveznice: <https://github.com/StjepanPetrovic/Prilagodljiv-sustav-za-smanjenje-svjetlosnog-zasljepljivanja-vozaca>

**Ključne riječi:** računalni vid; OpenCV; vozilo; zasljepljivanje; LCD; Python;

# Sadržaj

<b>1. Uvod</b>	1
1.1. Definicija problema	1
1.2. Motivacija za rad	2
1.3. Metode i tehnike rada	3
<b>2. Pregled literature</b>	4
<b>3. Izrada sustava</b>	5
3.1. Komponenta procesne jedinice i hardver	6
3.1.1. Hardver	6
3.1.2. OpenCV-Python biblioteka	7
3.1.3. Redovi kao struktura podataka za spremanje okvira	8
3.1.4. Višedretvenost zbog raspodijele I/O zadataka	10
3.1.5. Kamere - čitanje okvira s video izvora	11
3.2. Komponenta za prepoznavanje i pozicioniranje izvora svjetla	14
3.3. Komponenta za prepoznavanje i pozicioniranje očiju vozača	17
3.4. Komponenta za polarizaciju LCD matrice kao reaktivne komponente	18
<b>4. Testiranje sustava</b>	19
<b>5. Zaključak</b>	20
<b>Popis literature</b>	22
<b>Popis slika</b>	23
<b>Popis programskega kodova</b>	24

# 1. Uvod

Ovim završnim radom obrađeni su teorijski koncepti na kojima se temelji rad, istražena je literatura, opisan je tijek izrade i konačan rezultat izrade **prilagodljivog sustava za smanjenje svjetlosnog zasljepljivanja vozača** te je provedeno testiranje sustava.

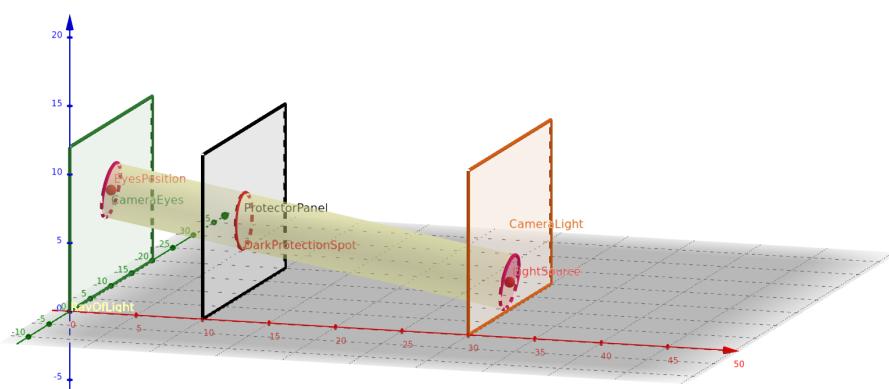
Programski kod i  $\text{\LaTeX}$  dokumentacija je verzionizirana na GitHub repozitoriju, kojem se može pristupiti preko poveznice: <https://github.com/StjepanPetrovic/Prilagodljiv-sustav-zasmanjenje-svjetlosnog-zasljepljivanja-vozaca>.

U nastavku rada za izraz „prilagodljiv sustav za smanjenje svjetlosnog zasljepljivanja vozača“ koristit će se skraćena inačica „**sustav protiv zasljepljivanja**“.

## 1.1. Definicija problema

Bilo je potrebno napraviti sustav koji u realnom vremenu prepoznaće izvor zasljepljujućeg svjetla te reagira na način da polarizira određeni dio reaktivne komponente (LCD matrice) koja bi se nalazila na vjetrobranskom staklu vozila te na taj način smanji jačinu zasljepljujućeg svjetla ispred vozača u vozilu.

Na slici 1 može se vidjeti da takav sustav treba imati ulazne uređaje pomoću kojih će vidjeti što se događa u okolini vozača. Za ulazne uređaje su uzete dvije web kamere čiji je sadržaj onoga što vide predstavljen kao narančasti i zeleni okvir na slici 1, a taj sadržaj će obrađivati istrenirani modeli za računalni vid iz biblioteke OpenCV te će tako procesna jedinica znati gdje se nalaze oči vozača i izvor svjetla koji su predstavljeni kao crveni krugovi odnosno baze valjka na narančastom i zelenom okviru na slici 1. Kada procesna jedinica to zna, potrebno je pomoću algoritma izračunati koji točno dio LCD matrice treba polarizirati/zatamniti, a taj dio koji treba polarizirati je prikazan na slici 1 kao crveni krug na crnom okviru odnosno intersekcija žutog plašta valjka, koji predstavlja svjetlost, sa crnim okvirom koji predstavlja reaktivnu komponentu (LCD matricu). Interaktivnom grafu sa slike 1 može se pristupiti preko linka: <https://www.geogebra.org/m/hvzfyjfz>.



Slika 1: Pojednostavljen prikaz sustava u trodimenzionalnom koordinatnom sustavu [autorski rad]

Razlog zbog čega je uzeta LCD matrica kao reaktivna komponenta koja će sprječavati zasljepljujuće svjetlo da dođe do očiju vozača je taj što može biti prozirna i moguće je gledati kroz nju, zbog čega neće smetati na vjetrobranskom staklu prilikom vožnje, a lako ju je moguće napraviti neprozirnom na način da se zaslon polarizira odnosno da se pikseli postave na crnu boju.

Sustav protiv zasljepljivanja se sastoji od četiri komponente koje su u radu obrađena:

- Procesna jedinica (laptop) i hardver (web kamere i LCD matrica),
- Komponenta za prepoznavanje i pozicioniranje izvora svjetla,
- Komponenta za prepoznavanje i pozicioniranje očiju vozača,
- Komponenta za polarizaciju LCD matrice kao reaktivne komponente.

Uz dodatna ulaganja i razvoj, ovaj fizički koncept može postati vrlo popularan i koristan proizvod svakom vozaču u vozilu jer će pružiti zaštitu u noćnoj vožnji od zasljepljujuće svjetlosti, koja usmjereni u oči vozača za vrlo kratak trenutak može ugroziti vozača. Najčešće su izvore svjetlosti duga svjetla na vozilu vozača koji zbog neopreznosti ne isključi duga svjetla u trenutku kada dolazi ususret drugom vozilu čiji će vozač zbog toga biti svjetlosno zasljepljen te na trenutak neće moći vidjeti kuda vozi što može loše utjecati na vozača. Zato je bilo potrebno napraviti sustav koji će:

- prepoznati i pozicionirati izvor svjetla te oči vozača koristeći kamere,
- kalibrirati komponente i uspješno ih povezati u jedan cjelovit funkcionalan sustav.

