Sustav praćenja pokreta oka

Varoščić, David

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

Stručni studij

SUSTAV ZA PRAĆENJE POKRETA OKA

Završni rad

David Varoščić

Osijek, 2022.

SADRŽAJ

1. UVOD

U današnjem svijetu postoji sve veća potreba za unaprjeđenjem tehnologije i računalnih sustava. Jedan od rezultata tih pothvata je računalni vid. On spada u područje umjetne inteligencije gdje je konačan rezultat prepoznavanje 2D i 3D predmete koji se nalaze na slikama i izvlačenje informacija vezanih uz njih. Bazira se na metodama stjecanja, analiziranja i obrade slike. Osnovni ciljevi su prepoznavanje objekata, praćenje objekata, rekonstrukcija slika i sl. Pomoću područja neuronskih mreža i dubokog učenja brzo se razvija računalni vid i imamo sustave koji rade s velikom brzinom i preciznošću u detekciji objekata na slikama i videouradcima. Većina njih se bazira na istreniranim modelima koji služe računalima kao baza po kojoj traže određene objekte na slici. Primjena računalnog vida je dosta široka. Primjenjuje se za detekciju i praćenje automobila u prometu, praćenje ljudi u trgovačkim centrima i na ulicama, daje mogućnost robotima koji imaju slobodu kretanja da se snalaze u raznim prostorima, u industrijskim postrojenjima za detekciju objekata na traci, u automobilskoj industriji se sve više implementira na sustavima vozila za unaprjeđenje sigurnosti tijekom vožnje, primjenjuje se u medicini za otkrivanje tumora, raka i sl.

Detekcija lica i njegovih karakteristika postaje sve popularnija, jer zbog napretka tehnologije pronalazimo sve više primjena za nju. Od korištenja vlastitog lica kao sigurnosnog elementa preko kojeg se otključavaju vrata za ulaz u dom, otključavanje mobitela, prijenosnih računala i sl. Može se koristiti za praćenje kupaca pri kupovini namjernica radi povratnih informacija koje se koriste u svrhu marketinga. U području kriminalistike pomaže u identifikaciji pojedinaca. Od mnogih karakteristika lica naš fokus će biti detekcija očiju i uspješno praćenje kretanja oka. Ovakva primjena se može koristiti kao mjere opreza tijekom vožnje, ako vozač često zatvara oči na duže periode. Može se koristiti u medicinske i psihološke svrhe za praćenje pacijenata, već postepeno unaprjeđuje kvalitetu onesposobljenih ljudi putem specijalnih uređaja i naprednog softvera. U novije vrijeme se koristi infracrvena tehnologija, virtualna realnost, naočale za pračenje smjera gledanja. Uz to sve se primjenjuje strojno učenje i sustavi koji se sastoje od više istreniranih modela. Samo praćenje pogleda se sve više koristi u statističke svrhe pri sakupljanju informacija.

1.1. Zadatak rada

Cilj završnog rada je napraviti softversko rješenje, te uz pomoć njega i kamere laptopa trebamo moći detektirati oči od osobe i pratiti njihovo kretanje. Također treba omogućiti detekciju treptanja osobe, te nadovezati akciju treptanja s jedinm ili dvostrukim klikom miša.

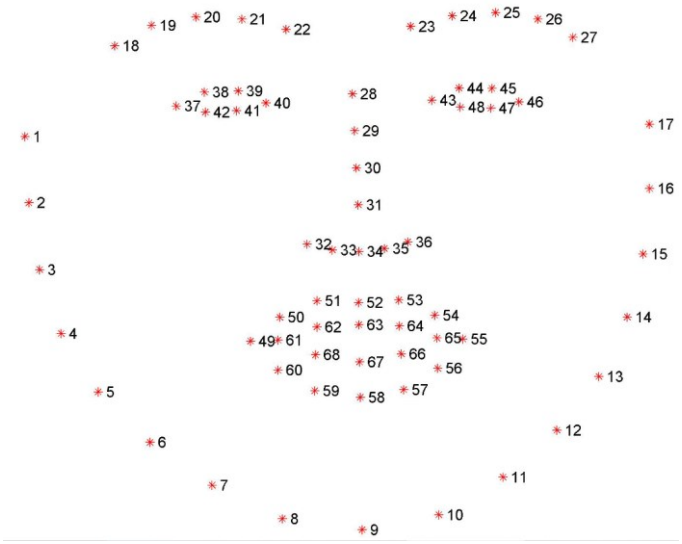
2. TEORIJSKA OBRADA SUSTAVA ZA PRAĆENJE OČIJU I DETEKCIJU TREPTANJA

Fokus ovog segmenta je opis biblioteka i njihovih funkcija koje će se koristiti za realizaciju projekta i njihovo međudjelovanje za postizanje detekcije očiju i treptanja. Objasniti tok programa i neka osnovna načela na kojima se baziraju spomenute radnje. *Editor* koji se koristi je *VS Code* (*Visual Studio Code*). Radi se u *Python* programskom jeziku koji je opće namjene i jednostavan za koristiti. Koristi se za pravljenje *web* stranica i softvera, automatizacija zadataka, analiza podataka, vizualizacija podataka i sl.

2.1. Teorijski osvrt i potrebni elementi za rješavanje zadataka

2.1.1. Dlib biblioteka s istreniranim modelom za detekciju i praćenje lica i njegovih karakteristika

Dlib je *open source* biblioteka napisana u C++ koja se koristi za pravljenje aplikacije koje se baziraju na strojnom učenju i analizi podataka. Glavni razlog za njegovo korištenje će biti „*Dlib's 68-point facial landmark detector*“. To je detektor za lica i njegove karakteristike, koji u sebi ima istrenirane modele za pronalazak lica na slikama. Bazira se na metodi HOG(„*Histogram of Oriented Gradients*“) pokazivača obilježja. Fokusira se na pronalasku 68 specifičnih točaka koje mapiraju lice osobe (slika 2.1.) i onda prati te točke. Sve korištene točke su indeksirane, tako da im je lagano pristupiti. Kao povratnu vrijednost daju svoje trenutne koordinate (x,y) za svaku točku.



Slika 2.1. Raspored predefiniranih točaka koje se koriste za detekciju lica i njegovih karakteristika, [1].

Ovo je osnova na kojoj gradimo ostatak programa, pošto se prvo treba pronaći lice i oči. Jasno se iz slike mogu vidjeti karakteristike lica kao što su usta, nos, oči, obrve i sl. Prethodno spomenute točke su specificirane u istreniranom modelu koji koristi iBUG300-W(*300 Faces In-The-Wild Challenge*) skup podataka. On je napravljen od 300 slika unutar objekata i 300 izvan njih, [1]. Obuhvaća veliku raznolikost od izraza lica, osvjetljenja, veličine lica, poza lica... Lica su uvijek bila jedno od problematičnijih dijelova tijela za detektirati i pratiti zbog velike fleksibilnosti i raznolikosti. Cilj je da se može vršiti uspješna detekcija u svakidašnjim uvjetima. Za treniranje modela je svakako bolje imati što više primjera, jer se s tim dobiva precizniji rezultat. Važno za napomenuti je da rezolucija slika igra veliku uloga, s većom rezolucijom se povećava vrijeme potrebno za pronalazak traženih elemenata zbog većeg raspona vrijednosti. O području primjene detektora ovisi želimo li više sličnih primjera izvedenih u nekim predefiniranim uvjetima ili želimo više raznolikosti ako se detekcija treba izvršavati u svakodnevnim uvjetima. Primjere korištenih slika za pravljenje *dataset-a* možemo vidjeti ispod (slika 2.2. i slika 2.3.).

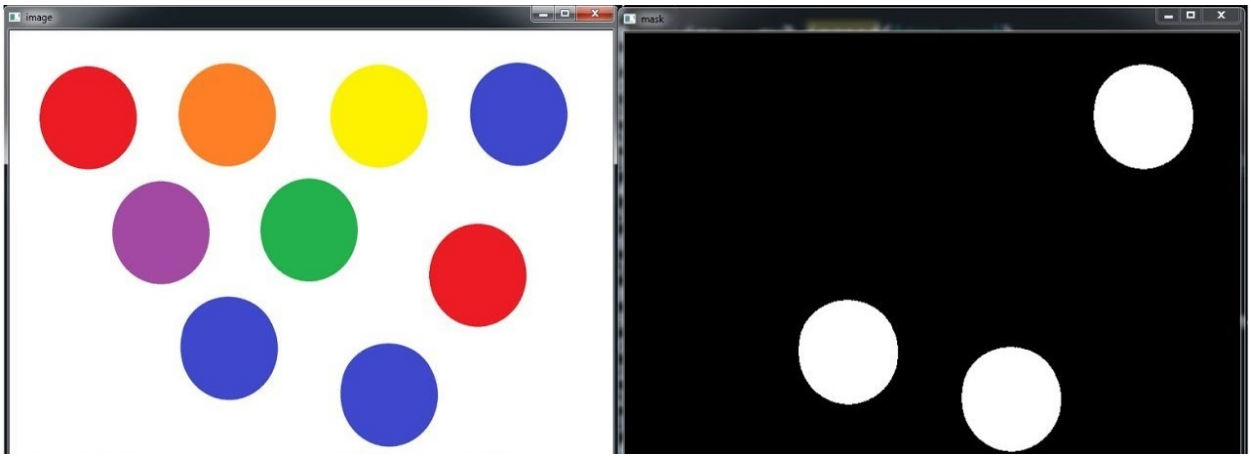
Slika 2.2. Primjer korišten za pravljenje modela, slika slavljenja publike na događaju, [1].

Slika 2.3. Primjer korišten za pravljenje modela, slika pobjednika s peharom, [1].

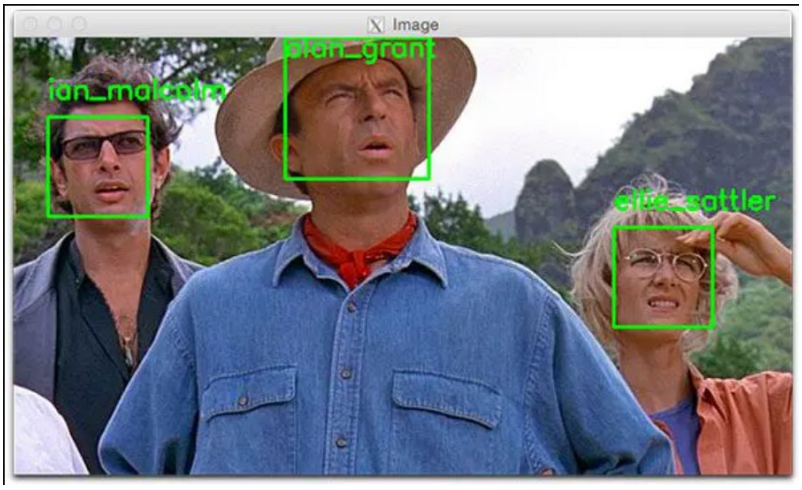
Na slikama s više lica su se često određena lica rezala i onda zasebno stavljala kao primjer za model. Osim ovog modela ima ih još koji se baziraju na istom principu rada, samo može varirati broj točaka za detekciju ili istreniranost. Većina modela koji se temelje na ovaj način rada imaju dobre rezultate unatoč elementima kao što su zakrenutost glave u odnosu na kameru, različita osvjetljenja, kutovi osvjetljenja i izrazi lica. Mogu se praviti zasebni modeli koji traže samo specifične dijelove lica, kao oči, nos ili slično. S time možemo ubrzati vrijeme detekcije, no moramo paziti da količina informacija za detekciju ne bude previše mala. Možemo narušiti cjelokupnu kvalitetu i točnost detekcije. Zbog svoje brzine i pouzdanosti u detekciji lica i njegovih karakteristika „*Dlib 68-point faical landmark detector*“ je jedan od najkorištenijih dektektora za lice, [2].

2.1.2. OpenCV biblioteka za računalni vid

OpenCV (*Open Source Computer Vision Library*) je biblioteka programskih funkcija koje su usmjerene na računalni vid. Započet je u Intel-u 1999. godine. Pomoću nje možemo napraviti grafičko korisničko sučelje, obrađivanje slika, implementacija detektora za specifične elemente u slikama, analiza i obrada videouratka, detekcija objekata, strojno učenje i sl (slika 2.4. i slika 2.5.). Koristan skup funkcija i mogućnosti koje implementiramo u projekt za pokretanje kamere laptopa i snimanje, obrađivanje slike i uređenje napravljenih prozora s tekstualnim i vizualnim elementima, [3].



Slika 2.4. Primjer korištenja OpenCV biblioteke za detekciju plave boje, [8].

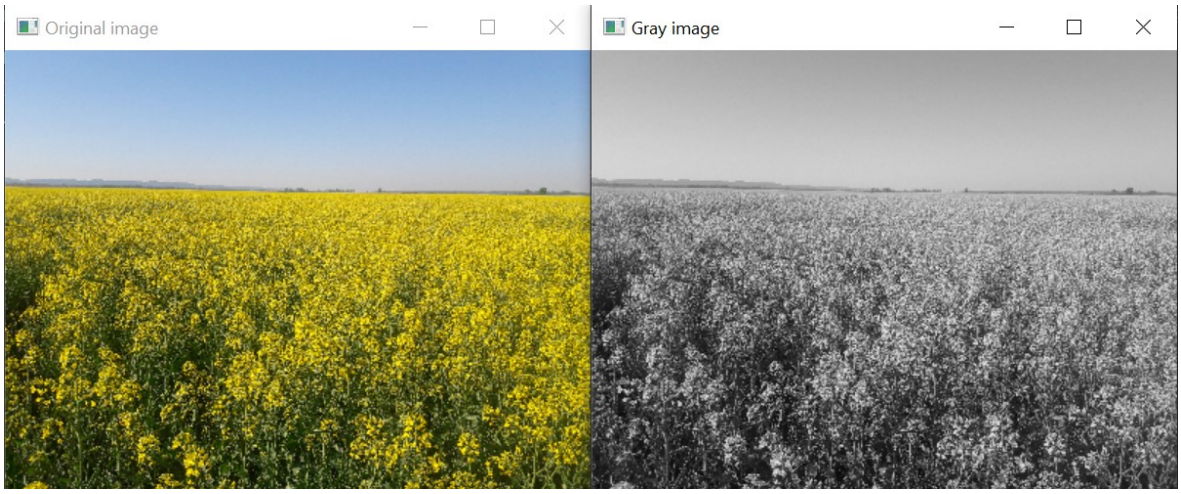


Slika 2.5. Primjena OpenCV biblioteke za pronalazak lica i identifikaciju osoba, [7].

Obrada slike je baza na kojoj se temelji većina područja računalnog vida. U nastavku se spominju neke od najčešćih metoda koje su primjenjivane u postupcima obrade slike.

