Virtualna šminka

Čavala, Bruno

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

Sveučilišni studij

VIRTUALNA ŠMINKA

Završni rad

Bruno Čavala

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 19.09.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Bruno Čavala
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R 4329, 22.07.2019.
OIB Pristupnika:	46330207974
Mentor:	Izv. prof. dr. sc. Časlav Livada
Sumentor:	.
Sumentor iz tvrtke:	
Naslov završnog rada:	Virtualna šminka
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak završnog rad:	U radu je potrebno korištenjem OpenCV biblioteke u stvarnom vremenu napraviti detekciju lica i značajki lica. Potrebno je napraviti GUI koji će omogućiti dodavanje šminke na odgovarajuće dijelove lica (usne, kapci...).
Prijedlog ocjene završnog rada:	Dobar (3)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 1 bod/boda Jasnoća pismenog izražavanja: 1 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene od strane mentora:	19.09.2022.
Datum potvrde ocjene od strane Odbora:	21.09.2022.

Potvrda mentora o predaji konačne verzije rada:	Mentor elektronički potpisao predaju konačne verzije.
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 22.09.2022.

Ime i prezime studenta:	Bruno Čavala
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 4329, 22.07.2019.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom: **Virtualna šminka**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Časlav Livada

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Proširena stvarnost je pojam koji je poznat već desetljećima u svijetu, ali tek se u novije vrijeme počeo širiti zbog razvoja sve bržih i boljih mobilnih telefona. Proširena stvarnost (engl. *Augmented Reality – AR)* opisuje poboljšani pogled na stvarni život tako što se spaja s računalno generiranim sadržajem. Taj sadržaj može biti bilo kakav trodimenzionalni model, videozapis, slika te također lokacija ili kombinacije više njih. Proširena stvarnost ima brojne praktične primjene u različitim sektorima industrije, kao što su automobilizam, maloprodaja, edukacija, financije i turizam. Prednosti proširene stvarnosti su to što pruža bogatije korisničko iskustvo, povećava vrijednost marki i proizvoda, mobilna je i dostupna brzo rastućem tržištu pametnih telefona, te je jeftina alternativa medijskim platformama. Primjer poboljšanja korisničkog iskustva je virtualna šminka koja se koristi na internet stranicama za prodaju šminke. Omogućuje isprobavanje šminke koristeći internet preglednik gdje korisnik može isprobati šminku prije nego je kupi te se time postiže veća mogućnost za prodaju tog proizvoda.

U radu će se opisati što je sve potrebno za izradu programa za virtualnu šminku. Prvo poglavlje je podijeljeno na dva dijela. U prvom dijelu se će se navesti koji programski jezik je korišten i koja bi njegova alternativa bila. U drugom dijelu će se opisati koji se *IDE*[1] koristio i dodatne biblioteke potrebne za izradu programa. U drugom poglavlju će se opisati potrebni alati za izradu aplikacije te koje su funkcije bile korištene i zašto su korištene. U trećem poglavlju će se spomenuti kako bih se program mogao poboljšati i koji su mu nedostaci.

1.1. Zadatak završnog rada

Zadatak završnog rada je napisati program koristeći *OpenCV* biblioteku. Program je u ovom slučaju pisan u programskom jeziku *Python*, ali postoje i druge alternative koje se mogu koristiti. Program mora u stvarnom vremenu primijeniti šminku tj. filter na lice osobe koja se nalazi ispred kamere.

[1] IDE – integrirano razvojno okruženje

2. *IDE* i komponente

2.1. *Python*

Python je objektno orijentirani programski jezik visoke razine. Njegove prednosti su to što je relativno lagan jezik za naučiti te je jedan od razloga zašto je korišten za izradu ovog programa. Drugi razlog je to što su biblioteke koje su potrebne za izradu programa kompatibilne s *Pythonom* logo na slici 2.1.

Slika 2.1. Logo Python [1]

Alternative koje su kompatibilne s bibliotekama potrebnim za izradu programa su programski jezici *C++* i *Java* koji su teži za naučiti za razliku od *Pythona*, ali su snažniji jezici. Logo programskih jezika su prikazani na slici 2.2.

Slika 2.2. Logo C++ i Java [10] [11]

2.2. *PyCharm IDE*

PyCharm je *IDE*, logo je prikazan na slici 2.3. Koji se koristi u računalnom programiranju najčešće za programski jezik *Python*. Ovaj program je odličan za programiranje u *Pythonu* jer je kod pregledniji i lakši za čitati te ima mogućnost automatskog popunjavanja teksta što omogućuje brže i točnije pisanje koda. Također je odličan jer ima repozitorij biblioteka koji nam omogućuje lakše preuzimanje i njihovo korištenje.