## 1.2. Motivacija za rad

Motivacija za odabir ove teme mi je bila misao da će se okušati u stvaranju sustava protiv zasljepljivanja za kojeg i u modernoj automobilskoj industriji još ne postoji izrađeno rješenje koje je optimalno za korištenje u realnim uvjetima – zbog čega sam gore i rekao da bi uz daljnja ulaganja i razvoj, fizički koncept koji je izrađen u svrhu ovog završnog rada mogao biti popularan. Postoji velik broj raspisanih patenata od strane najkonkurentnijih svjetskih proizvođača što ostavlja dojam da će skorija budućnost biti jako dinamična utrka za osvajanje tržišta proizvodom koji će, osim borbe sa svjetlosnim zasljepljenjem, donijeti i dodatne mogućnosti kao što je uvođenje proširene stvarnosti (engl. *Augmented Reality - AR*) na vjetrobransko staklo.

Velik je broj nesreća prouzrokovani svjetlosnim zasljepljenjem vozača, a još veći je broj vozila koji se svakim danom povećava na prometnicama širom svijeta, stoga moderna automobilска industrija sve više pokušava proizvesti automobile koji će imati ugrađen takav sustav za zaštitu vozača – što donosi velik značaj ovoj temi te poticaj za daljnje istraživanje i razvoj proizvoda koji će spriječiti povećanje broja prometnih nesreća prouzrokovanih svjetlosnim zasljepljenjem vozača.

### **1.3. Metode i tehnike rada**

U ovom poglavlju treba opisati koje će metode i tehnike biti korištene pri razradi teme, kako su provedene istraživačke aktivnosti, koji su programski alati ili aplikacije korišteni.

## **2. Pregled literature**

### 3. Izrada sustava

U ovom poglavlju će biti opisana izrada prilagodljivog sustava za smanjenje svjetlosnog zasljepljivanja vozača. Ovo će poglavlje biti podijeljeno na četiri potpoglavlja od kojih će svaki opisivati određenu komponentu budući da se sustav sastoji od četiri komponente:

- procesna jedinica (laptop) i hardver (web kamere i LCD matrica),
- komponenta za prepoznavanje i pozicioniranje izvora svjetla,
- komponenta za prepoznavanje i pozicioniranje očiju vozača,
- komponenta za polarizaciju LCD matrice kao reaktivne komponente.

Programski kod koji se bude prikazivao u radu može se pronaći na GitHub repozitoriju preko poveznice: <https://github.com/StjepanPetrovic/Prilagodljiv-sustav-za-smanjenje-svetlosnog-zasljepljivanja-vozaca>. Svi isječci programskog koda su uzeti iz jedne datoteke *main.py* te će se u isjećima programskog koda moći vidjeti i redni brojevi linija koda koji se odnose na redne brojeve linija koda iz datoteke *main.py*.

Ovim radom izrađen je koncept koji će objasniti i prikazati ideju za izradu ovakvog sustava, ali ovaj koncept nije spremjan za upotrebu u stvarnoj okolini. Ono što nije ovim radom obrađeno je navedeno u nastavku, to su neke od glavnih značajki koje bi sustav trebao ispunjavati kako bi pronašao svrhu u stvarnoj okolini:

- korištenje mikroprocesora koji će biti zadužen samo za obavljanje funkcionalnosti sustava,
- korištenje posebno izrađene prozirne LCD matrice ili korištenje drugog medija kao reaktivne komponente koja bi poslužila svrsi,
- korištenje posebnog modela i senzora za otkrivanje jačine svjetlosti,
- korištenje posebno istreniranog modela ili senzora koji će moći izračunati udaljenost zasljepljućeg svjetla i očiju od reaktivne komponente,
- korištenje algoritma koji će uzeti u obzir sve udaljenosti, nagib vjetrobranskog stakla i klasifikacije objekata od interesa kako bi što kvalitetnije izračunavao mjesto na kojemu treba zaustaviti svjetlost preko reaktivne komponente,
- spremnost na rad u noćnim uvjetima,
- velika količina testiranja sustava u raznovrsnoj i dinamičnoj okolini (posebno noćnoj okolini).

## 3.1. Komponenta procesne jedinice i hardver

Ovo poglavlje opisuje komponentu procesne jedinice kao komponentu koja čini temelj i softverski povezuje ostale komponente, a također opisuje i kako je sustav hardverski povezan.

### 3.1.1. Hardver

U ovom radu komponentu procesne jedinice predstavlja laptop na kojem će se izvršavati programski kod i čiji će procesor obrađivati ulazne informacije koje šalju kamere, a kamere su obične web kamere od kojih je jedna ugrađena u laptop, a druga je eksterna i priključena je preko USB priključka u laptop. Jedna kamera je namijenjena za snimanje okoline ispred vozača, a druga kamera je namijenjena za snimanje samog vozača.

Ispred vozača bi se nalazila LCD matrica koja će smanjiti i sprječiti zasljepljujuću svjetlost da dođe do očiju vozača. LCD matrica, zajedno sa elektronikom, za ovaj rad je izvađena iz običnog monitora te povezana preko HDMI priključka u laptop i ponaša se kao drugi zaslon laptopa. Na slici 2 i 3 može se vidjeti kako izgleda LCD matrica zajedno sa elektronikom i priključcima kada je izvađena iz monitora. Kao što se može vidjeti, LCD matrica kada se izvadi iz monitora je tamna te je potrebno imati jak izvor svjetla kako bi se vidjelo kroz nju dok je samostalno izvan monitora. TODO - iz toga razloga planiram postaviti LED svjetla iznad i ispod LCD matrice kako bih poboljšao njezinu providnost.



Slika 2: Prikaz LCD matrica s prednje strane kada je izvađena iz monitora [autorski rad]

TODO pronaći još jednu lcd matricu i pokušati s njom jer sam ovu pokvario :)



Slika 3: Prikaz LCD matrica sa zadnje strane kada je izvađena iz monitora [autorski rad]

### 3.1.2. OpenCV-Python biblioteka

Budući da je potrebno prepoznati i pronaći točne pozicije na kojima se nalaze objekti od interesa odnosno svjetlost i oči, potrebno je koristiti algoritme za računalni vid. Kako IBM navodi (engl. *International Business Machines Corporation - IBM*) [1], računalni vid je grana umjetne inteligencije (engl. *Artificial intelligence - AI*) koja omogućava računalima da pruže smislene informacije koje pronađu obradom slika, videa ili drugog vizualnog izvora te da reagiraju shodno toj informaciji. Zbog toga u ovom radu koristit će se biblioteka OpenCV za programski jezik Python.