Konture su linije koje ocrtavaju granice nekog lika. Zbog toga su korisne za analizu oblika i detekciju objekata. Najčešće korištene metode su za pronalazak kontura i nakon toga njihovo označavanje radi lakšeg prikazivanja na slikama. U slučaju jasnih slika gdje se likovi mogu lagano raspoznati, pronalazak kontura nije problem. No, ako se radi o kompleksnijim slikama gdje imamo puno ispreplitanja boja, objekata i drugih stvari, sama slika se mora pripremiti da bi mogli uspješno konturirati željene elemente. Metoda aproksimacije kontura se često koristi za njihovo pojednostavljenje. Primjer korištenja ovoga je u slučaju da nas samo zanimaju krajnje točke nekog oblika.

Boje piksela u digitalnim slikama su kombinacija nekih elementarnih boja i elemenata ovisno o korištenom modelu. Često se koristi RGB (Red, Green, Blue) model. Kombinacijama te 3 boje možemo dobiti sve ostale boje. Vrijednosti tih boja se prikazuje u obliku 3 broja (R,G,B), gdje svaki broj zastupa jačinu svoje boje od vrijednosti 0 - 255. Gdje je 0 najslabije, a 255 najjače. Postoje i drugi modeli boja kao što su CMYK, Grayscale, HSB i sl. Svaki ima svoj princip na kojem se mogu dobivati druge boje. Grayscale ili model sivih tonova se često koristi u obradi slike. On koristi samo komponentu osvjetljenja da bi definirao „boju“. Ima samo jednu vrijednost koja varira od 0 – 255 (0 za crnu boju, a 255 za bijelu). Pretvaranje RGB slike u Grayscale, dobivamo sliku u sivim tonovima (slika 2.6.). Jedan razlog za to pretvaranje je da smanjimo količinu informacija u pikselima, s time smo smanjili kompleksnost slike. Na taj način možemo istaknuti osvjetljenost u slikama, koja je u nekim slučajevima puno važnija za prepoznavanje vizualnih značajki. Postoje algoritmi koji će bolje raditi ili su napravljeni da samo rade sa slikama koje su u grayscale formatu, [11].



Slika 2.6. Usporedba slike u RGB modelu i Grayscale modelu.

Thresholding je metoda segmentiranja slike gdje mijenjanjem piksela slike olakšavamo analliziranje slike, to je binariziranje slike. Imamo više načina na koje možem postaviti *threshold*(granica ili prag) na sliku. *Simple thresholding* (slika 2.7.) se odnosi na to da se ista vrijednost *thresholda* ili praga postavlja za sve piksele. Ako je vrijednost piksela ispod praga postavlja se na 0, u suprotnom ako je iznad praga postavalja se na maksimalnu vrijednost. Pikseli kojima je pridodana vrijednost 0 postaju crni, a pikseli kojima je pridodana maksimalna vrijednost poprimaju nijansu ekvivalentu toj vrijednosti. Imati na umu da za bijelu boje vrijednost treba biti 255. Izvorna slika na kojoj se vrši *thresholding* je najčešće prvo pretvorena u grayscale radi boljeg rezultata. Druga metoda se zove *adaptive thresholding* koja se koristi u slučaju da na slici ima

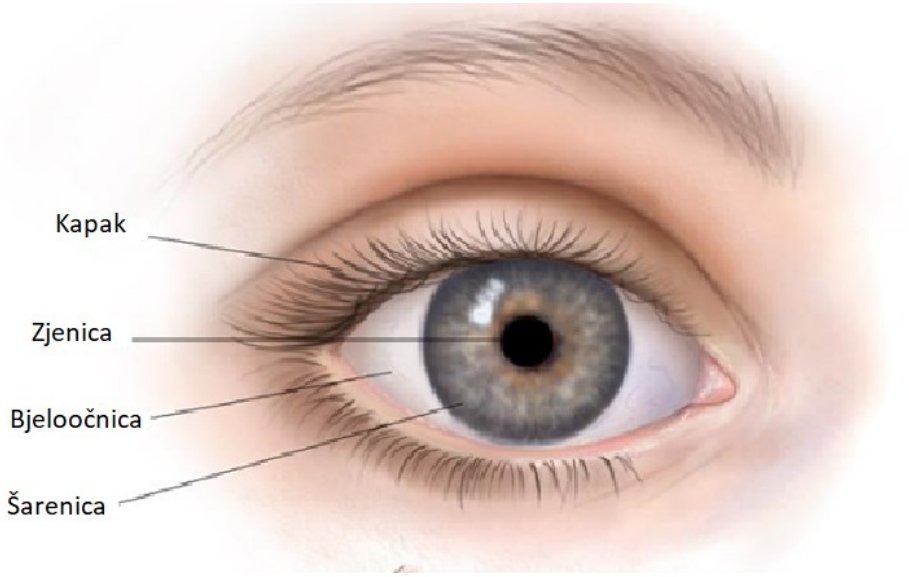
osvjetljanja raznih jačina na više područja. Zbog toga se određuje drugačija vrijednost praga za sva područja, za razliku od *simple thresholding*, [12].

Slika 2.7. Primjer simple thresholding metode.

U sljedećem dijelu pričati će se o izvlačenju određenih dijelova slika. Za takve zadatke se često koritse *bitwise* operatori. U njih spadaju AND, OR, NOT i XOR operatori. Oni se koriste u slučajevima kad područje koje nas zanima nije pravokutnog oblika. Zajedno u kombinaciji s *bitwise* operatorimo koristimo principe maskiranje za izvlačenje područja interesa od ostatka slike, [4].

2.1.3. Detkecija očiju i njihovog pokreta

Ljudsko oko je jedan od najsloženijih organa u našim tijelima. Ono nam omogućuje vid, percepciju svjetla, percepciju dubine i razlikovanje boje. Njegova glavna funkcija je pretvaranje svjetlosti u živčane impulse. Kada svjetlost dođe do mrežnice fotoreceptori pretvaraju svjetlost u električni signal. Ti signali putuju od mrežnice do mozga koji ih onda pretvara u slike koje vidimo u tom trenutku.



Slika 2.8. Vanjski dijelovi oka, [9].

Ovdje (slika 2.8.) možemo vidjeti vanjske dijelove oka, to jest dijelovi oka koji nas zanimaju,[9]:

- Kapak
- Zjenica
- Bjeloočnica
- Šarenica

Naš konačni cilj je praćenje pokreta oka, a za to bi bilo najbolje i najjednostavnije da se fiksiramo na zjenicu ili šarenicu oka i po tome vršimo praćenje.

Sama detekcija pokreta oka zna biti problematična, jer se ne mogu sve oći detektirati i pratiti. Tu igraju ulogu boja zjenice i šarenice, leće, naočale, osvjetljenje i veličina očiju na cjelokupnoj slici. Mnogi detektori se zaustavljaju na detekciji očiju. Može se napraviti model koji se fokusira na detekciji i praćenju kretanja oka, no trebalo bi puno resursa i vremena za tako nešto. I opet će biti situacije gdje neće biti zadovoljavajući rezultati zbog već navedenih razloga.

Puno će biti jednostavnije koristiti od dlib-a detektor za detekeciju lica i očiju, te pomoću navedenih biblioteka i funkcija napraviti praćenje zjenice ili šarenice oka. Vidjeli smo da su na „*Dlib's 68-point facial landmark detector*“ sve točke indeksirane i na taj način znamo kojem dijelu lica trebaju pripadati.

Slika 2.9. Predefinirane točke za oči, [1].

Pošto nas zanimaju samo oči od istreniranog modela koristimo točke 37 – 42 za lijevo oko i 43 – 48 za desno oko (slika 2.9.). Na pronalazak očiju baziramo ostatak programa. Sada kad imamo naše područje interesa, trebamo ga odvojiti da bi ga mogli početi obrađivati. Područje interesa je čest pojam u segmentima praćenja objekata i obrađivanju slika, zato što s njim „odvajamo“ dio slike koji nas zanima od ostatka. Nije potrebno obrađivati ostatak slike jer će to samo povećati zahtjevnost programa, a s time nismo ništa dobili. Područje interesa možemo dobiti definiranjem koordinata u čijim granicama se nalazi područje. Za to postoje predefinirane i već spomenute funkcije koje možemo koristiti. Ovo može biti jednostavno ili teško ovisno o obliku području interesa koje želimo odvojiti i načinu obrade same slike. Nakon toga svega možemo koristiti kombinacije već prethodno navedenih metoda da dobijemo željeni rezultat. Postupci će se detaljnije objasniti u segmentu za testiranje i krajnje rezultate.

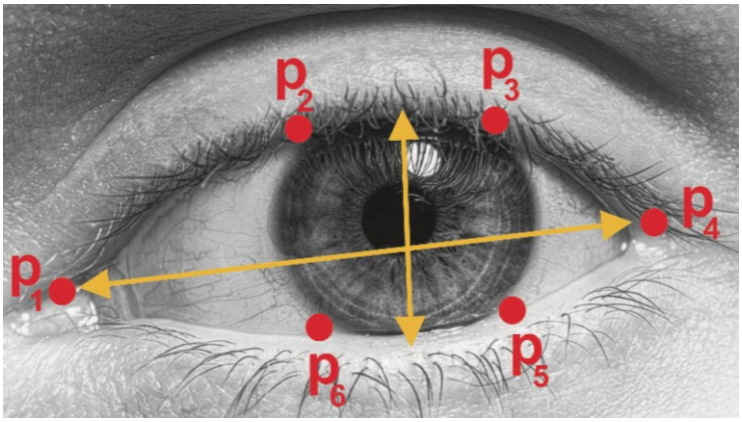
2.1.4. Detekcija treptanja

Detektiranje treptanja može imati više primjena. Indikacija na pospasnost vozača ili operatora nekog vozila, pomoć u asistentskim uređajima za onesposobljene ljude, sigurnosni element u sustavima za prepoznavanje. U ovom projektu treptanje oka treba biti ekvivalentno jednom ili dvostrukom kliku miša. Primjena ovakvog rješenja bila bi korisna kod ljudi koji se ne mogu služiti rukama za upravljanje miša.

Postojeće metode za detekciju treptanja se mogu podjeliti na aktivne i pasivne. Aktivne metode koriste specijalan hardver najčešće u obliku nosivih uređaja kao naočale koje iz blizine prate stanje oka i infracrvene kamere. Pasivne metode koriste samo obične kamere. Neke metode prate količinu pomaka u određenom vremenu, te pokušavaju odrediti da li je oko zatvoreno ili ne.

Metoda koja se primjenjuje je vrlo jednostavno i bazira se na *facial landmark* detektoru koji već koristimo za praćenje očiju. Svaka osoba ima drugačiji način treptanja, pod ovo se misli na brzinu zatvaranja i otvaranja oka, vrijeme zatvorenosti oka i stupanj do kojeg se oko zatvara. U prosjeku zatvorenost oka tijekom treptanja traje 100 – 400 ms. Od poznatih koordinata koje čine okvir oka

uzimamo omjer oka. Za svako oko imamo 6 točaka, možemo ih numerirati p1 – p6 za objašnjavanje načela po kojem se detektirati treptanje.



Slika 2.10. 6 točaka korištenih za lociranje oka, [10].

Po prikazanoj slici (slika 2.10.) možemo vidjeti raspodjeljenost točaka na području oka. Ujedno dobivamo informaciju za stupanj otvorenosti oka od predefinarnih točaka koje se koriste za njegovo praćenje. Mi moramo dobiti visinu i širinu između točaka. Na taj način se za svaku sličicu u sekundi računa visina i širina između točaka, te mi to možemo primjeniti za računanje omjera oka, [10].

Omjer oka:

$$OMJER\ OKA = \frac{visina\ oka}{širina\ oka} \tag{2.1}$$

Računanje omjera oka s točkama:

$$OMJER\ OKA = \frac{||p2 - p6|| + ||p3 - p5||}{2||p1 - p4||} \tag{2.2}$$

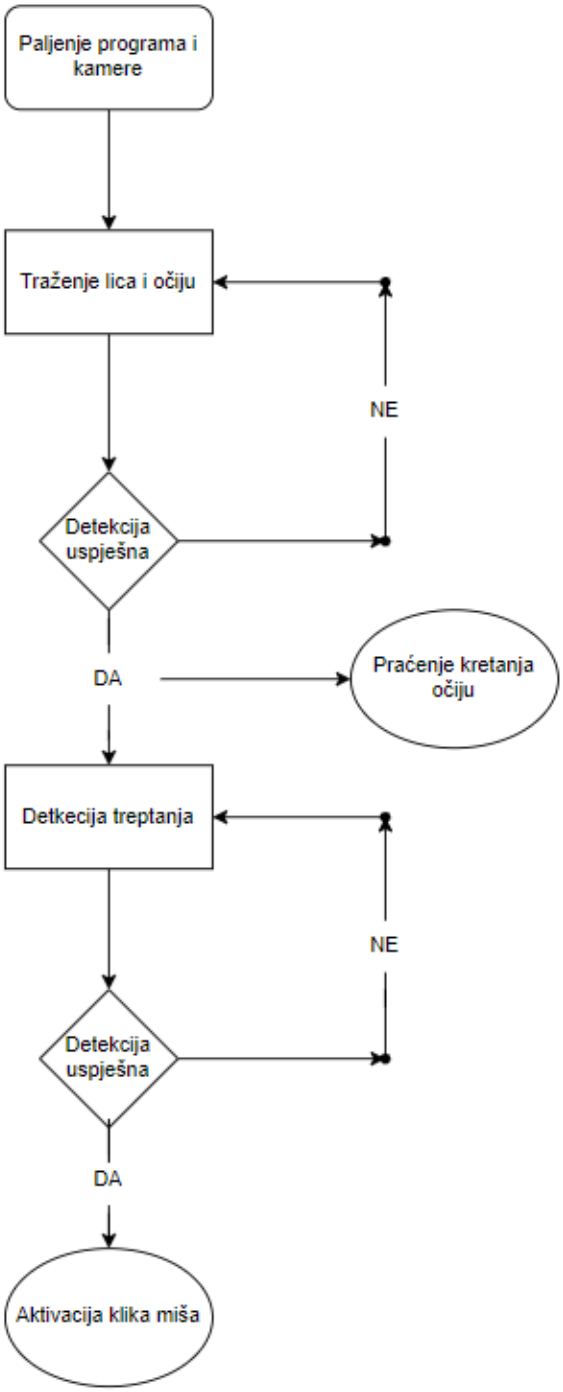
Pomoću gore navedene formule možemo dobivati u stvarnom vremenu trenutnu vrijednost omjera oka, to jest stupanj otvorenosti oka. Pod treptanje se smatra da osoba zatvori i otvori oči u

određenom vremenskom intervalu. Pošto se točke za označavanje očiju razlikuju potrebno je i za lijevo i za desno oko izračunati omjer zasebno. Nakon toga se omjer može zbrojiti i podjeliti s 2, da bi dobili srednju vrijednost rezultata. Koristiti dobiveni rezultat kao cjelokupni omjer oka, [5].

Dobiveno rješenje se treba implementirati i napraviti da se detekcijom treptanja zamjeni klik računalnog miša, taj dio se detaljnije spominje u segmentu za testiranje.