OpenCV je biblioteka otvorenog koda (engl. *open source*) koja služi za rješavanja problema vezanih za računalni vid (engl. *computer vision*) i strojno učenje (engl. *machine learning*) te pruža često potrebnu infrastrukturu za aplikacije koje integriraju računalni vid [2]. OpenCV biblioteka podržava programske jezike C++, Python, Java, itd., i dostupna je na Windows, Linux, OS X, Andorid, iOS platformama. U radu će se koristiti OpenCV-Python biblioteka koja je Python API (engl. *Application Programming Interface - API*) za OpenCV biblioteku koja uzima najbolje kvalitete OpenCV C++ API-ja i Python programskog jezika [3].

Programski jezik Python je izabran zbog osobnih preferencija autora. Treba se uzeti u obzir da je Python sporiji programski jezik u odnosu na C++ koji je se također mogao koristiti u ovom radu, no moguće je imati i Python module koji će sadržavati C++ programski kod za procesorski intenzivne zadatke, a rezultat toga je da se izvršavanja Python programskog koda izvršava približno jednakom brzinom kao i brzina izvršavanja C++ programskog koda jer se C++ kod izvršava u pozadini te lakše je programirati u Python programskom jeziku nego u C++ programskom jeziku [3].

Kako bi se koristila biblioteka OpenCV sa programskim jezikom Python potrebno ju je prvo instalirati. U terminalu unesite komandu: `pip install opencv – python` i OpenCV biblioteka

će biti instalirana. Ako ju želite korisiti u *conda* radnom okruženju koje nudi dodatne pogodnosti slijediti upute za odgovarajući operativni sustav (OS) za:

- **Linux OS:** <https://youtu.be/gt0Mpj6FFzQ?si=QCLjiDrJcBCP5qV8>,
- **Windows OS:** <https://youtu.be/RfFiTozvOdQ?si=6jtyHU4YK9jsi6kv>,
- **MacOS:** <https://youtu.be/hZWgEPOVnuM?si=kD9CNUf4kNqyWBBK>,

te je potrebno postaviti *conda* radno okruženje: <https://www.jetbrains.com/help/pycharm/conda-support-creating-conda-virtual-environment.html>.

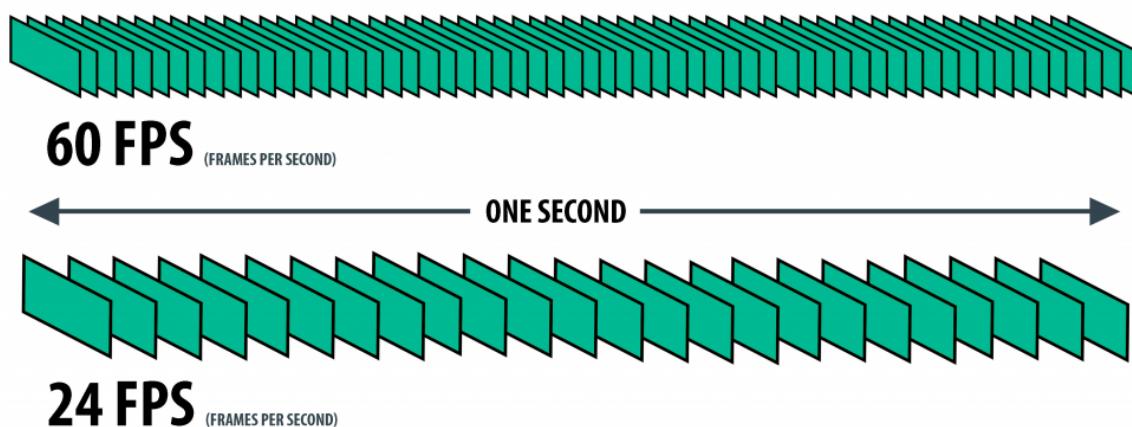
Nakon instalacije moguće je uključiti biblioteku pomoću sljedećeg programskog koda 1:

```
1 import cv2 as cv
```

Programski kod 1: Uključivanje biblioteke *OpenCV*

### 3.1.3. Redovi kao struktura podataka za spremanje okvira

Ono što algoritam za računalni vid uzima kao ulazni podatak je fotografija odnosno okvir (engl. *frame*) koji se dobiva od kamere koja cijelo vrijeme snima okolinu. Slika 4 objašnjava kako je skup okvira odnosno fotografija fotografiranih uzastopno u kratkom vremenskom periodu jednak videu te na taj način video i nastaje [4].

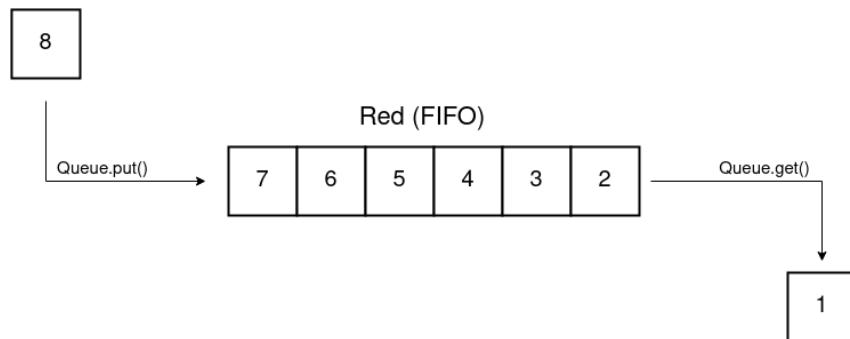


Slika 4: Prikaz uzastopnih fotografija/okvira koji čine video od jedne sekunde [4]

Budući da kamere cijelo vrijeme snimaju okolinu, one generiraju mnogo fotografiju u stvarnom vremenu od kojih će program uzimati po jednu fotografiju u određenom trenutku, analizirati ih i spremati ih u posebne strukture podataka za daljnju obradu. U ovom radu strukture podataka koje su uzete za ovu svrhu spremanja potrebnih informacija su redovi (engl. *Queues*) tipa FIFO - "prvi ušao, prvi izašao" (engl. *First In First Out - FIFO*).

Razlog zbog čega su odabrani redovi kao struktura podataka u koju će se spremati podaci je taj što redovi osiguravaju sigurno korištenje podataka između više dretava, a tip FIFO

zbog toga što je bitno da se prvo analizira okvir koji je najprije došao [5]. To će biti vrlo korisno budući da će ovaj program koristiti glavnu dretvu za čitanje okvira iz redova i njihovo prikazivanje, i drugu dretvu za dobijanje okvira pomoću kamere, analiziranje i njihovo spremanje u redove. Slika 5 slikovito prikaziva red kao strukturu podataka te prikaziva funkcije *get()* i *put()* klase *Queue* pomoću kojih se podaci dodavaju i uzimaju iz redova. Programski kod 2 prikazuje kako uključiti biblioteku i inicijalizirati redove.