2.2. Prijedlog programskog rješenja

Ovdje se navodi redosljed radnji koje bi se trebale izvršavati da bi se zadatak smatrao uspješnim. Prvobitno se treba program ukljuciti, nakon čega on pali kameru laptopa za korištenje. Pri paljenju kamere će nam se otvoriti prozor koji će prikazivati snimani sadržaj. Zatim se kreće sa detekcijom lica i očiju, bez kojega se ne može preći na drugi dio. U trenutku kad se izvši detekcija i lice se pronađe onda se prelazi na praćenje lica i očiju. Tek nakon toga se može provjeriti detekcija treptanja korisnika. U slučaju da se omjer oka spusti ispod postavljene razine i onda poveća, to se računa kao da je osoba trepnula i izvršava se aktivacija klika miša. U slučaju da osoba izađe iz kadra, udalji se ili se detekcija lica prekine ponavlja se proces detekcije. Ispod se nalazi dijagram toka koji opisuje ciklus rada programa (slika 2.11.).



Slika. 2.11. Dijagram toka programa.

3. REALIZACIJA „SUSTAVA ZA PRAĆENJE POKRETA OKA“

U ovom dijelu će se objašnjavati metodologija iza pravljenja projekta, te spominjati će se principi implementacije zasebnih elemenata. Metode koje će se spominjati nisu jedine za rješavanje određenih problema, no izabrane su zbog dobrih rezultata tijekom korištenja i njihove jednostavnosti.

3.1. Korišteni alati i njihova implementacija u program

3.1.1. Korištenje kamere i prikazivanje slike

Kamera i miš su jedini hardverski element koji su nam potreban za rad i testiranje programa. Iako kamera igra veću ulogu. Tijekom pravljenja projekta se cijelo vrijeme koristila kamera na laptopu. Koristilo smo biblioteku Opencv, to jest cv2 da bi mogli upravljati kamerom.

Koraci pri korštenju kamere:

- `cv2.VideoCapture()` se koristi za inicijaliziranje snimanja kamere.
- postaviti beskonačnu while petlju u kojoj se koristi `read()` za čitanje sličica.
- primjenjujemo `cv2.imshow()` metodu za prikazivanje prozor sa sličicama iz videa.
- napravljeno da se stiskom Q tipke na tipkovnici izlazi iz beskonačne petlje, prekida korištenje kamere i zatvara sve napravljene prozore. S time je program gotov i njegovo izvođenje se završava.

Pri uspoređivanju metoda obrade slike, često bi koristio više `cv2.imshow()` funkcija. Na taj način bi se otvorilo više prozora istovremeno s različitim prikazima.

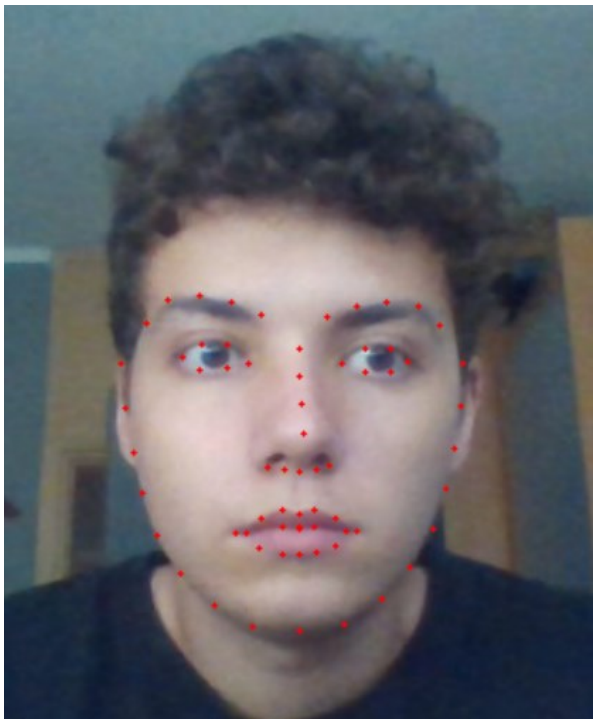
3.1.2. Implementacija Dlibovog istreniranog modela

Prethodno napomenuti model trebamo preuzet da bi ga mogli implementirati. Puni naziv korištene datoteke je „*shape_predictor_68_face_landmarks.dat*“. Datoteke s .dat ekstenzijom su često napravljene da bi ih mogli koristiti samo predefinirane aplikacije i softveri.

Koraci za implementaciju i korištenje prediktora:

- Prvo je potrebno implementirati detektor koji dohvaćamo pomoću dlib funkcije `dlib.get_frontal_face_detector()`.
- Poslije toga unosimo datoteku za prediktor, to jest njezin direktorij u `dlib.shape_predictor()` funkciju, da bi je mogli korisiti u programu.

- Od ovog dijela pa nadalje se sintaksa izvodi u glavnoj petlji programa. Potrebno je pronaći lica u dobivenom sadržaju videozapisa. I na svako uspješno pronđeno lice treba implementirati prediktor.
- Dobivenu informaciju za točke pretvaramo u listu koja sadrži x,y koordinate tih točaka.
- Na svaku koordinatu crtamo krug da bi mogli vidjeti rezultat prediktora na videouratku.



Slika 3.1. Primjena istreniranog modela.

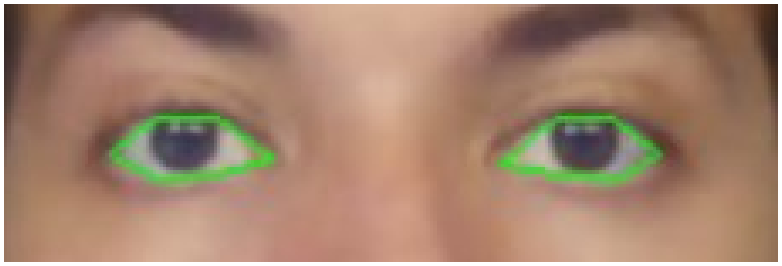
Na slici (slika 3.1.) možemo vidjeti raspoređenost točaka na licu, te dijelove lica koje obuhvaćaju. Ovo je među jednostavnijim primjerima jer je lice ravno s obzirom na kameru i ne predstavlja poteškoće pri detekciji. Ispitivanja modela u ovisnosti o osvjetljenju, udaljenosti od kamere i položaju lica naspram kamere je odrađeno u segmentu za testiranje.

S ovim smo sada uspješno implementirali prediktor, te dok je detekcija lica uspješna preditkor će pratiti naše lice.

3.1.3. Detekcija očiju i njihovo praćenje

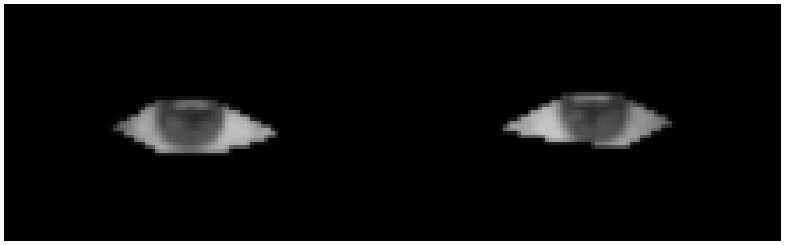
Praćenje samih očiju je najzahtjevniji dio rada, zato što je potrebno dosta toga napraviti da bi mogli imati zadovoljavajući rezultat. Iako postoji više načina na koje možemo pristupiti ovome problemu, svi se baziraju na sličnim principima.

Prvu stvar koju trebamo napraviti je odvojiti oči od ostatka slike, pošto nam ostali elementi u slici nisu više potrebni. Korištenjem dlibovog modela već imamo okvir oko očiju. Sada trebamo te poznate točke odvojiti od ostatka. Za to koristimo `cv2.convexHull()` metodu, koja radi konveksnu ljusku oko određenog područja. Za granice te ljuske koristimo predefinirane točke modela. Sada smo dobili područje interesa s kojim možemo nastaviti raditi (slika 3.2.).



Slika 3.2. Konveksne ljuske oko očiju.

Sljedeće možemo napraviti masku za područje koje nas zanima. Za to se koristimo napravljenom funkcijom `maskOn()`. Pomoću nje odvajamo segment za oči u potpunosti (slika 3.3.), te pri testiranjima i radu možemo se fokusirati samo na njih.



Slika 3.3. Rezultat konveksne ljuske i maskiranja.

Sada slijedi obrađivanje područja interesa da bi naš detektor mogao što bolje pratiti šarenicu oka. U ovom području je potrebno puno testiranja i kombiniranja metoda, ovisno o potrebnom rezultatu. Primjenjuju se morfološke operacije koje se baziraju na obliku slike. Koristimo `cv2.morphologyEx(cv2.MORPH_CLOSE)`. Metoda primjenjuje dilataciju pa nakon nje eroziju. Dobro je za rješavanje crnih točaka koje se znaju javljati na objektu obrade. Implementirana je `cv2.adaptiveThreshold(cv2.ADAPTIVE_THRESH_MEAN_C)` metoda koja se bazira na pronalasku srednje vrijednosti susjedstva piksela, te se od toga oduzima odabrana konstanta C. Ovo je puno

prikladnije nego da koristimo *simple thresholding* metodu, gdje se za cijelu sliku postavlja ista granica (slika 3.4.).

Slika 3.4. Primjena adaptivne binarizacije koja se bazira na računanju srednje vrijednosti susjednih piksela.

Sada treba implementirati cv2 metode `erode()`, `dilate()` i `GaussianBlur()`. Glavni razlog za implementaciju metoda je uklanjanja smetnji i uglađivanje slike za poboljšanu detekciju šarenice. Ovaj dio se sveo na testiranje metode i njihovih iteracija da bi dobili vjerodostojan rezultat. Rezultat njihove implementacije se nalazi na slici ispod (slika 3.5.). Jedino što se sada može detektirati na slici su šarenice i zjenice.

Slika 3.5. Rezultat navedenih metoda obrade slike.

Kao rezultat možemo vidjeti crne mrlje u području oka, najveća točka prezentira šarenicu i zjenicu. Sada nam ostaje primjeniti algoritam koji će se služiti ovime za detekciju i praćenje. U sklopu Blob detektor algortima s njegovim parametrima smo dobili najbolje rezultate za detekciju i praćenje šarenice oka.

Nakon što smo šarenicu i zjenicu pripremili za detekciju možemo implementirati algoritam koji će je vršiti. Blob detektor se bazira na jednostavnom algoritmu za detekciju blobova. *Blob* sam po sebi je skupina povezanih piksela koji imaju nekakva zajednička svojstva. Primjer bloba mogu biti crne mrlje koje možemo vidjeti na slici iznad. Open CV nudi jednostavan algoritam za detekciju blobova. Način rada blob algoritma u koracima, [6]:

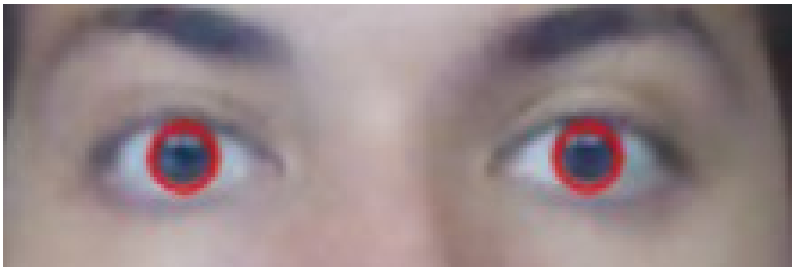
1. Pretvara izvornu sliku u više binarnih inačica, gdje svakoj slici postavlja drugačije parametre `minThreshold` i `maxThreshold`.

2. Izdvaja povezane komponente iz binarnih inačica s `findContours()` metodom, te računa centre tih komponenti.
3. Zatim se ti centri grupiraju ovisno o njihovim koordinatama. Bliski centri će formirati jednu grupu, koja će činiti jedan blob. U ovom formiranju igra ulogu `minDistBetweenBlobs` parametar.
4. Iz dobivenih grupa, to jest blobova se opet računa njihov centar i radijus. Te vrijednosti vraća kao lokacije i veličine ključnih točaka koje označuju sam blob.

Veliku ulogu u detekciji blobova imaju parametri, tako da treba namještati vrijednosti parametara ovisno o veličini i tipovima blobova koje želimo detektirati (slika 3.6.).

Slika 3.6. Parametri Blob algoritma, [6].

Napravljena je funkcija `initBlobDetector()` u kojoj se obavlja postavljanja parametara detektora i njegova inicijalizacija. Sada samo što ostaje je korištenje blob detektora i iscrtavanje dobivenih ključnih točaka, da možemo vidjeti je li detekcija uspješna.



Slika 3.7. Primjena Blob detektora.

Na primjeru iznad (slika 3.7.) možemo vidjeti rezultat detekcije. Bilo je potrebno više rada s parametrima detektora i sa samom detekcijom, no o tome ću više pričati u poglavlju vezanom za testiranje.

3.1.4. Detekcija treptanja

Kao što sam naveo u teorijskom dijelu za detekciju treptanja koristiti ću načelo mjerenja omjera oka korištenjem točaka implementiranog modela. Da bi pristupili samim točkama, potrebno je poznavati broj kojim su numerirane. Princip se objašnjava na lijevom oku (slika 3.8.).

Slika 3.8. Numerirane točke za lijevo oko.

Potrebno je pronaći udaljenost između točaka 37 i 40, točaka 38 i 42 i točaka 39 i 41. Pomoću implementiranog modela već znamo koordinate svih točaka, tako da nije problem pronaći udaljenost između njih. Za pronalazak distance koristimo `math.distance()` metodu, koja vraća vrijednost udaljenosti između 2 točke. S pronađenim udaljenostima smo dobili visinu i širinu oka. Mogli smo i na drugi način pristupiti ovome. Pronaći sredinu između točaka 38 i 39 i 42 i 41. Te onda pronađena udaljenost između tih novih točaka bi bila visina oka. Dobivene vrijednosti zatim treba uvrstiti u jednadžbu za omjer oka.

U svrhu pojednostavljenja primjera, to sve možemo uvrstiti na ovaj način:

$$OMJER\ LIJEVOG\ OKA = \frac{\left||točka[38] - točka[42]|\right| + ||točka[39] - točka[41]||}{2||točka[37] - točka[40]||} \quad (3.1)$$

Na isti način računamo omjer za desno oko, samo što su druge točke u pitanju, pa su im drugačiji indeksi. Na kraju zbrojimo oba omjera, te podjelimo s 2, da bi dobili srednju vrijendnost.

$$OMJER\ OČIJU = \frac{omjer\ lijevog\ oka + omjer\ desnog\ oka}{2} \quad (3.2)$$

Nakon dobivenog omjera treba postaviti vrijednost kao granicu između otvorenog i zatvorenog oka. Prelaskom granice se smatra da se oko otvorilo ili zatvorilo ovisno o prijašnjem stanju u kojem je bilo. Testiranje ovih granica će se izvoditi u sljedećem poglavlju.