Slika 5: Slikovit prikaz reda kao strukture podataka [autorski rad]

```

4 from queue import Queue

6 eyes_frames_queue = Queue()
7 light_frames_queue = Queue()

9 eyes_position_queue = Queue()
10 light_position_queue = Queue()
  
```

Programski kod 2: Uključivanje biblioteke *queue* i inicijaliziranje redova

Programskim kodom 2 inicijalizirani su *eyes\_frames\_queue* i *light\_frames\_queue* redovi koji služe za spremanje okvira koji se dobiju pomoću kamera i prikazivanje istih okvira nakon njihova analiziranja te još su inicijalizirani *eyes\_position\_queue* i *light\_position\_queue* redovi koji služe za spremanje koordinata za pozicije očiju i izvora svjetla koji se dobiju nakon analiziranja okvira i služe za kasnije računanje prilikom stvaranje sloja zaštite koji će se prikazivati na LCD matrici.

Okviri se prikazuju u posebno otvorenim prozorima na zaslonu laptopa i LCD matrice koje otvaramo sa programskim kodom 3. Prvi prozor prikaziva okvire od kamere koja snima oči vozača, drugi prozor prikaziva okvire od kamere koja snima vanjsku okolinu koja dolazi ususret vozaču, a treći prozor je prozor koji će biti postavljen na LCD matricu i prikazivat će zaštitni okvir koji je ustvari okvir popunjen bijelom bojom, a crnom bojom na mjestima koja su izračunata kako bi se zatamnio određen dio matrice i spriječilo prodiranje svjetlosti. Kontinuirano prikazivanje okvira rezultira time da se može u realnom vremenu pratiti ono što kamere gledaju u obliku videa uživo (engl. *live stream*).

```

142 win_name_eyes = 'Eyes Camera Preview'
143 open_window(win_name_eyes)

145 win_name_light = 'Light Camera Preview'
  
```

```
146 open_window(win_name_light)  
148 win_name_protection = 'Protection Preview'  
149 open_window(win_name_protection)
```

Programski kod 3: Otvaranje prozora na zaslonu

### 3.1.4. Višedretvenost zbog raspodijele I/O zadataka

Zadatci koje program treba odradivati su:

1. dohvačanje/čitanje okvira iz video izvora - ulaznog uređaja (kamere),
2. analiziranje okvira (otkrivanje svjetlosti i očiju),
3. spremanje pozicija i okvira u redove,
4. čitanje okvira iz redova,
5. izračunavanje pozicije koju treba zatamniti na LCD matrici,
6. prikazivanje okvira u prozorima na zaslonu.

Navedeni zadatci su većinom vezani za ulazno/izlazne operacije (engl. *input/output bound - I/O bound*) te ih je sve potrebno izvršavati istovremeno, zbog čega je korisno koristiti dretve kako bi se zadatci podijelili po dretvama koje možemo zamisliti kao dodatne radnike u firmi zbog kojih će se moći obaviti više posla paralelno s ostalim poslom. Dretve donose pojednostavljen dizajn koda i njihovo pravilno implementiranje ne može stvoriti situaciju da jedan zadatak zaustavlja izvođenje ostalih zadataka [6].

U CPython implementaciji treba uzeti u obzir da se zbog GIL-a (engl. *Global Interpreter Lock - GIL*) samo jedna dretva može izvršavati Python kod odjednom, a ovo se ograničenje može izbjegić korištenjem specijaliziranih biblioteka, no dojam paralelnosti se ipak postiže visokofrekventnom izmjenom rada nad dretvama. Dretve su prikladne za korištenje kod ulazno/izlaznih operacija, dok se kod procesorski složenijih zadataka savjetuje korištenje procesa [7]. U ovom radu zadatci za analiziranje i izračunavanje nisu procesorski zahtjevni, stoga će ih dretve izvršavati.

Potrebno je uključiti biblioteku pomoću sljedećeg programskog koda 4:

```
2 import threading
```

Programski kod 4: Uključivanje biblioteke *threading*

Iz glavne dretve programa će se kreirati nova dretva koja će obavljati gore prva tri navedena zadatka: čitanje, analiziranje i spremanje pozicija i okvira, dok će glavna dretva obavljati gore zadnja tri navedena zadatka: čitanje, izračunavanje i prikazivanje pozicija i okvira. Programski kod 5 prikaziva definiranje i pokretanje nove dretve (153. do 157. linija koda) te pozivanje funkcije (159. linija koda) i inicijaliziranje dretvenog događaja (151. linija) na glavnoj dretvi koji će služiti za prekidanje čitanja okvira iz kamere na novoj dretvi. Kod stvaranja dretve definirana je funkcija koju ona treba izvršavati, proslijeden joj je dretveni događaj kako bi ga

mogla osluškivati i označena je kao *daemon* dretva što znači da glavna dretva smije završiti iako ona nije završila. Dretveni događaji su mehanizmi komunikacije između dretava na način da jedna dretva može čekati postavljanje određenog događaja od strane druge dretve kako bi krenula sa svojim radom [7].

```
151 stop_read_event = threading.Event()
152
153 threading.Thread(
154     target=read_analyze_and_save_frames,
155     args=(stop_read_event,),
156     daemon=True
157 ).start()
158
159 read_calculate_and_show_frames(win_name_eyes, win_name_light, win_name_protection)
```

Programski kod 5: Inicijaliziranje dretvenog događaja *stop\_read\_event*, stvaranje i pokretanje nove dretve i pozivanje funkcije *read\_analyze\_and\_save\_frames()* u glavnoj dretvi

### 3.1.5. Kamere - čitanje okvira s video izvora

Kako bi se u realnom vremenu moglo otkriti zasljepljuće svjetlo i oči u cilju smanjivanja zasljepljujećg svjetla potrebno je cijelo vrijeme neprekidno pratiti sadržaj onoga što kamere vide odnosno čitati okvire sa video izvora. Podsjetnik - jedan pročitan okvir je ustvari jedna fotografija iz videa. Programski kod 6 prikazuje definiciju *read\_analyze\_and\_save\_frames()* funkcije koja se izvršava na posebnoj novoj dretvi, a iz definicije funkcije može se vidjeti kako isprogramirati čitanje okvira sa video izvora.

```
13 def read_analyze_and_save_frames(stop_event):
14     camera_indexes = [0, 2]
15
16     eyes_source = cv.VideoCapture(camera_indexes[0])
17     light_source = cv.VideoCapture(camera_indexes[1])
18
19     while not stop_event.is_set():
20         has_eye_frame, eyes_frame = eyes_source.read()
21         has_light_frame, light_frame = light_source.read()
22
23         if not has_eye_frame or not has_light_frame:
24             print("Frame not found. Check cameras.\n")
25             break
26
27         detect_eyes(eyes_frame)
28         detect_light(light_frame)
29
30     eyes_source.release()
31     light_source.release()
```