Uz to treba postaviti nekakav vremenski element koji će služiti kao brojač. Njegova svrha je da možemo razlikovati radnje treptanja i zatvaranja oka na duži vremenski interval. Kada osoba zatvori oči treba se inicijalizirati brojač i ako se oko otvori unutar intervala od 1 sekunde, izvedena radnja će se računati kao treptaj očiju. U suprotnom se radnja neće računati kao treptaj, te će se trebati ponoviti.

3.1.5. Implementacija miša računala

Jedan od zadataka rada je da ukomponiramo miš, te u slučaju treptanja treba se izvršiti radnja jednaka jednom ili dvostrukom kliku miša. Da bi miš mogli koristiti uvodimo *mouse* biblioteku koja nam daje sve potrebne funkcije za njegovu implementaciju u ostatak programa. Ovaj dio zadatka realiziran je na sljedeći način. Napravljen je kvadrat koji se nalazi na jednom dijelu prozora. U slučaju da se miš nalazi na koordinatama tog kvadrata i ako je osoba trepnula, to se računa kao klik miša te kvadrat mijenja boju. Funkcija `mouse.get_position()` nam vraća koordinate miša, koje koristimo za izvođenje logike.

3.2. Realizacija programskog rješenja i dijagram toka

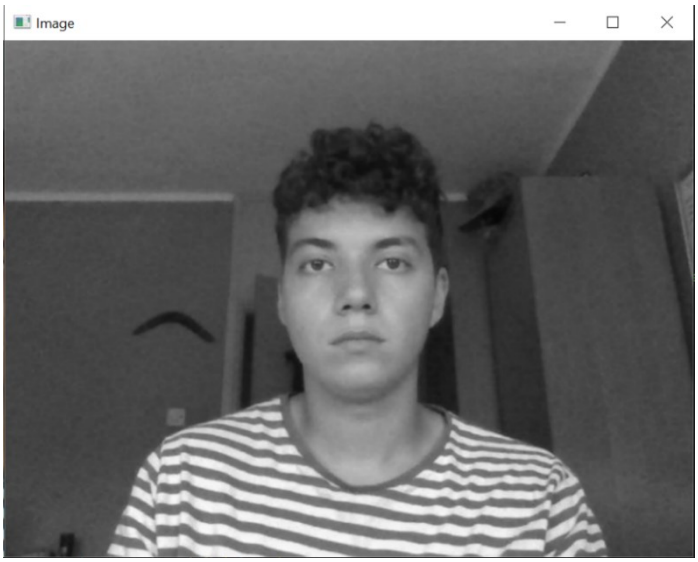
Pri objašnjavanju je program podijeljen na dijelove, radi njihovog lakšeg razumijevanja, pa će se tako objašnjavati njihov rad i tok. Segmenti će se objašnjavti redom kojim su navedeni.

Podjela programa:

1. Okvir programa
2. Glavna petlja
    a. Praćenje pokreta oka
    b. Detekcija treptanja
    c. Prikazivanje podataka

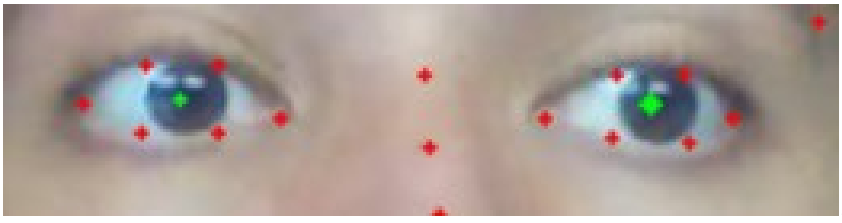
Paljenjem programa se vrši prvobitna inicijalizacija mnogih varijabli koje se koriste u programu. Inicijaliziramo detektore i prediktore pomoću kojih se izvršava detekcija i praćenje lica, te njegovih karakteristika. Pozivamo funkciju `initBlobDetector()` koja služi za inicijalizaciju Blob deteketora i njegovih parametara. Odvajamo točke oko očiju od ostatka točaka, da bi im mogli lakše kasnije pristupiti. Nakon toga palimo kameru `cv2.videoCapture()`. Sve funkcije koje su napravljene se u ovom dijelu nalaze, a u slučaju njihovog korištenja ih pozivamo u kasnijim djelovima. Poslije tih početnih inicijalizacija se kreće izvršavati glavna petlja programa. Potrebno je postaviti petlju da bi kamera mogla uzastopno snimati, snimanje je osnova na kojoj se bazira ostatak programa. U slučaju da želimo ugasiti program, to jest izaći iz glavne petlje. Potrebno je pritisnuti tipku „Q“ na tipkovnici. Nakon toga se prestaje snimati i zatvaraju se svi prozori koju se otvorili povodom snimanja.

Sada kad smo opisali okvir našeg programa, možemo preći na sadržaj unutar glavne petlje. Snimanje kamere se odrađuje pomoću `cap.read()` metode. Nakon toga mijenjamo veličinu prozora i pretvaramo prikazanu sliku iz BGR u Grayscale područje (slika 3.9.). Svrha ovoga je da pojednostavimo sliku i smanjimo zahtjevnost za daljnju obradu.



Slika 3.9. Rezultat nakon konverzije u Grayscale područje.

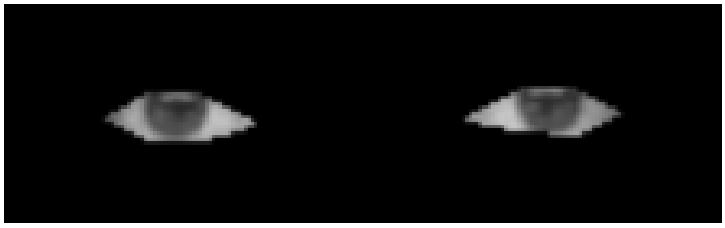
Stvaramo for petlju u kojoj implementiramo prediktor i dajemo mu informacije koje vraća detektor `shape = predictor(gray, rect)`. Shape varijabla služi za uzimanje vrijednosti trenutnih točaka, nakon toga pozivanjem `shapeToNp()` funkcije prelazimo preko svih 68 točaka i njihove povratne vrijednosti pretvaramo u koordinate (x,y). Stvaramo varijable za lijevo i desno oko, pridjeljujemo im specificirane točke s dlibovog modela. Sada imamo točke oko očiju i imamo njihove koordinate, koje se stalno ažuriraju. Pozivamo funkciju `getMiddlePoint()` koja izračunava pomoću postojećih točaka oko očiju središte oka, to koristimo da bi dobili točne koordinate središta za oba oka (slika 3.10.).



Slika 3.10. Točke koje se koriste za opisivanje očiju.

Nakon toga slijedi izdvajanje očiju od ostatka slike. Funkcijom `convexHull()` implementiramo konveksnu ljusku na točke koje čine okvir očiju. S funkcijom `getPolyArea()` računamo površinu unutar konveksne ljuske. Pozivanjem funkcije `maskOn()` vršimo operacije odvajanja područja

obuhvaćenog konveksnom ljuskom od ostatka slike (slika 3.11.), te s tim možemo daljnju obradu fokusirati samo na područje očiju.



Slika 3.11. Rezultat konveksne ljuske i maskiranja.

Sada kada imamo samo oči, možemo početi implementirati morfološke operacije i binarizaciju na sliku. Nećemo previše ulaziti u ovaj dio jer je prethodno objašnjen. Nakon primjenjenih metoda možemo primjeniti blob detektor. Pomoću `blob_detector.detect()` tražimo ključne točke na obrađenoj slici, a s `cv2.drawKeypoints()` ih iscrtavamo na sliku. Bitno je napomenuti da su blob detektor i njegovi parametri postavljeni i inicijalizirani na početku s funkcijom `initBlobDetector()`. Ako imamo potrebu za izmjenjivanje nečega vezano za detektor to radimo unutar te funkcije. Poslije toga se odrađuje dio za gledanje u stranu (slika 3.12.). Zbog njega znamo gleda li osoba u lijevo ili desno. Odrađen je u funkciji `getGazeRatio()`. U toj funkciji uzimamo oko i dijelimo ga na lijevu i desnu stranu. Poslije toga računamo omjer bijelih piksela, koji reprezentiraju bjeloočnicu. Primjenom ove metode dobivamo omjer pomoću kojeg možemo zaključiti gleda li osoba lijevo ili desno. U slučaju da je omjer < x osoba gleda lijevo, a ako je omjer > y osoba gleda desno. Varijable x i y su granice koje postavljamo, ovisno o rezultatima mjerenja.

Slika 3.12. Primjer koji se koristi za određivanja smjera gledanja.

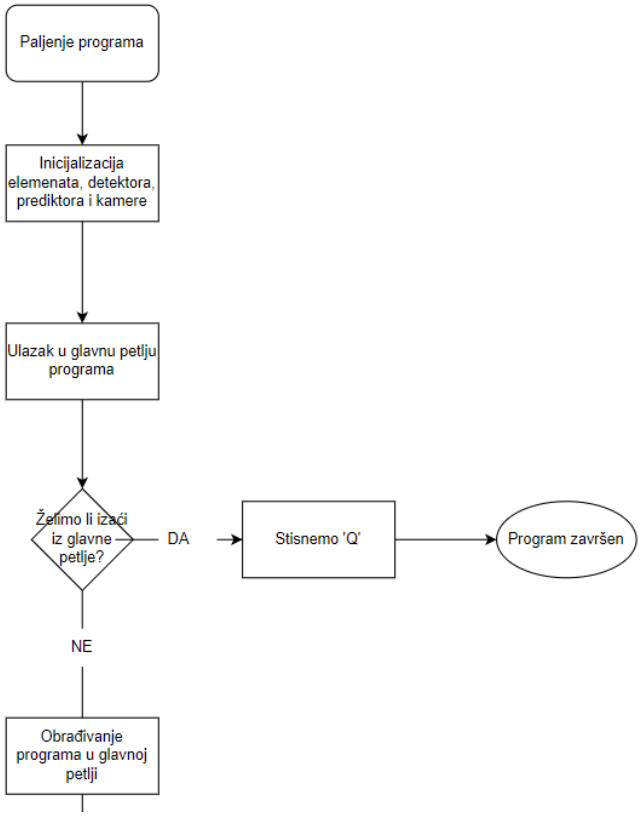
Nakon toga slijedi dio za detekciju treptanja. Pozivamo funkcije `getEyeAscpectRatio()`. U njoj se računa omjer visine i širine očiju pomoću točaka dlib modela. Omjer je proporcionalan otvorenosti oka. Pri paljenju programa se prvo izvodi `earAdaptation()` funkcija. Ona je zadužena za mjerenje granice između otvorenog i zatvorenog oka. Osoba treba par puta trepnuti i nakon toga

stisnuti 'S' na tipkovnici. Nakon toga je omogućena detekcija treptanja. U slučaju da omjer oka padne ispod granice smatra se da je osoba zatvorila oko. Ako nakon toga u određenom vremenu omjer oka postane veći od granice, smatra se da je osoba otvorila oko i registrira se treptaj. Ako se oko držalo zatvorenim više od postavljenog vremena i nakon toga otvori oko, smatra se da osoba nije trepnula. U slučaju da je osoba trepnula dva puta u vremenu od 0.8 sekundi to se registrira kao dupli klik miša.

Sada kada imamo detekciju treptanja, možemo preći na korištenje miša i povezivanje ta dva elementa. Sljedeće se postavljaju koordinate kvadrata. Uzimaju trenutne koordinate miša s `mouse.get_position()` naredbom. Provjerava se da li se miš nalazi na području kvadrata. Kada je taj uvijet prošao provjeravamo jesu li se dogodila dva treptaja ili jedan. Te ovisno o tome se registriraju klikovi, ispisuje se informacija o jednom ili dva klika. Jednako ovim radnjama se mijenja boja kvadrata svaki put kad se klik izvrši.

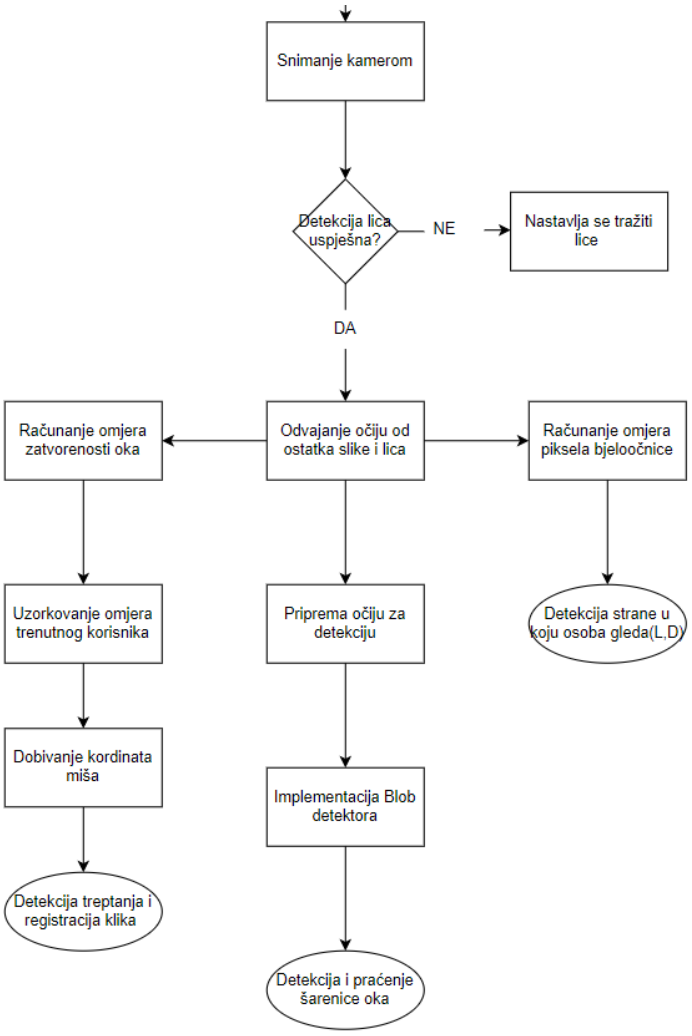
Zadnje što preostaje je prikaz podataka i prikazivanje prozora sa slikom, nakon čega smo došli na kraj glavne petlje. Dok se početno mjerenje granice između otvorenog i zatvorenog oka ne izvede na ekranu će pisati tekst „Trepnite nekoliko puta i pritisnite 'S' nakon toga.“. Ako lice nije moguće detektirati pisati će „Pronalazak lica u tijeku!“. Od informacija možemo uvijek vidjeti ukupan broj treptaja, smjer gdje osoba gleda i registrirani klik ili dvoklik mišem. A u slučaju da želimo vidjeti dodatne informacije trebamo pritisnuti tipku 'W' na tipkovnici. Nakon toga se prikazuje omjer oka, omjer pogleda i koordinate za oba oka. Prikazivanje slike se izvodi s `cv2.imshow()` metodom, pomoću koje možemo vidjeti rezultate programa.

Slijedi dijagram toka programa, koji sadrži njegov pojednostavljen rad. Prvobitno se bavimo s okvirom programa (slika 3.13.). Detaljnija objašnjenja dijelova programa i specifičnih linija koda će se nalaziti u dijelu za priloge.