Programski kod 6: Definicija funkcije *read\_analyze\_and\_save\_frames()*

Potrebno je inicijalizirati video izvore na način da se pronađu odgovarajući indeksi za kamere koje će se koristiti. U ovom radu, index za ugrađenu kameru laptopa je 0, dok je za eksternu kameru indeks 2 (14. linija koda). Kada su uspješno identificirani indeksi za kamere moguće je inicijalizirati video izvore korištenjem klase *VideoCapture* iz OpenCV biblioteke koja omogućava video snimanje iz kamera i dodatno je moguće snimati iz video datoteka i nizova fotografija/okvira [8] (16. i 17. linija koda). Čitanje iz nizova okvira će se koristiti u ovom radu prilikom prikazivanja okvira gdje će se okviri čitati iz redova *eyes\_frames\_queue* i *light\_frames\_queue*. Nakon prestanka korištenja video izvora potrebno je video izvore otpustiti (30. i 31. linija koda).

Nakon što su video izvori inicijalizirani moguće je čitati okvire iz njih. Kako bi cijelo vrijeme neprekidno čitali jedan po jedan okvir sa kamere i prikazivali ih u stvarnom vremenu, potrebno je koristiti *while True* petlju. U ovom radu koristi se *while not stop\_event.is\_set()* petlja (19. linija koda) kod koje je izraz *notstop\_event.is\_set()* uvijek jednak Booleovoj vrijednosti *True*, zbog čega će se stalno izvršavati dok događaj *stop\_event* koji je kao argument proslijeden u *read\_analyze\_and\_save\_frames()* funkciju ne bude postavljen na *True* u slučaju kada korisnik želi prekinuti program.

Okvir se čita sa metodom *read()* iz klase *VideoCapture* (20. i 21. linija koda), a ona objedinjuje metode *grab()* i *retrieve()* te dekodira vrijednost okvira [8]. Metoda *read()* vraća informacije o tome je li okvir uspješno pročitan i njegovu vrijednost, a njegova vrijednost je u formi *NumPy* niza [9]. Na slici 6 može se vidjeti da taj NumPy niz sadrži cjelobrojne vrijednosti (engl. *integers*) koje predstavljaju RGB vrijednosti kanala piksela sa fotografije odnosno okvira. *NumPy* biblioteka služi za rad sa nizovima [10]. Ako okvir nije uspješno pročitan, prekida se izvođenje petlje (23. do 25. linija koda).

```
[[137 137 137]
 [133 133 133]
 [136 139 130]
 ...
 [121 119 126]
 [112 108 118]
 [ 92  89  98]]

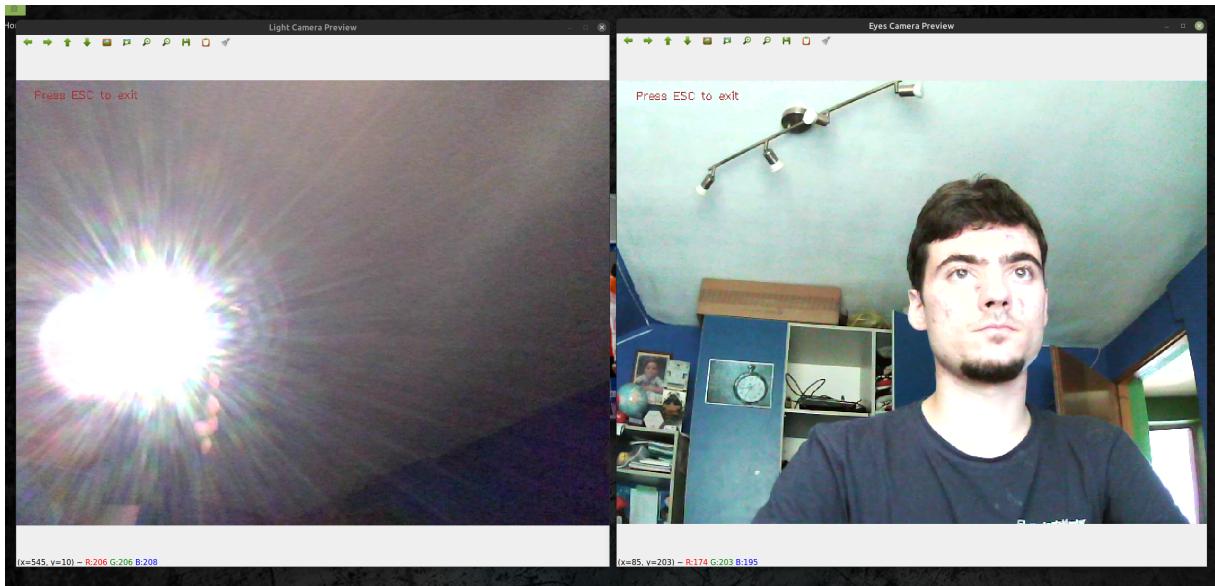
[[137 137 137]
 [132 132 132]
 [136 138 133]
 ...]
```

Slika 6: Ispis dijela vrijednosti za okvir koju vrati *cv :: VideoCapture :: read* metoda [autorski rad]

U svakom krugu petlje čita se po jedan okvir i odmah se taj okvir analizira funkcijama *detect\_eyes()* i *detect\_light()* kako bi se otkrila pozicija zasljepljujućeg svjetla i očiju (27. i 28. linija koda). Funkcije *detect\_eyes()* i *detect\_light()* predstavljaju komponente sustava koje su obrađene u sljedećim poglavljima: "Komponenta za prepoznavanje i pozicioniranje izvora svjetla" i "Komponenta za prepoznavanje i pozicioniranje očiju vozača".

Slika 7 prikaziva kako okviri izgledaju prije otkrivanja zasljepljujućeg svjetla i očiju. Na

zaslonu ne trebaju biti otvoreni prikazani prozori sa slike 7, ali su tu samo kako bi se uvidjelo da sustav odradišva ono što treba i kako bi se bolje shvatilo kako sustav radi.



Slika 7: Prikaz okvira prije otkrivanja zasljepljujućeg svjetla i očiju [autorski rad]

## 3.2. Komponenta za prepoznavanje i pozicioniranje izvora svjetla

Nakon što se uspješno čitaju okviri s kamere koja gleda okolinu koja je ispred vozača, potrebno je analizirati svaki okvir i otkriti nalazi li se na njima zasljepljujuća svjetlost. Ako se svjetlost nalazi na okviru, potrebno je pronaći točne koordinate svjetlosti na okviru. Ovo poglavlje će kroz programski kod 8 koji prikaziva definiranje funkcije *detect\_light()* za otkrivanje zasljepljujuće svjetlosti opisati navedeni potrebnii zadatak.