Slika 3.13. Okvir programa koji sadrži elemente što se samo jednom izvrše.

Sljedeće na redu je dio programa unutar glavne petlje (slika 3.14.), koji se izvodi stalno. Imati na umu da se podijela vrši više po logici programa, a ne po sekvecnijalnom radu.



Slika 3.14. Podijela programa po njegovim funkcijama.

Zadnje što preostaje je ukratko u koracima objasniti rad s programom i način na koji možemo pristupiti informacijama.

Koraci pri korištenju programa:

1. Paljenje programa
2. Na početku će pisati tekst „Trepnite nekoliko puta i pritisnite 'S' nakon toga.“. Ovo služi za mjerenje minimalne vrijednosti omjera oka kod osobe. Nakon toga se treptanje može detektirati.
3. U slučaju da se ne može pronaći lice, bit ćete informirani s tekstom „Pronalazak lica u tijeku.“.
4. Samo praćenje očiju će se izvršavati cijelo vrijeme i informirati će nas u slučaju da gledamo u lijevu ili desnu stranu.

5. Kvadrat koji se nalazi uz desni rub služi za testiranje klikanja. Ako stavimo miš u područje kvadrata i trepnemo to će se registrirati kao klik i kvadrat će promjeniti boju. A ako brzo trepnemo 2 puta to će se računati kao dvoklik miša.
6. Ako pritisnemo 'W' na tipkovnici prikazati će nam se dodatne informacije. Vrijednosti za omjer otvorenosti oka, omjer pomoću kojeg se određuje u koju stranu gledamo i središnje koordinate oba oka.
7. U slučaju da želimo izaći iz programa samo stisnemo 'Q' na tipkovnici.

4. TESTIRANJE I REZULTATI

4.1. Metodologija testiranja

Nakon istraživanja i stjecanja teorijske osnove za primjenjivanje određenih alata i metoda koje su potrebne za realizaciju dijelova rada se prešlo na pisanje programa. Sam kod se pisao u segmentima i kada bi dio bio završen onda bi se ispitivala njegova funkcionalnost. Open CV biblioteka ima funkcije s brojnim parametrima kod kojih ima mnogo izbora. Bilo je mnogo testiranja s te strane, da imamo što bolje rezultate detekcije i praćenja potrebnih elemenata. Potrebno je bilo kombinirati razne metode i ideje da bi dobili pomake u rezultatu. Od velike koristi su bile naredbe `print()` i `cv2.imshow()`, zato što smo pomoću njih prikazivali i uspoređivali rezultate testiranja. Na nekim elementima su rezultati dosta varirali pri istim uvjetima, pa je bilo potrebno puno uzoraka uzeti. Pri završetku rada su se vršila još jednom testiranja zasebnih elemenata i njihov kombinirani rad, kako bi bili sigurni da su zadovoljeni standardi projekta.

4.2. Rezultati testiranja

Od svih elemenata najviše se treba posvetiti testiranju detektora za lice, detekciji i praćenju šarenice oka, smjeru gledanja osobe i detekciji treptanja. Na kraju ćemo vršiti testiranje cjelokupnog rada da osiguramo dinamičnost i nesmetanost pri izvršavanju programa. Imati na umu da iako je veliki broj mjerenja odrađena pri istim uvjetima, možemo većinu vremena očekivati variranje u rezultatima.

4.2.1. Detekcija lica pomoću istreniranog modela

Dlib model i njegove točke su osnova na kojoj radi cijeli program. Ako lice nije pronađeno to znači da nisu ni oči, a s tim onda ne možemo testirati funkcionalnost ostatka programa. Cilj je ispitati detektor pod drugačijim uvjetima. Glavni elementi koji nas zanimaju su osvjetljenje i položaj lica naspram kamere. Testiranja su bila prilično izravna i nije bilo teško ispitati model.

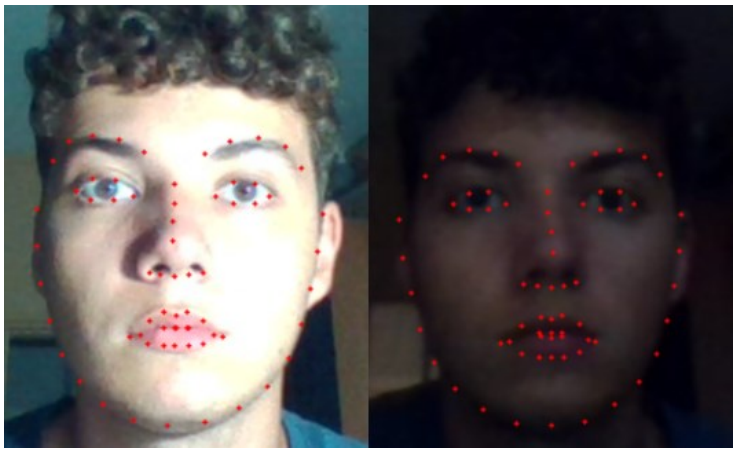
Prvo slijedi testiranje položaj lica naspram kamere (slika 4.1.) i s time provjeriti koje su granice do kojih je detekcija moguća.

Slika 4.1. Testiranje položaja glave i lica naspram kamere.

Kroz ove primjere možemo vidjeti rezultate detekcije pod različitim kutovima i položajima glave. U prve tri slike je lice do određenih granica okrenuto od kamere, te gleda u određenom smjeru. No, detekcija se i dalje izvršava. U 4. slici je glava skoro za 90° zakrenuta naspram kamere, unatoč tome detekor je uspješan u pronalasku. Vidimo da su se točke nakupile u određenim djelovima pošto kamera nema pristup ostalom dijelu lica, pa se predviđa položaj ostalih segmenata.

Slijedi testiranje detektora pod drugačijim uvjetima osvjetljanja (slika 4.2. i slika 4.3.). Mijenjati će se jačina svjetlosti, te će se mijenjati osvjetljenje lica. U nekim slučajevima će lice biti ravnomjerno osvjetljeno, dok će se u drugim primjerima primjetiti razlika.

Slika 4.2. Usporeda rezultata pod različitim osvjetljenjem 1.dio.



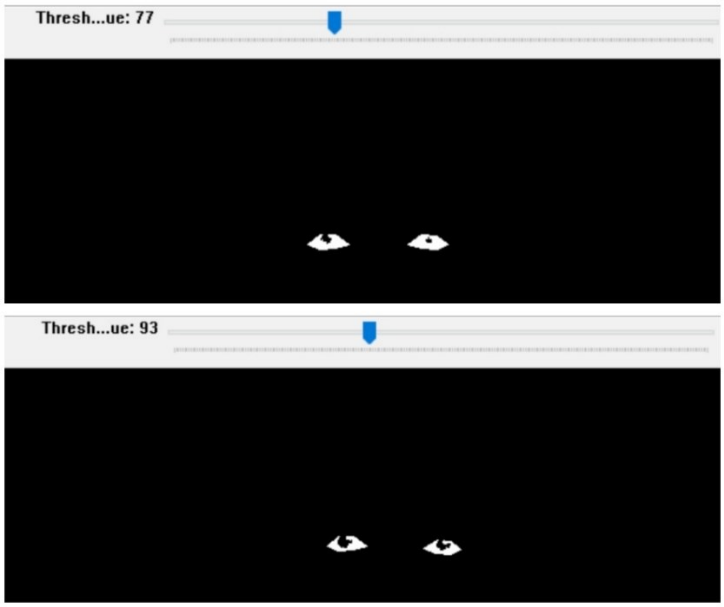
Slika 4.3. Usporeda rezultata pod različitim osvjetljenjem 2.dio.

Kroz ovih 5 primjera s jasno vidljivim razlikama osvjetljenja vidimo da je detektor na svakoj slici uspješno pronašao lice. Zahvaljujući brojnim slikama koja su korištena za pravljenje detektora, rezultat je kvalitetna detekcija i predikcija lica i njegovih komponenti u različitim uvjetima.

4.2.2. Praćenje očiju

Postupak za praćenje kretanja očiju je prethodno objašnjen pa se sada fokus stavlja na načine testiranja dobivenih rezultata. Prvobitno se pokušalo pomoću metoda za pronalazak i crtanje kontura pristupiti problemu. No, problem je bio s hijerarhijom kontura i biranjem ispravne konture koja je okružila šarenicu. Sama detekcija kontura je bila nepouzdana jer uz najmanje pomake su konture nestajale. Zatim se isprobala metoda koja spada pod Open CV biblioteku. To je bio Blob detection algoritam. Taj algoritam se sam bavi detekcijom blobova i njihovim praćenjem u našem slučaju. Preostalo nam je obrađivanje oka i rad s parametrima blob algoritma, da bi dobili što kvalitetniju detekciju.

Prvo se radi na obrađivanju same slike. Binarizacija je jako bitan korak u obradi slike, jer je pojednostavi i pripremi za daljnju obradu. Prvobitni plan je bio koristiti jednostavnu binarizaciju gdje bi testirali s postavljanjem različitih granica. Moramo imati na umu da ovdje veliki faktor ima osvjetljenje i zbog toga se ne može jedna vrijednost granice koristiti cijelo vrijeme. Mogla bi teoretski ako bi osvjetljenje uvijek bilo isto, no to za našu primjenu nije slučaj. Problem nastupa s time da bi morali izaći svako malo iz programa da promjenimo granicu binarizacije. Ovo smo mogli riješiti na par načina. Mogli smo napraviti logiku da se može kadgod želimo unijeti vrijednost granice u program, te bi se onda to spremalo u varijablu i primjenjivalo. Drugo rješenje je bilo da stavimo kliznu traku na naš prozor i pomoću toga bi mogli manualno u programu izmjenjivati granicu binarizacije (slika 4.4. i slika 4.5.).



Slika 4.4. Korištenje klizne trake za mijenjanje granice binarizacije tijekom izvršavanja programa 1.dio.

Slika 4.5. Korištenje klizne trake za mijenjanje granice binarizacije tijekom izvršavanja programa 2.dio.

Svi predstavljeni primjeri su napravljeni pod istim uvjetima u vremenskom razdoblju od 1 minute. Korištenjem ove metode i Blob detektora smo pokušali ostvariti detekciju i praćenje šarenice oka.

Slika 4.6. Detekcija očiju s jednostavnom binarizacijom.

Detekcija je bila uspješna samo u slučajevima kad smo gledali direktno u kameru. Kada bi pokušali gledati u strane ili micati oči ona bi bila rijetko zastupljena. Mijenjanje granica i metoda binarizacijskog postupka je pomoglo s rezultatom. Same ključne točke od detektora su bile previše široko postavljene kao što možemo vidjeti na slikama (slika 4.6.). Nakon ovih rezultata probala se primjeniti adaptivna binarizacija. Ona je malo zahtjevnija na sustav no ovisno o primjeni može donijeti puno bolje rezultate. Kod adaptivne binarizacije (slika 4.7.) primjenjujemo metodu računanja srednje vrijednosti susjednih piksela.

Slika 4.7. Detekcija očiju s adaptivnom binarizacijom.

Odma se vidi napredak u detekciji. Sada se u slučaju gledanja u strane može prepoznati šarenica, te sama detekcija je puno preciznija. No, ima još mjesta za napredak da detekcija bude pouzdanija i dinamičnija. U idućem koraku smo implementirali razne metode za obradu slike da bi poboljšali rezultate.

Statistička usporeda rezultata se odrađivala na sljedeći način. Paralelno se izvodila detekcija s jednostavnom i adaptivnom binarizacijom. Vrijeme izvođenja pokusa je nešto manje od 5 sekundi. Primjer formata koji će se prikazivati je:

- Vrijeme: 15.3059810000000001 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1.

Prvi element naznačuje točan trenutak izvršavanja detekcije, dok drugi element predstavlja broj detektiranih elemenata. On može biti 0, 1 ili 2. Ako je broj elemenata 2 to znači da su oba oka detektirana, a to nam je najpoželjniji rezultat. Prvo uspoređujemo rezultate gledanja direktno u kameru (slika 4.8.).

```
Vrijeme: 15.305981000000001 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 15.4503158 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 15.5679666 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 2
Vrijeme: 15.6802325 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 15.8082321 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 2
Vrijeme: 15.933796 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 16.041803 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 2
Vrijeme: 16.1817334 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 16.3049507 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 0
Vrijeme: 16.415587799999997 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 16.541660300000004 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 16.667169899999998 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 1
Vrijeme: 16.777411999999998 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 16.9194678 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 17.039472000000004 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 17.149825399999997 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 17.299148799999998 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 17.4189219 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 17.538345300000003 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 17.667279399999998 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 0
Vrijeme: 17.7937887 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 17.8952703 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 18.031336000000003 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 0
Vrijeme: 18.1592866 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 18.2592822 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 18.393863099999997 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 18.520936900000002 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 18.633850799999998 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 2
Vrijeme: 18.761653100000004 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 18.883632499999997 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 0
Vrijeme: 18.9932527 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 0
Vrijeme: 19.154716200000003 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 19.272429199999998 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 0
Vrijeme: 19.382235200000004 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 1
Vrijeme: 19.521993000000002 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 19.635930100000003 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 0
Vrijeme: 19.7373289 || Jednostavna binarizacija: 2 // Adaptivna binarizacija: 1
Vrijeme: 19.8863878 || Jednostavna binarizacija: 1 // Adaptivna binarizacija: 2
```

Slika 4.8. Primjer usporedbe rezultata jednostavne i adaptivne binarizacije.

Mjerenja su u istim uvjetima odrađena za oba primjera. Sljedeće možemo uzeti srednje vrijednosti oba mjerenja i vidjeti po rezultatu koja metoda je uspijela više detekcija imati. Srednju vrijednost se dobije tako da zbrojimo detekcije za jedan primjer i to podjelimo s brojem mjerenja koji je uzet za taj primjer. Tu imamo jako puno faktora koji utječu na rezultate i najmanji pokreti mogu utjecati na detekciju. Zbog toga ponavljamo sveukupno 10 puta mjerenja na način kao prethodno i na kraju uzimamo srednju vrijednost svih mjerenja.

Broj mjerenja:	Ostvarene detekcije za jednostavnu bin.:	Ostvarene detekcije za adaptivnu bin.:	Broj mjerenja u 5 sekundi:	Rezultati jednostavne binarizacije:	Rezultati adaptivne binarizacije:
1.	67	67	45	1.489	1.489
2.	30	44	47	0.638	0.936
3.	47	78	47	1.000	1.659
4.	87	81	48	1.813	1.688
5.	32	32	49	0.653	0.653
6.	46	51	49	0.939	1.041
7.	22	38	48	0.458	0.792
8.	43	31	48	0.896	0.646
9.	32	41	47	0.681	0.872
10.	73	77	46	1.587	1.674

Tablica 4.1. Usporedba metoda binarizacije, rezultati detekcije šarnice pri statičnom gledanju

Sada možemo naći srednju vrijednost svih rezultata, kako bi konačnu usporedbu mogli napraviti.

$$Jednostavnva\ bin._{sr} = \frac{10.154}{10} = 1.015 \tag{4.1}$$

$$Adaptivna\ bin._{sr} = \frac{11.45}{10} = 1.145 \tag{4.2}$$

Primjećujemo da su oba načina slična u rezultatu, detekcija s adaptivnom binarizacijom je postigla malo bolji rezultat. Problem s jednostavnom i načinom implementacije je da ljudski faktor namještanja granice jako utječe na konačni rezultat. Slijedi mjerenje u slučajevima kada se oko kreće i gleda u strane. Za izvođenje ovog testa će oko pomicati redosljedom gore, dolje, lijevo i desno.

Broj mjerenja:	Ostvarene detekcije za jednostavnu bin.:	Ostvarene detekcije za adaptivnu bin.:	Broj mjerenja u 5 sekundi:	Rezultati jednostavne binarizacije:	Rezultati adaptivne binarizacije:
1.	24	54	44	0.545	1.227
2.	13	46	44	0.295	1.045
3.	7	45	45	0.156	1.000
4.	31	48	45	0.689	1.067
5.	29	48	45	0.644	1.067
6.	37	52	45	0.822	1.156
7.	19	43	45	0.422	0.956
8.	13	40	45	0.289	0.889
9.	28	56	45	0.622	1.244
10.	15	35	45	0.333	0.778

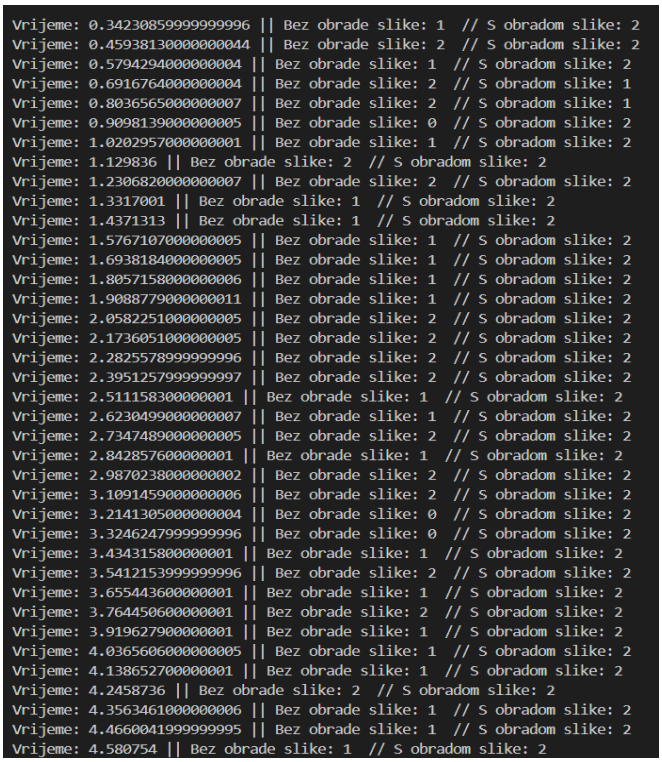
Tablica 4.2. Usporedba metoda binarizacije, rezultati detekcije šarnice u pokretu i gledanje u strane

$$Jednostavnva\ bin._{sr} = \frac{4.817}{10} = 0.482 \tag{4.3}$$

$$Adaptivna\ bin._{sr} = \frac{10.429}{10} = 1.043 \tag{4.4}$$

U drugom testiranju je adaptivna binarizacija imala bolji rezultat i općenito bolje detektira oko u pokretu. Puno nam je važnije da imamo dobru detekciju oka u pokretu, a sada samu preciznost detekcije možemo poboljšati korištenjem morfoloških operacije i drugih metoda za obradu slike. S usporedbom mjerenja i još nekim elementima koje smo uzeli u obzir, adaptivna binarizacija je bolji izbor za korištenje u svrhu našeg rada.

Slijedi usporedba detekcije prije i poslije primjene morfoloških operacija i zamućenja slike (slika 4.9.). Testiranje će se vršiti istim postupcima kao prethodno. Imamo paralelno mjerenje oba slučaja i usporedbu konačnih rezultata.



Slika 4.9. Primjer usporedbe rezultata detekcija bez obrade slike i s obradom.

Broj mjerenja:	Ostvarene detekcije za primjer bez obrade:	Ostvarene detekcije za primjer s obradom:	Broj mjerenja u 5 sekundi:	Rezultati detekcije bez obrade slike:	Rezultati detekcije s obradom slike:
1.	86	93	47	1.830	1.979
2.	74	94	47	1.574	2.000
3.	62	88	45	1.378	1.956
4.	54	90	45	1.200	2.000
5.	69	93	47	1.468	1.979
6.	42	92	46	0.913	2.000
7.	41	90	45	0.911	2.000
8.	76	90	45	1.689	2.000
9.	62	90	45	1.378	2.000
10.	49	90	45	1.089	2.000

Tablica 4.3. Usporedba obrade slike, rezultati detekcije šarnice pri statičnom gledanju

$$Bez\ obrade\ slike._{sr} = \frac{13.43}{10} = 1.343 \tag{4.5}$$

$$S\ obradom\ slike._{sr} = \frac{19.914}{10} = 1.991 \tag{4.6}$$

S obradom slike smo postigli gotovo savršenu detekciju tijekom gledanja u zaslon i kameru laptopa. Što se ovog dijela tiče nema potrebe za daljnjim radom na segmentu za detekciju. Slijedi provjera kad se predmet detekcije kreće i gleda u strane. Oko će se pomicati gore, dolje, lijevo i desno, a nakon toga se gleda u razne strane da se na malo izazovniji način testira praćenje.

Broj mjerenja:	Ostvarene detekcije za primjer bez obrade:	Ostvarene detekcije za primjer s obradom:	Broj mjerenja u 5 sekundi:	Rezultati detekcije bez obrade slike:	Rezultati detekcije s obradom slike:
1.	52	90	46	1.130	1.957
2.	67	91	46	1.457	1.978
3.	50	90	46	1.087	1.957
4.	60	90	45	1.333	2.000
5.	44	94	47	0.936	2.000
6.	29	89	46	0.630	1.935
7.	30	89	47	0.638	1.894
8.	51	90	46	1.109	1.957
9.	44	87	46	0.957	1.891
10.	57	91	47	1.213	1.936

Tablica 4.4. Usporedba obrade slike, rezultati detekcije šarnice u pokretu i gledanje u strane

$$Bez\ obrade\ slike._{sr} = \frac{10.49}{10} = 1.049 \tag{4.7}$$

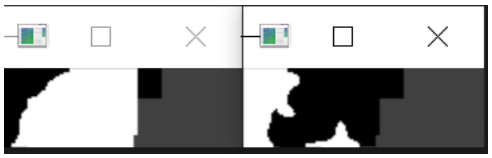
$$S\ obradom\ slike._{sr} = \frac{19.505}{10} = 1.951 \tag{4.8}$$

I dalje se pomoću rezultata vidi da smo s daljnjom pripremom slike za detekciju i praćenje postigli puno bolje rezultate. Nakon testova završavamo s dijelom za praćenje očiju.

Što se tiće parametara blob detektora, nas samo zanimaju svojstva za površinu i boju. Boju postavimo na 0, zato što je to jednako crnoj boji. A površinu smo testirali na način da bi postavili granice, to jest minimum i maksimum i pratili kakva je detekcija bila. Treba imati na umu da površina oka nije velika kada gledamo s obzirom na cijelu sliku. Cilj je bio da se oči mogu detektirati, a da ne dobivamo neku dodatnu detekciju točaka oko šarnice i slično.

Za određivanje strane u koju gledamo smo iskoristili bijeloočnicu. Kao što smo prethodno napomenuli način kako smo ovo riješili sliku oka smo podijelili na pola.

Slika 4.10. Gledanje u desno.



Slika 4.11. Gledanje u lijevo.

Zbog prethodne binarizacije vidimo crni dio koji predstavlja šarenicu i zjenicu u slučaju gledanja da je reprezentabilniji na jednoj strani oka (slika 4.10. i slika 4.11.). Sada se samo izračuna na kojoj strani ima više bijelih piksela. Testiranje nastupa u dijelu s omjerom. Pošto bilo kakvo gledanje u strane mijenja vrijednost, trebamo odrediti granice za desnu i lijevu stranu. Uzimaju se desni i lijevi kraj zaslona kao granice za omjer.

```
Vrijeme: 0.43393079999999973 || Omjer: 0.9788707278577066
Vrijeme: 0.5898336999999998 || Omjer: 1.014565904443205
Vrijeme: 0.6933568000000001 || Omjer: 0.8534725602112
Vrijeme: 0.7901347999999997 || Omjer: 0.9976228859512853
Vrijeme: 0.8882594999999993 || Omjer: 0.858768154922001
Vrijeme: 0.9822848999999998 || Omjer: 0.858768154922001
Vrijeme: 1.0693947999999995 || Omjer: 1.0450783328406739
Vrijeme: 1.1578286999999996 || Omjer: 0.8871216617210682
Vrijeme: 1.2499763000000002 || Omjer: 0.897935691318328
Vrijeme: 1.3388511999999997 || Omjer: 0.8496625923096511
Vrijeme: 1.4307659 || Omjer: 0.8496625923096511
Vrijeme: 1.5183286999999996 || Omjer: 0.8919998804614189
Vrijeme: 1.6087070999999993 || Omjer: 0.8637590677895439
Vrijeme: 1.7010205000000003 || Omjer: 0.8789578312904884
Vrijeme: 1.7905487000000004 || Omjer: 0.9273811136702345
Vrijeme: 1.8797730000000001 || Omjer: 0.9273811136702345
Vrijeme: 1.9688546000000002 || Omjer: 0.8609080248801877
Vrijeme: 2.0571506999999993 || Omjer: 0.9710284174006572
Vrijeme: 2.1469410999999994 || Omjer: 0.8246623026342418
Vrijeme: 2.2417979 || Omjer: 0.8246623026342418
```

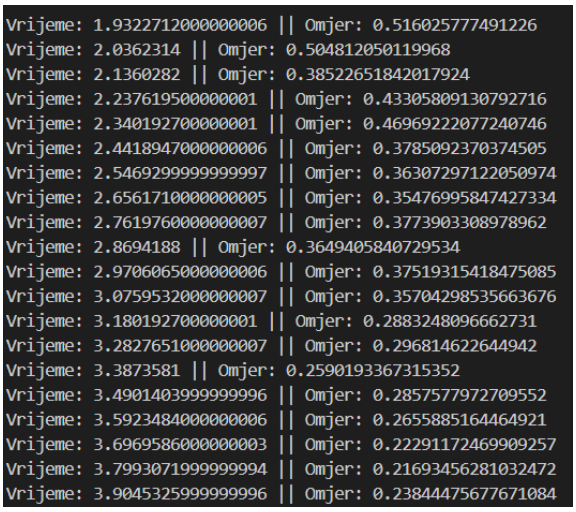
Slika 4.12. Uzorci gledanja u sredinu.

Ovdje se samo nalazi primjer omjera kada gledamo ravno u sredinu zaslona (slika 4.12.). Najmanja vrijednost je 0.824, a najveća 1.014. Možemo slobodno reći da vrijednost varira od 0.8 – 1.0 s mogućim odstupanjima.

```
Vrijeme: 3.6426350000000003 || Omjer: 1.8230493377291919
Vrijeme: 3.7477117 || Omjer: 1.7353760114231318
Vrijeme: 3.8474652999999996 || Omjer: 1.7560065163264402
Vrijeme: 3.9487018999999997 || Omjer: 1.8792623696549793
Vrijeme: 4.0521533000000005 || Omjer: 1.6424557054154252
Vrijeme: 4.152693899999999 || Omjer: 1.8073470338242998
Vrijeme: 4.253991599999999 || Omjer: 1.6392917617610248
Vrijeme: 4.3590429 || Omjer: 1.8225252349796381
Vrijeme: 4.460425799999999 || Omjer: 1.9950634832833785
Vrijeme: 4.5624951 || Omjer: 1.7909953613540597
Vrijeme: 4.667063300000001 || Omjer: 1.7239631838627223
Vrijeme: 4.7704983 || Omjer: 1.825151166138785
Vrijeme: 4.8811737 || Omjer: 1.8374280705845123
Vrijeme: 5.0053886 || Omjer: 1.993053334754646
Vrijeme: 5.118932900000001 || Omjer: 2.1365200515354634
Vrijeme: 5.2229727 || Omjer: 2.180365877613584
Vrijeme: 5.3275276 || Omjer: 3.007471619647072
Vrijeme: 5.426579800000001 || Omjer: 2.9714433832763807
Vrijeme: 5.5280966 || Omjer: 2.7785714285714285
Vrijeme: 5.6331521 || Omjer: 3.625178867122817
```

Slika 4.13. Uzorci gledanja u lijevu stranu zaslona.

Sada smo postepeno s očima gledali u lijevu stranu zaslona i izvan njega (slika 4.13.). Najmanja vrijednost je 1.756, a najveća je 3.625. Primjećujemo u zadnjem uzorku skokovit porast vrijednosti. U ovom slučaju možemo slobodno reći da vrijednosti variraju od 1.7 – 3.6.



Slika 4.14. Uzorci gledanja u desnu stranu zaslona.

Zadnji test je od gledanja u desnu stranu zaslona i izvan njega (slika 4.14.). Minimalno postignuta vrijednost je 0.216, a maksimalna vrijednost je 0.516. Ovdje raspon vrijednosti je od 0.2 do 0.5. Zapamtimo da dijelimo broj piksela na bijeloj strani s brojem na desnoj, jednako tome je dobiveni omjer.

4.2.3. Detekcija treptanja i omjer otvorenosti oka

Testiranje ovih radnji je bilo jednostavno. S informacijom za omjer oka morali smo samo testirati koja je njegova vrijednost kada osoba zatvori oko. I onda postaviti granicu između otvorenog i zatvorenog oka. U sklopu toga se testiralo 10 puta zatvaranjem očiju da dobijemo vrijednost omjera zatvorenog oka.

Broj mjerenja:	Vrijednost omjera:
1.	0.08
2.	0.09
3.	0.09
4.	0.08
5.	0.10
6.	0.09
7.	0.09

8.	0.08
9.	0.07
10.	0.07

Tablica 4.5. Vrijednosti omjera zatvorenih očiju

$$Omjer\ zatvorenog\ oka._{sr} = \frac{0.84}{10} = 0.084 \tag{4.9}$$

Zbog mogućih odstupanja srednjoj vrijednosti treba pridodati konstantu C da bi se lakše ostvarila sama detekcija. S ovim temeljnim konceptom savladanim, trebamo saznati da li postoje drugi elementi koji utječu na promjenu omjera. Prvo smo koristili samo mjeru visine između točaka za praćenje omjera otvorenosti oka. Tu je problem predstavljalo približavanje i udaljavanje od kamere. S time bi se udaljenost između točaka za visinu mijenjala drastično, onda bi izgubili na kvaliteti detekcij treptaja. Jednostavno rješenje je da se dobivena vrijednost visine podijeli sa širinom oka. Obje veličine bi se proporcionalno mijenjale s približavanjem i udaljavanjem osobe, tako bi rezultat za omjer ostajao isti.