Potrebno je uključiti biblioteku pomoću sljedećeg programskog koda 7:

```
3 import numpy as np
```

Programski kod 7: Uključivanje biblioteke *numpy*

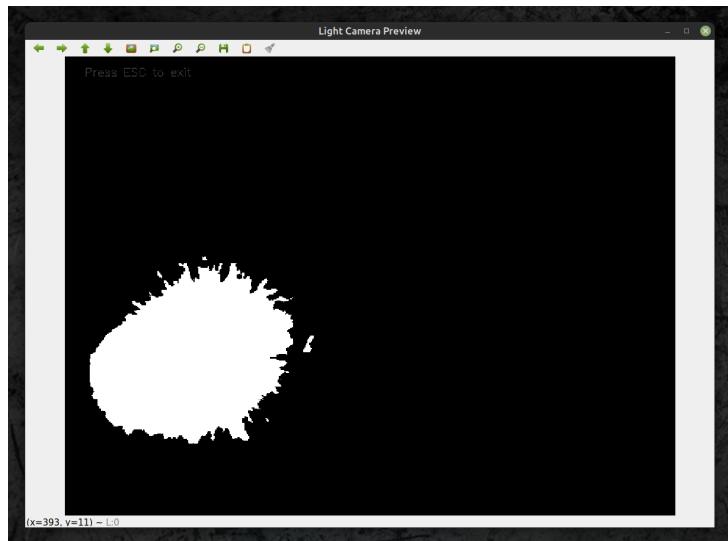
```
50 def detect_light(light_frame):
51     lower_range = np.array([0, 0, 255])
52     upper_range = np.array([240, 11, 255])
53
54     light_hsv_frame = cv.cvtColor(light_frame, cv.COLOR_BGR2HSV)
55
56     color_mask = cv.inRange(light_hsv_frame, lower_range, upper_range)
57
58     contours, _ = cv.findContours(color_mask, cv.RETR_EXTERNAL, cv.
59     CHAIN_APPROX_SIMPLE)
60
61     min_contour_area = 4000
62     big_contours = [contour for contour in contours if cv.contourArea(contour) >
63     min_contour_area]
64
65     light_positions = []
66
67     for contour in big_contours:
68         light_positions.append(cv.boundingRect(contour))
69
70         x, y, w, h = cv.boundingRect(contour)
71         cv.rectangle(light_frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
72
73     light_position_queue.put(light_positions)
74
75     drawText(light_frame, 'Press ESC to exit', (20, 20))
76
77     light_frames_queue.put(light_frame)
```

Programski kod 8: Definicija funkcije *detect\_light()*

Idealno bi bilo kada bi ovakvo rješenje mogli kombinirati i sa senzorom koji može izmjeriti intenzitet svjetlosti jer bi se mogla otkriti i zasljepljujuća svjetlost drugih boja osim raspona bijele boje koji je jedini definiran u ovom rješenju za otkrivanje zasljepljujuće svjetlosti te također bi se svjetlost mogla lakše klasificirati. U programskom kodu 8 definiran je raspon bijele boje koji će

se otkrivati na okviru (51. i 52. linija koda). Bijela boja je izabrana zato što je svjetlost najčešće bijele do žute boje.

Kao argument ova funkcija prima okvir koji treba analizirati. Potrebno je tom okviru promijeniti prostor boja (engl. *color space*) iz BGR prostora boja (engl. Blur-Green-Red - BGR; OpenCV koristi BGR prostor boja umjesto RGB prostora boja zbog toga što je prilikom početnih godina OpenCV-a BGR prostor boja bio dosta popularniji kod proizvođača kamere i pružatelja softvera, a tako je i ostalo do danas [11]) u HSV prostor boja (54. linija koda) zbog toga što funkcija *inRange()* biblioteke OpenCV koristi HSV prostor boja te ona će kao argumente primiti originalni okvir i raspon boja. Funkcijom *inRange()* dobit ćemo masku okvira popunjenu crno-bijelom bojom gdje bijela boja predstavlja objekt od interesa koji se svojom bojom nalazi u definiranom rasponu boje [12] (56. linija koda). Na slici 8 možete vidjeti kako je funkcija *inRange()* stvorila masku okvira sa crnom i bijelom bojom gdje bijela boja predstavlja zasljepljujuću svjetlost:

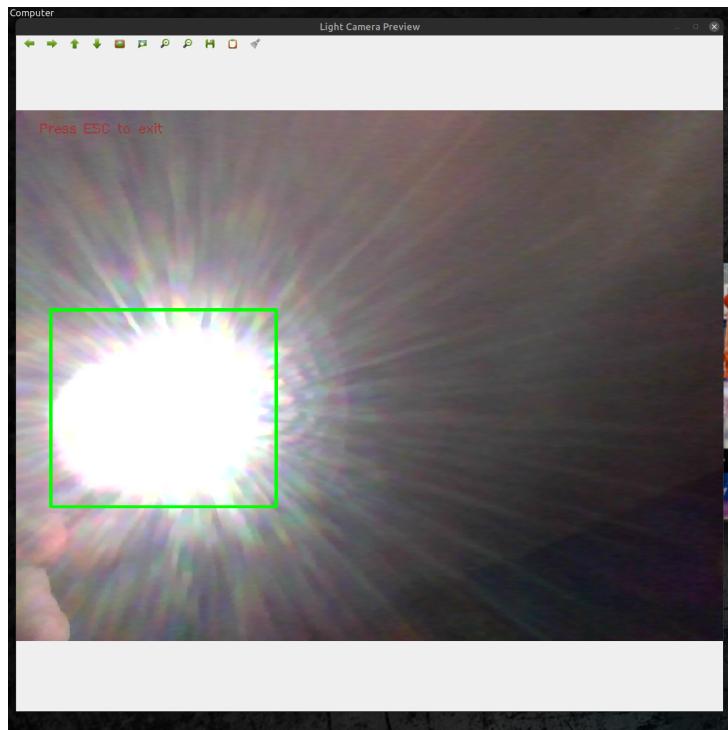


Slika 8: Prikaz crno-bijele maske okvira [autorski rad]

Sada se može reći da je zasljepljujuća svjetlost otkrivena, ali se ne zna njezina točna pozicija s koordinatama što će poslužiti kod obavljanja računanja zaštitnog okvira za LCD matricu. Stoga svo ovo provedeno pretvaranje okvira je učinjeno zbog toga što će maska okvira biti proslijedena funkciji *findContours()* biblioteke OpenCV koja još prima argumente *cv.RETR\_EXTERNAL* i *cv.CHAIN\_APPROX\_SIMPLE* koji definiraju kakve će se konture praviti (58. linija koda). U ovom rješenju kontura predstavlja pravokutnik koji obuhvaća objekt od interesa i za taj pravokutnik dobijemo koordinate gornje lijeve točke, visinu i širinu. Funkcija *findContours()* nije obavezno tražila svo provedeno pretvaranje okvira ali zbog boljeg pronalaska kontura preporučeno je da funkcija primi binarni okvir - crno-bijelu masku kako bi rubovi bili što izraženiji [13]. Kako bi samo imali konture najizraženijih i najvećih svjetlosti odrađeno je i filtriranje (60. i 61. linija koda).