Sljedeća stvar je rad s granicom između otvorenog i zatvorenog oka. Dobivena granica i mjera su prihvatljivi u slučaju da ista osoba u sličnim uvjetima koristi program. Treba imati na umu da različite osobe imaju drugačiji omjere visine i širine oka. Spuštenost kapka, veličina oka, otvorenost oka pod drugačijim osvjetljenjem i drugi uvjeti utječu na konačni rezultat. Testiralo se četiri osoba da bi došli do zaključka koliko prethodno spomenuti elementi igraju ulogu. Ispod možemo vidjeti četiri tablice u kojima se nalaze najveća i najmanja vrijednost omjera otvorenosti oka koja je uzorkovana u trenutnom mjerenju.

1. Osoba		
Broj mjerenja:	Vrijednost otvorenog oka	Vrijednost zatvorenog oka
1.	0.304	0.081
2.	0.296	0.093
3.	0.311	0.078

Tablica 4.6. Uzorci minimalnog i maksimalnog omjera kod 1. osobe

2. Osoba		
Broj mjerenja:	Vrijednost otvorenog oka	Vrijednost zatvorenog oka
1.	0.377	0.093
2.	0.359	0.088
3.	0.279	0.104

Tablica 4.7. Uzorci minimalnog i maksimalnog omjera kod 2. osobe

3. Osoba		
Broj mjerenja:	Vrijednost otvorenog oka	Vrijednost zatvorenog oka
1.	0.309	0.107
2.	0.281	0.091
3.	0.325	0.096

Tablica 4.8. Uzorci minimalnog i maksimalnog omjera kod 3. osobe

4. Osoba		
Broj mjerenja:	Vrijednost otvorenog oka	Vrijednost zatvorenog oka
1.	0.251	0.115
2.	0.209	0.073
3.	0.214	0.104

Tablica 4.9. Uzorci minimalnog i maksimalnog omjera kod 4. osobe

Prvo što možemo primjetiti je da i mjerenja istih osoba imaju odstupanja. Vidimo da rezultati variraju kao očekivano. Sada trebamo pronaći srednju vrijednost mjerenja za oba slučaja i nakon toga možemo izračunati odstupanja. Radi jednostavnije usporedbe rezultata kod različitih osoba, za svaku prvo računamo srednju vrijednost.

$$Srednja\ vrijednost\ otv.oko_{1.osoba} = 0.304$$

$$Srednja\ vrijednost\ otv.oko_{2.osoba} = 0.338$$

$$Srednja\ vrijednost\ otv.oko_{3.osoba} = 0.305$$

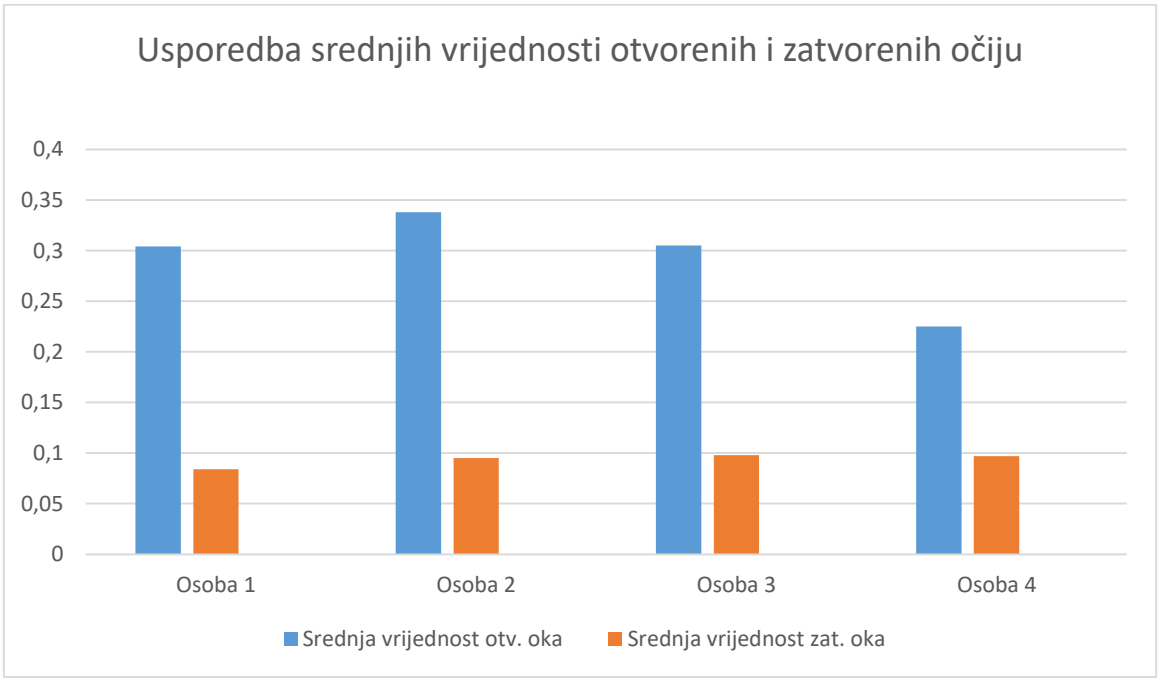
$$Srednja\ vrijednost\ otv.oko_{4.osoba} = 0.225$$

$$Srednja\ vrijednost\ zat.oko_{1.osoba} = 0.084$$

$$Srednja\ vrijednost\ zat.oko_{2.osoba} = 0.095$$

$$Srednja\ vrijednost\ zat.oko_{3.osoba} = 0.098$$

$$Srednja\ vrijednost\ zat.oko_{4.osoba} = 0.097$$



Grafikon 4.1. Usporedba rezultata mjerenja maksimalnog i minimalnog omjera otvorenoti oka

Slijedi računanje cjelokupne srednje vrijednosti i kvadranog odstupanja.

$$Srednja\ vrijednost\ otv.oka = 0.293$$

$$Srednje\ kvadratno\ odstupanje\ otv.oka\ = \ 0.048$$

$$Srednja\ vrijednost\ zat.oka = 0.094$$

$$Srednje\ kvadratno\ odstupanje\ zat.oka\ = 0.006$$

Po prikazu grafikona i dobivenim vrijednostima možemo vidjeti da vrijednost odstupanja su puno manja za slučaj zatvorenog oka. S ovim potvrđujemo da sama vrijednost omjera otvorenosti oka ovisi i o osobi. Zbog ovoga se implementira element, da pri paljenju programa se mjeri najmanja vrijednost omjera. Zbog sigurnosti i mogućih odstupanaj se toj vrijednosti pridodaje mala konstanta koja osigurava da će se treptaj detektirati.

4.2.4. Testiranje cjelokupnog projekta

Na kraju nam ostaje isprobati funkcionalost cijelog projekta, osigurati da svi dijelovi mogu nesmetano raditi. Prijašnja testiranja su se fokusirala na zaseban rad elemenata, dok se ovdje fokusiramo na kombinirani.

Kroz ovo konačno testiranje su se pronašli problemi koji su utjecali na rad nekih komponenti, no to se ispravilo. Iako osvjetljenje igra i dalje veliku ulogu u detekciji šarenice, osim ako je lice jako slabo osvjetljeno problema nema. Konačni rezultati cjelokupnog rada su zadovoljavajući i sve radi na način na koji je koncipirano. Na skupovima slika ispod (slika 4.15. i slika 4.16.) možemo vidjeti koje su mogućnosti ispitane i kakvi su rezultati.

Slika 4.15. Testiranje rada programa 1.dio.

Slika 4.16. Testiranje rada programa 2.dio.

5. ZAKLJUČAK

Zadatak ovog završnog rada je bio realizirati praćenje pokreta oka i detekciju treptanja korištenjem kamere. Pravljenje, testiranje i izvođenje rada se radilo uz pomoć laptopa i njegove kamere. Rad se izveo kombinirajući resurse iz otvorenih izvora i primjenjivajući vlastite ideje i rješenja na prepreke koje su se pojavljivale. Potrebno je bilo upoznavanje s područjem računalnog vida, te s raznim metodama koje se primjenjuju u obradi slike i za detekciju tijela. Shvatiti na koji način funkcioniraju određene metode, da bi ih mogli što bolje koristiti. Zbog mnogobrojnih elemenata koji utječu na konačne rezultate, bilo je potrebno izvoditi puno testiranja i uspoređivanja.

Prvi zadatak je bio osigurati detekciju lica. Pronalazak dlibovog istreniranog modela je bio veliki korak. Rezultati detekcije i praćenja su bili dobri, te su karakterisitke lica kao oči bile obilježene. Nakon toga su se oči izdvojile. To područje se pripremalo za daljnju detekciju. Na dobivenu sliku se implementirao Blob algoritam za detekciju. Poslije dosta testiranja i izmjena, došli smo do zadovoljavajućeg rezultata za detekciju. Za detekciju treptanja se primjenjuje metoda s omjerom oka koja donosi dobar ishod. S uspješnim mjerenjem treptaja smo nadovezali miš i napravili da je radnja treptaja jednaka kliku miša. S napravljenim zadaćama rada jedino što je preostalo je poboljšati funkcionalnost programa i dodati tekstualne elemente koji informiraju o izvršenim radnjama.

LITERATURA

[1] *„300 Faces In-the-Wild Challenge“*, https://ibug.doc.ic.ac.uk/resources/300-W/, Pristup: 14.8.2022.

[2] *68 točaka za označavanje lica i dlib detektor*, https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/, Pristup: 14.8.2022.

[3] *Open Cv biblioteka*, https://opencv.org/about/, Pristup: 14.8.2022.

[4] *Izdvajanje područja interesa*, https://pyimagesearch.com/2021/01/19/opencv-bitwise-and-or-xor-and-not/, Pristup: 14.8.2022.

[5] Tereza Soukupova, Jan Čech, *„Real-TimeEye Blink Detection using Facial Landmarks“*, https://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf, 5.2.2016., Rimske Toplice, Slovenija.

[6] *Blob algoritam za detekciju*, https://learnopencv.com/blob-detection-using-opencv-python-c/, Pristup: 14.8.2022.

[7] *Prepoznavanje lica*, https://pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/, Pristup: 4.9.2022.

[8] *Detekcija plave boje*(slika), https://i.ytimg.com/vi/sB9DjPH6Xmg/maxresdefault.jpg, Pristup: 4.9.2022.

[9] *Dijelovi ljudskog oka*, https://plano.co/the-different-parts-of-the-eye/, Pristup: 4.9.2022.

[10] Ang-Cang Phan, Ngoc-Hoang-Queyen Nguyen, Thanh-Ngoan Trieu i Thuong-Cang Phan, *„An Efficient Approach for Detecting Driver Drowsiness Based on Deep Learning“, https://www.mdpi.com/2076-3417/11/18/8441/htm,* 11.9.2021, Vinh Long, Vijetnam.

[11] RGB i Grayscale modeli, https://www.baeldung.com/cs/convert-rgb-to-grayscale, Pristup: 22.9.2022.

[12] Metode binarizacije slike, https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html, Pristup: 22.9.2022.

SAŽETAK

U okviru teme završnog rada je napravljen program koji prati kretanje očiju, te vrši detekciju treptanja koju povezuje s klikom miša. Za snimanje koristimo kameru laptopa. Bazira se na istreniranom Dlib modelu koji vrši detekciju i praćenje lica pomoću 68 točaka. Odvojili smo područje očiju od ostatka, jer samo s njima radimo. Primjenjivali smo metode obrade slike da bi mogli izdvojiti šarenicu od ostatka oka, radi primjene detektora. Korišten je Blob algoritam za detekciju i praćenje šarenice oka. Uz to također pratimo da li osoba gleda lijevo ili desno. Detekciju treptanja smo realizirali pomoću mjerenja omjera visine i širine točaka oko očiju. Kada se uvjeti koji opisuju treptaj ispune, program registrira treptaj. Provjeravamo koordinate miša, u slučaju da se miš nalazi na području postavljenog kvadrata kad osoba trepne, detektira se klik miša i kvadrat mijenja boju. Prozor koji prikazuje snimani sadržaj kamere, također prikazuje tekstualne i brojčane elemente koji bilježe događaje.

Ključne riječi:

Blob detektor ,detekcija pokreta očiju, detekcija treptanja, Dlib detektor, obrada slike

ABSTRACT

Title: Eye tracking system

The focus of this final paper is a program that was created to monitor the movement of eyes and to detect blinking, which it connects to a mouse click. We use a laptop camera for recording. The project is based on a trained Dlib model that performs face detection and tracking using it's 68 landmarks feature. The eye area is separated from the rest, because we only need to work with the eyes. We applied image processing methods to be able to separate the iris from the rest of the eye, in order to apply the planned detector. The Blob algorithm was used to detect and track the iris. In addition, we also monitor whether the person is looking left or right. Blink detection was realised by measuring the ratio of the height and width of the points around the eyes. When the conditions describing the blink are met, the program registers the blink. The coordinates of the mouse are being checked, in case the mouse is in the area of the set square when the person blinks, the mouse click is detected and the square changes color. The window that displays the recorded content also has text and number elements that record events.

Keywords:

Blob detector, blink detection, Dlib detector, eye movement detection, image proccesing

ŽIVOTOPIS

Autor ovog završnog rada, David Varoščić student je Sveučilišta Josipa Jurja Strossmayera u Osijeku na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. Rođen je 19.2.2000. u Osijeku. Završio je osnovnu školu „Mladost“ u Osijeku. Nakon toga je išao u srednju „Elektrotehničku i prometnu školu Osijek“ gdje je išao na smjer Tehničara za mehatroniku. Godine 2019. započinje školovanje na FERIT-u, stručni studij Elektrotehnike, smjer Automatika.