Sada je potrebno proći kroz svaku pronađenu konturu kako bi ju dodali u *light\_position\_queue* red i kako bi na originalnom okviru iscrtali zeleni pravokutnik oko zasljepljujuće svjetlosti i spremili originalni okvir u *light\_framesq\_queue* red (63. do 75. linija koda). Potrebno je naglasiti da

pralaleno radi i glavna dretva koja odmah uzima okvire stavljene u red i prikaziva ih u prozorima na zaslonu, a o tome će biti više riječi u poglavlju "Komponenta za polarizaciju LCD matrice kao reaktivne komponente". IsCRTavanje zelenog pravokutnika na originalnom okviru (slika 9) napravljeno je samo zbog toga kako bi se uvjerili da otkrivena zasljepljuće svjetlosti radi:



Slika 9: Prikaz otkrivene svjetlosti [autorski rad]

### 3.3. Komponenta za prepoznavanje i pozicioniranje očiju vozača

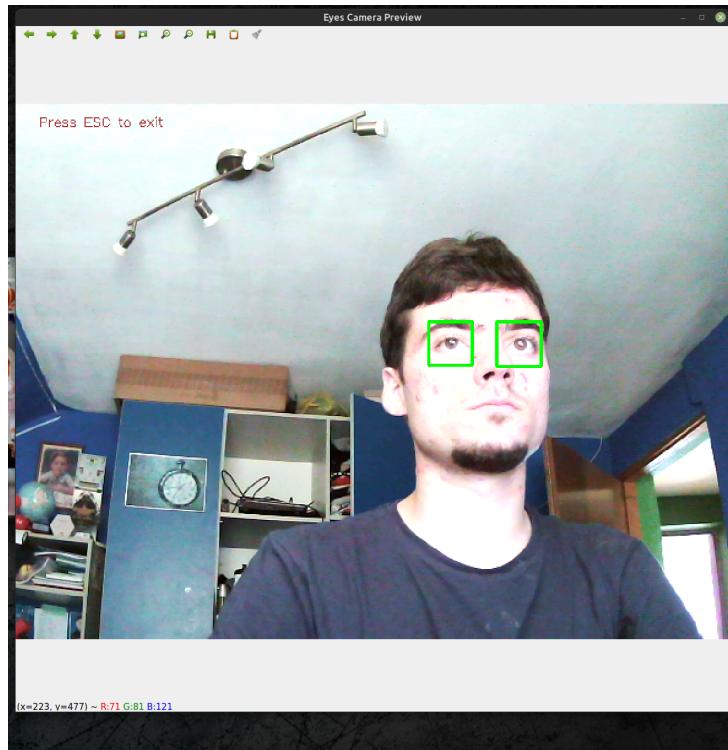
Nakon što se uspješno čitaju okviri s kamere koja gleda u vozača, potrebno je analizirati svaki okvir i otkriti nalaze li se na njima oči. Ako se oči nalaze na okviru, potrebno je pronaći točne koordinate očiju na okviru. Ovo poglavlje će kroz programski kod 9 koji prikaziva definiranje funkcije *detect\_eyes()* za otkrivanje očiju opisati navedeni potrebnii zadatak.

```
33 def detect_eyes(eyes_frame):
34     eyes_gray_frame = cv.cvtColor(eyes_frame, cv.COLOR_BGR2GRAY)
35
36     eye_cascade_model = cv.CascadeClassifier(cv.data.haarcascades + 'haarcascade_eye
37 .xml')
38
39     eyes = eye_cascade_model.detectMultiScale(eyes_gray_frame, scaleFactor=1.1,
40     minNeighbors=5, minSize=(30, 30))
41
42     eyes_position_queue.put(eyes)
43
44     for (x, y, w, h) in eyes:
45         cv.rectangle(eyes_frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
46
47     drawText(eyes_frame, 'Press ESC to exit', (20, 20))
48
49     eyes_frames_queue.put(eyes_frame)
```

Programski kod 9: Definicija funkcije *detect\_eyes()*

Otkrivanje očiju se vrši pomoću OpenCV klase *CascadeClassifier* koja je model za otkrivanje objekata pomoću Haar kaskadnih klasifikatora, a to je pristup temeljen na strojnom učenju gdje je je kaskadna funkcija trenirana na mnogo pozitivnih (sadržavaju objekt od interesa) i negativnih fotografija (ne sadržavaju objekt od interesa) [14]. Prije nego se kreće analizirati okvir, preporučeno je zbog boljih rezultata da se okvir prebací u sivi prostor boja (34. linija koda).

Prilikom inicijaliziranja objekta klase *CascadeClassifier* navodi se što je objekt od interesa - ovdje je to oko (36. linija koda). Kada je objekt inicijaliziran možemo njegovoj metodi *detectMultiScale* proslijediti okvir koji se treba analizirati te je još moguće proslijediti argumenta koji bi utjecali na rezultat. Kao rezultat metoda vraća otkrivene objekte različitim veličinama u obliku liste pravokutnika koji opisuju točnu poziciju očiju na okviru [15]. Listu pravokutnika spremamo u *eyes\_position\_queue* red za daljno računanje (40. linija koda), a još će se i lista pravokutnika iscrtati na originalnim okvirima i spremiti u *eyes\_frames\_queue* red kako bi prilikom prikazivanja okvira uvidjeli da komponenta pravilno radi (slika 10).



Slika 10: Prikaz otkrivenih očiju [autorski rad]

### 3.4. Komponenta za polarizaciju LCD matrice kao reaktivne komponente

## **4. Testiranje sustava**

## **5. Zaključak**

Ovdje treba sažeto rezimirati najvažnije rezultate razrade teme rada. Potrebno je sažeto opisati što je predmet rada, koje su metode, tehnike, programski alati ili aplikacije korištene u razradi rada te koje su pretpostavke dokazane, a koje opovrgnute. Sadržajno, ono što se u uvodu rada najavljuje i kasnije je obuhvaćeno u samom radu, moralo bi biti opisano u zaključnom dijelu kroz rezultate rada.