_____

Potpis autora

PRILOZI I DODACI

U ovom dijelu će se nalaziti sam kod programa i na detaljan način će se objasniti rad upotrebljenih elemenata. Kako je program postavljan tako se i izvodi. Komentari su označeni s „hash - #“ znakom.

```
# IMPORTANJE BIBLIOTEKA I NJIHOVIH ELEMENATA ZA KORIŠTENJE U PROGRAMU

import cv2
from cv2 import ADAPTIVE_THRESH_GAUSSIAN_C, ADAPTIVE_THRESH_MEAN_C,
adaptiveThreshold, fillConvexPoly
import mouse
import dlib
import numpy as np
import math
import random
import imutils
from imutils import face_utils
from timeit import default_timer as timer
----------------------
#OVDJE SE NALAZE FUNKCIJE KOJE KORISTIMO U GLAVNOM PROGRAMU

# Funkcije koje pomažu s izvođenjem programa

# initBlobDetector() služi za postavljanje parametara i inicijalizaciju blob
#algoritma

def initBlobDetector():

    params = cv2.SimpleBlobDetector_Params()

    params.filterByArea = True
    params.maxArea = 500
    params.minArea = 65

    params.filterByCircularity = False
    params.minCircularity = 0.3
    params.maxCircularity = 1

    params.filterByInertia = False
    params.filterByConvexity = False

    params.filterByColor = True
    params.blobColor = 0

    blob_detector = cv2.SimpleBlobDetector_create(params)

    return blob_detector
-----------------------
# shapeToNp() pretvaramo vrijednost 68 točaka za lice dlibovog modela u (x,y)
#koordinate kojima se dalje možemo služiti u programu

def shapeToNp(shape, dtype="int"):

   coords = np.zeros((68, 2), dtype=dtype)
```

```
   for i in range(0, 68):
     coords[i] = (shape.part(i).x, shape.part(i).y)

   return coords


# getEyeAspectRatio() vrši proračun za omjer oka koji je osnova na kojoj se
#temelji detekcija treptanja.

#NAPOMENA: pogledati formulu 3.1.

def getEyeAspectRatio(eye):
    A = math.dist(eye[1], eye[5]) # udaljenost od točaka 1 i 5
    B = math.dist(eye[2], eye[4]) # udaljenost od točaka 2 i 4
    C = math.dist(eye[0], eye[3]) # udaljenost od točaka 0 i 3

    ear = (A+B) / (2*C)

    return ear

-------------------------
# getMiddlePoint() računa središte oka u osnovi na točke modela

def getMiddlePoint(eye):


    top_point = (eye[1] + eye[2]) / 2 # pronalazi središte između točke 1 i 2
    bot_point = (eye[5] + eye[4]) / 2 # pronalazi središte između točke 4 i 5

    mid_point = (top_point + bot_point) / 2 # računa središte oka


    return mid_point
-----------------------

# maskOn() funkcija se bavi segmentiranjem slike, gdje odvajamo oči od
#ostatka. Napravili smo crnu masku istih dimenzija kao slika. I pošto imamo
#našu konveksnu ljusku, trebamo samo ispuniti to područje. Onda to na kraju
#nadovezujemo s bitwise_and operatorom.

def maskOn(img,shape,left_eye_hull,right_eye_hull):

    mask = np.zeros(img.shape[:2],dtype = "uint8")
    kernel = np.ones((9,9), np.uint8)
    mask = cv2.dilate(mask, kernel, 5)

    left = fillConvexPoly(mask,left_eye_hull,255)
    right = fillConvexPoly(mask,right_eye_hull,255)
    mask = cv2.bitwise_and(img,img, mask = mask)

    return mask
-----------------------


# polyArea() računa površinu područja konveksne ljuske koja obuhvaća oko.
#Uzima točke za oči,
#Trenutno ova funkcija nema neku daljnju svrhu, no pomoću nje bi mogli
#računati recimo udaljenost očiju od kamere.

def getPolyArea(hull_pts):
```

```
    lines = np.hstack([hull_pts,np.roll(hull_pts,-1,axis = 0)])
    area = 0.5 * abs(sum(x1*y2-x2*y1 for (x1,y1),(x2,y2) in lines))
    return area

----------------------
# getGazeRatio() s njim dobivamo omjer bijelih piksela na stranama očiju, po
#tome određujemo gleda li osoba L ili D. Uzimamo koordinate oka i s njima
#dobivamo visinu i širinu oka. Binariziramo sliku oka da imamo samo crne i
#bijele piksele, te dijelimo oko po širini na pola. I uspoređujemo bijele
#piksele s cv2.countNonZero() metodom. Zadnje dijelimo lijevu stranu oka s
#desnom stranom i tako dobivamo omjer.


def getGazeRatioNew(eye):
    min_x = np.min(eye[:,0])
    max_x = np.max(eye[:,0])
    min_y = np.min(eye[:,1])
    max_y = np.max(eye[:,1])

    eye = gray[min_y : max_y, min_x : max_x]
    eye = cv2.resize(eye,None,None,fx = 4, fy = 3)
    _,eye = cv2.threshold(eye,127,255,cv2.THRESH_BINARY)
    height,width = eye.shape

    left_side_tresh = eye[0:height, 0:int(width/2)]
    left_side_white = cv2.countNonZero(left_side_tresh)

    right_side_thresh = eye[0:height, int(width/2):width]
    right_side_white = cv2.countNonZero(right_side_thresh)

    try:
        gaze_ratio = left_side_white / right_side_white

    except ZeroDivisionError:

        gaze_ratio = 0

    return gaze_ratio

----------------------------

# earAdaptation() služi za uzimanje omjera očiju i sprema vrijednosti u
#listu. Traži se najmanji omjer i njemu se dodaje mala konstanta zbog
#odstupanja. Na kraju izbacuje vrijednost koju koristimo kao granicu za
#detekciju treptaja.


def earAdaptation(ear):

    ear_list.append(ear)
    min_value = min(ear_list)
    min_value += 0.03

    return min_value
-----------------------------
# DEKLARIRANJE I INICIJALIZACIJA VARIJABLI
# Postavljamo vrijednosti koje se koriste za blinkanje

# Varijable vezane za detekciju treptanja

EYE_BLINK_THRESH = None
```

```
EYE_AR_CONSEC_FRAMES = 1

COUNTER = 0
TOTAL = 0

ear_list = []
ear_adaptation = False
ear = 0.0
eye_blink = None

double_click = None


# Izbor boja za kvadrat pri registraciji klika
BGR = ((255,0,0),(0,255,0),(0,0,255),
(255,255,0),(255,0,255),(0,255,255),(0,0,0),(255,255,255),
    (100,100,100),(200,200,200))

#inicijalizacija timer-a
timer_start = 0


# Inicijaliziramo detektor i prediktor
detector= dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

# Pozivamo funkciju za Blob detection
blob_detector = initBlobDetector()

# Izvlačimo potrebne koordinate za točke oko očiju
(left_start, left_end) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(right_start, right_end) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]

# Pripremamo kameru

cap = cv2.VideoCapture(0)


------------------------
#ULAZIMO U GLAVNU PETLJU PROGRAMA

while(True):

    #Nastavljamo kameru koristiti
    ret,img = cap.read()

    # Postavljamo kolike će biti dimenzije prozora
    img = imutils.resize(img, width = 600, height=400)


    # Sliku pretvaramo u grayscale područje
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    #Detektoru implementiramo na sliku naše kamere
    rects = detector(gray, 0)

     #Provjeravamo da li je detektor pronašao lice
     # U slučaju da nije nastavljamo snimanje, provjeravamo stanje
     # i obavještavamo korisnike da se lice ne može detektirati.
    while len(rects) == 0:
        ret,img = cap.read()
```

```
        img = imutils.resize(img, width = 600, height=400)


        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

        rects = detector(gray, 0)


        if len(rects) == 1:
            break
        elif cv2.waitKey(1) & 0xFF == ord('q'):
            break

       cv2.putText(img,"Pronalazak lica u tijeku!",
     (220,100),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)
        cv2.imshow("Eye tracking system", img)

     #Potrebno za implementiranje prediktora
    for rect in rects:

        shape = predictor(gray, rect)

        shape = shapeToNp(shape)

     #Pravljenje zasebnih varijabli za oči
     try:
        left_eye = shape[left_start : left_end]
        right_eye = shape[right_start : right_end]
    except NameError:
        left_eye = None
        right_eye = None

     # Iscrtavanje svih točaka dlib modela na licu

    #for (x,y) in shape:
    #    cv2.circle(img,(x,y),1,(0,0,255),-1)

     # Pozivamo funkciju za dobivanja središta očiju
     left_mid_p = getMiddlePoint(left_eye)
     right_mid_p = getMiddlePoint(right_eye)

     # Pravimo konveksne ljuske oko očiju
     left_eye_hull = cv2.convexHull(left_eye, None, None, returnPoints=True)
    right_eye_hull = cv2.convexHull(right_eye, None, None, returnPoints=True)


     #Pozivamo funkcije za računanje površine očiju
     left_eye_area = getPolyArea(left_eye_hull)
    right_eye_area = getPolyArea(right_eye_hull)

     #Koristimo maskOn funkciju za odvajanje očiju od ostatka
     mask = maskOn(gray,shape,left_eye_hull, right_eye_hull)

     #Primjenjujemo kombinaciju morfoloških operacija koje služe za micnanje
#crnih točkica koje se nalaze na predmetu obrade
     closing = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, (3,3))


     # Primjena adaptiveThresholda koji pojednostavljuje sliku za daljnju
#obradu binarizacijom,
```

```
    # koristi metodu računanja srednje vrijednosti
    eye_thresh_adaptive =
adaptiveThreshold(closing,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,3,
2)



     # Daljnja obrada slike radi primjene Blob detektora
    eye_erode = cv2.erode(eye_thresh_adaptive, None, iterations= 1)#1
    eye_dilate = cv2.dilate(eye_erode, None, iterations= 2)#2
    eye_gaus = cv2.GaussianBlur(eye_dilate,(5,5),0)


     # Implementiramo blob detektor na sliku i crtamo rezultate detekcije
    keypoints_adaptive = blob_detector.detect(eye_gaus)
     drawBlob = cv2.drawKeypoints(img,
keypoints_adaptive,img,(0,0,255),cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)

     # Računamo omjere bijelih piskela bjeloočnice na stranam svakog oka,
nakon
     #toga računamo njihovu srednju vrijednost
     gaze_ratio_left = getGazeRatioNew(left_eye)
    gaze_ratio_right = getGazeRatioNew(right_eye)

    gaze_ratio = (gaze_ratio_left + gaze_ratio_right) / 2


     # Stavlja se crta na prozoru za odvajanje dva tekstualna elementa
     cv2.putText(img,"-------"
,(10,100),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,0,255),2)

     # Tu se provjerava koliki je omjer bijelih pikesla očiju i ovisno o
#uvjetima ispisuje gleda li osoba na LIJEVU ili DESNU stranu
    if ear > 0.22:
        if gaze_ratio > 1.3:
            cv2.putText(img,"Gledanje
Lijevo.",(10,80),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)
        elif gaze_ratio < 0.7:
            cv2.putText(img,"Gledanje
Desno.",(10,80),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)



  # DETEKCIJA ŽMIRENJA & KLIKANJE MIŠ


     # Uzimamo omjere otvorenosti očiju koje nam računaju funkcije
    left_EAR = getEyeAspectRatio(left_eye)
    right_EAR = getEyeAspectRatio(right_eye)
     # Računamo srednju vrijednost oba oka i postavljamo vrijednost na 4
#decimale
    ear = (left_EAR + right_EAR) / 2.0
    ear = round(ear,4)



     #  Prvo provjeravamo je li korak za adaptaciju obavljen.
     #  Ako nije ispisuje se tekst da ga treba obaviti.
     #  Ako je onda se uzima granica za treptaj i može se započeti s
        detekcijom.
     if ear_adaptation == False:
```

```
        cv2.putText(img, "Trepnite nekoliko puta", (150,150), # 120
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        cv2.putText(img, "i pritisnite 'S' nakon toga.", (150,170), # 120
        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        min_value = earAdaptation(ear)

     elif ear_adaptation == True:
        EYE_BLINK_THRESH = min_value
        # Ovdje se nalazi logika za provjeru je li osoba trenpnula.
       # Ako je omjer < postavljenje granice brojač se pokrene, nakon što
   #omjer dođe iznad granice provjerava se vrijednost brojača i ako su uvjeti
   #ispunjeni radnja se računa kao treptaj.
   # U međuvremenu se računa vremenska razlika između svaka 2 treptaja i ako
   #je vrijeme < 0.8 sekundi to će se kasnije uzeti kao da je ekvivalentno
   #dvokliku miša.
        if ear < EYE_BLINK_THRESH:
            COUNTER += 1

        else:

            if COUNTER >=  EYE_AR_CONSEC_FRAMES and COUNTER <= 10:
                TOTAL += 1
                eye_blink = True
                if timer_start == 0:
                    start = timer()

                timer_start += 1

            COUNTER = 0

        if timer_start == 2:
            timer_start = 0
            end = timer()
            first = True

            if  (end - start) <= 0.8:
               double_click = True




     # Postavljanje koordinata za kvadrat
    pt1 = (500, 300)
    pt2 = (600,400)

    if TOTAL == 0:
        rand_clr = None

    # Uzimamo koordinate miša
    mouse_cords = mouse.get_position()

   # Radi se provjera je li miš na koordinatama kvadrat
    mouse_in_region = False
    if (mouse_cords[0] >= 500 and mouse_cords[0] <= 600 ) and (mouse_cords[1]
>= 300 and mouse_cords[1] <= 400):
        mouse_in_region = True
    else:
        mouse_in_region = False
```

```
    # Provjerava se da li je osoba trepnula i ako je miš na koordinatama
#kvadrata. Ako su uvjeti ispunjeni ispituje se je li bio klik ili dupli
#klik. Ispisuje se na prozor tekst o ćemu se radi i bira se druga boja za
#kvadrat.
    if eye_blink == True and mouse_in_region == True:

        if double_click == True:
            cv2.putText(img, "Dupli Klik", (10,125), # 120
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)


        else:
            cv2.putText(img, "Klik", (10, 125),
           cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

        rand_clr = random.choice(BGR)



    # Iscrtava se kvadrat
    rectag = cv2.rectangle(img,pt1,pt2,rand_clr,-1)

    eye_blink = False

    # Ispisujemo broj detektiranih treptaja
    cv2.putText(img, "Broj treptaja: {}".format(TOTAL), (10, 30),
  cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

   double_click = False
    # Dopušta prikazivanje prozora, dok ne stisnemo određenu tipku
   key = cv2.waitKey(1) & 0xFF

   if TOTAL == 0:
       viewData = False
    # Ako smo stisnuli tipku 'w' prikazujemo dodatne informacije, ako je opet
#stisnemo nakong toga maknemo ih
   if key == ord('w') and viewData == False :
       viewData = True
   elif key == ord('w') and viewData == True:
       viewData = False

    # Ako smo stisli tipku 's' izvršava se postavljanje granice između
#otvorenog i zatvorenog oka
   if key == ord('s'):
       ear_adaptation = True

    # Prikazujemo sve navedeno informacije ovisno o stanju varijable
#„viewData"
    if viewData == True:
       cv2.putText(img, "OMJER OKA: {:.2f}".format(ear), (300, 30),
       cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
       cv2.putText(img,"OMJER POGLEDA: {:.2f}".format(gaze_ratio),(300,55),
       cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)
       cv2.putText(img, "Lijevo oko (x,y): ({:.2f},
{:.2f})".format(left_mid_p[0], left_mid_p[1]), (10, 400),
       cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
       cv2.putText(img, "Desno oko (x,y): ({:.2f},
{:.2f})".format(right_mid_p[0], right_mid_p[1]), (10, 420),
       cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
```

```
     # Prikazujemo prozor koji sadrži snimku kamere i sve prethodno navedene
#elemente
     cv2.imshow("Eye tracking system", img)

     # Ako stisnemo tipku 'q' izlazimo iz glavne petlje
     if key == ord('q'):
        break
     # Prestajemo koristiti kameru i zatvarmo otvoreni prozor
     cap.release()
     cv2.destroyAllWindows()
```