# Popis literature

- [1] IBM, (bez dat.) "What is Computer Vision?" Adresa: <https://www.ibm.com/topics/computer-vision> (pogledano 20.8.2023.).
- [2] OpenCV, (bez dat.) "About - OpenCV". adresa: <https://opencv.org/about/> (pogledano 20.8.2023.).
- [3] A. Mordvintsev, (bez dat.) "OpenCV: Introduction to OpenCV-Python Tutorials". adresa: [https://docs.opencv.org/4.x/d0/de3/tutorial%7B%5C\\_%7Dpy%7B%5C\\_%7Dintro.html](https://docs.opencv.org/4.x/d0/de3/tutorial%7B%5C_%7Dpy%7B%5C_%7Dintro.html) (pogledano 22.8.2023.).
- [4] Animotica Blog, "Everything You Need To Know About FPS in Video Editing" [Slika], 2020. adresa: <https://www.animotica.com/blog/fps-in-video-editing/> (pogledano 25.8.2023.).
- [5] Python Software Foundation, (bez dat.) "queue — A synchronized queue class — Python 3.11.4 documentation". adresa: <https://docs.python.org/3/library/queue.html> (pogledano 25.8.2023.).
- [6] Anderson Jim, (bez dat.) "An Intro to Threading in Python – Real Python". adresa: <https://realpython.com/intro-to-python-threading/> (pogledano 26.8.2023.).
- [7] Python Software Foundation, (bez dat.) "threading — Thread-based parallelism — Python 3.11.5 documentation". adresa: <https://docs.python.org/3/library/threading.html> (pogledano 26.8.2023.).
- [8] OpenCV, (bez dat.) "cv::VideoCapture Class Reference". adresa: [https://docs.opencv.org/3.4/d8/dfe/classcv%7B%5C\\_%7D1%7B%5C\\_%7D1VideoCapture.html](https://docs.opencv.org/3.4/d8/dfe/classcv%7B%5C_%7D1%7B%5C_%7D1VideoCapture.html) (pogledano 26.8.2023.).
- [9] N. Reshma, "OpenCV cv2.VideoCapture() Function - Scaler Topics", 2023. adresa: <https://www.scaler.com/topics/cv2-videocapture/> (pogledano 26.8.2023.).
- [10] NumPy, (bez dat.) "Array objects — NumPy v1.25 Manual". adresa: <https://numpy.org/doc/stable/reference/arrays.html> (pogledano 26.8.2023.).
- [11] M. Satya, "Why does OpenCV use BGR color format ? | LearnOpenCV". adresa: <https://learnopencv.com/why-does-opencv-use-bgr-color-format/> (pogledano 26.8.2023.).
- [12] OpenCV, (bez dat.) "Thresholding Operations using inRange". adresa: [https://docs.opencv.org/3.4/da/d97/tutorial%7B%5C\\_%7Dthreshold%7B%5C\\_%7DInRange.html](https://docs.opencv.org/3.4/da/d97/tutorial%7B%5C_%7Dthreshold%7B%5C_%7DInRange.html) (pogledano 26.8.2023.).

- [13] OpenCV, (bez dat.) "Contours : Getting Started". adresa: [https://docs.opencv.org/3.4/d4/d73/tutorial%7B%5C\\_%7Dpy%7B%5C\\_%7Dcontours%7B%5C\\_%7Dbegin.html](https://docs.opencv.org/3.4/d4/d73/tutorial%7B%5C_%7Dpy%7B%5C_%7Dcontours%7B%5C_%7Dbegin.html) (pogledano 26. 8. 2023.).
- [14] OpenCV, (bez dat.) "Cascade Classifier". adresa: [https://docs.opencv.org/3.4/db/d28/tutorial%7B%5C\\_%7Dcascade%7B%5C\\_%7Dclassifier.html](https://docs.opencv.org/3.4/db/d28/tutorial%7B%5C_%7Dcascade%7B%5C_%7Dclassifier.html) (pogledano 26. 8. 2023.).
- [15] OpenCV, (bez dat.) "cv::CascadeClassifier Class Reference". adresa: [https://docs.opencv.org/3.4/d1/de5/classcv%7B%5C\\_%7D1%7B%5C\\_%7D1CascadeClassifier.html%7B%5C#%7Daaf8181cb63968136476ec4204ffca498](https://docs.opencv.org/3.4/d1/de5/classcv%7B%5C_%7D1%7B%5C_%7D1CascadeClassifier.html%7B%5C#%7Daaf8181cb63968136476ec4204ffca498) (pogledano 26. 8. 2023.).

# Popis slika

1.	Pojednostavljen prikaz sustava u trodimenzionalnom koordinatnom sustavu [autorski rad] . . . . .	1
2.	Prikaz LCD matrica s prednje strane kada je izvađena iz monitora [autorski rad] .	6
3.	Prikaz LCD matrica sa zadnje strane kada je izvađena iz monitora [autorski rad]	7
4.	Prikaz uzastopnih fotografija/okvira koji čine video od jedne sekunde [4] . . . . .	8
5.	Slikovit prikaz reda kao strukture podataka [autorski rad] . . . . .	9
6.	Ispis dijela vrijednosti za okvir koju vrati <i>cv :: VideoCapture :: read</i> metoda [autorski rad] . . . . .	12
7.	Prikaz okvira prije otkrivanja zasljepljućeg svjetla i očiju [autorski rad] . . . . .	13
8.	Prikaz crno-bijele maske okvira [autorski rad] . . . . .	15
9.	Prikaz otkrivene svjetlosti [autorski rad] . . . . .	16
10.	Prikaz otkrivenih očiju [autorski rad] . . . . .	18

# Sadržaj

1. Uključivanje biblioteke <i>OpenCV</i> . . . . .	8
2. Uključivanje biblioteke <i>queue</i> i inicijaliziranje redova . . . . .	9
3. Otvaranje prozora na zaslonu . . . . .	9
4. Uključivanje biblioteke <i>threading</i> . . . . .	10
5. Inicijaliziranje dretvenog događaja <i>stop_read_event</i> , stvaranje i pokretanje nove dretve i pozivanje funkcije <i>read_analyze_and_save_frames()</i> u glavnoj dretvi . .	11
6. Definicija funkcije <i>read_analyze_and_save_frames()</i> . . . . .	11
7. Uključivanje biblioteke <i>numpy</i> . . . . .	14
8. Definicija funkcije <i>detect_light()</i> . . . . .	14
9. Definicija funkcije <i>detect_eyes()</i> . . . . .	17