Slika 2.3. Logo PyCharm [1]

Biblioteke koje su korištene za izradu ovog programa su:

- *OpenCV*
- *NumPy*
- *Dlib*

OpenCV je biblioteka softvera za računalni vid i strojno učenje otvorenog koda te bez ove biblioteke ovaj program ne bih bilo moguće napraviti. *NumPy* je *Python* biblioteka za rad s nizovima i matricama koja će pomoći u izradi programa. *Dlib* je detektor orijentira lica s unaprijed obučenim modelima. Koristi se za procjenu lokacije 68 koordinata u dvije dimenzije (x,y) koje mapiraju točke lica modela na lice osobe. O ovome će se više pričati u sljedećim poglavljima.

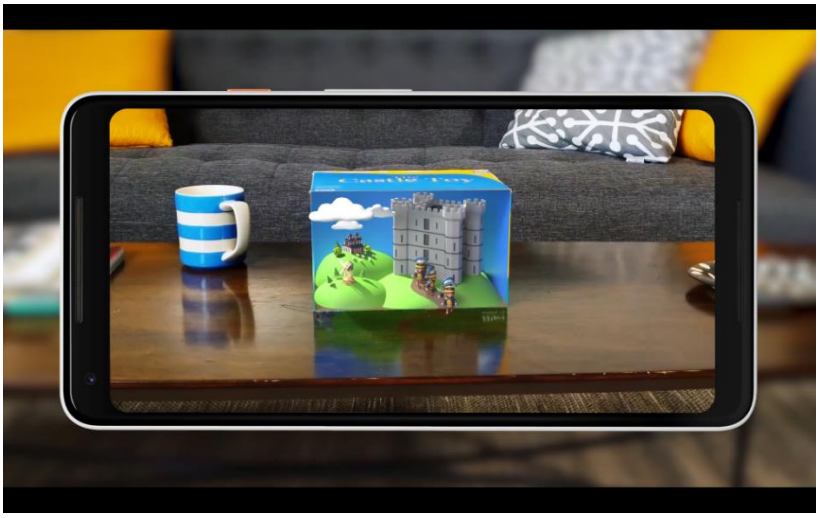
2.3. Postojeća rješenja

Naravno ovaj zadatak se mogao izvršiti pomoću drugih programa i *IDE*-a. Na primjer kao što su:

- *ARCore*
- *Apple ARKit*
- *Unity*

2.3.1. ARCore

ARCore je Google-ova platforma za izgradnju doživljaja proširene stvarnosti. Na slici 2.4. je prikazana aplikacija napravljena pomoću *ARCorea*.



Slika 2.4. Primjer aplikacije napravljene u ARCoreu [6]

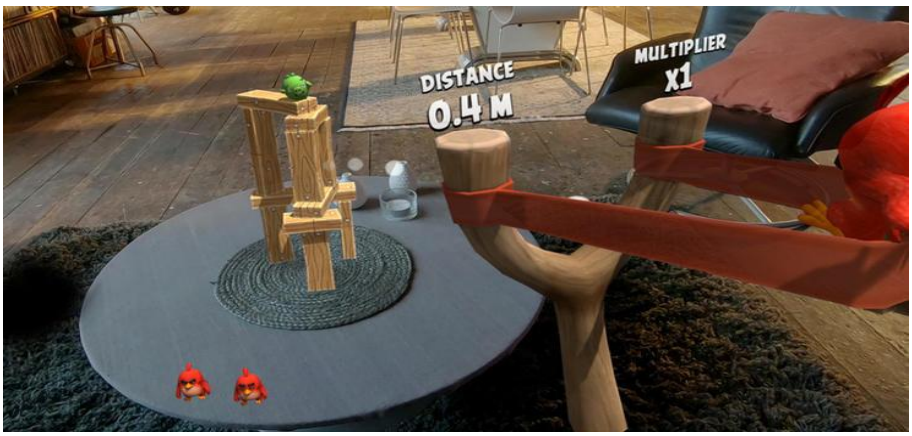
2.3.2. Apple ARKit

Apple ARKit kombinira praćenje kretnji uređaja, snimanje scene kamerom, naprednu obradu scene i pogodnosti prikaza kako bi se pojednostavio zadatak izgradnje AR iskustva. Na slici 2.5. je prikazana aplikacija napravljena pomoću *Apple ARKita*.

Slika 2.5. Primjer aplikacije napravljene u Apple ARKit-u [7]

2.3.3. Unity

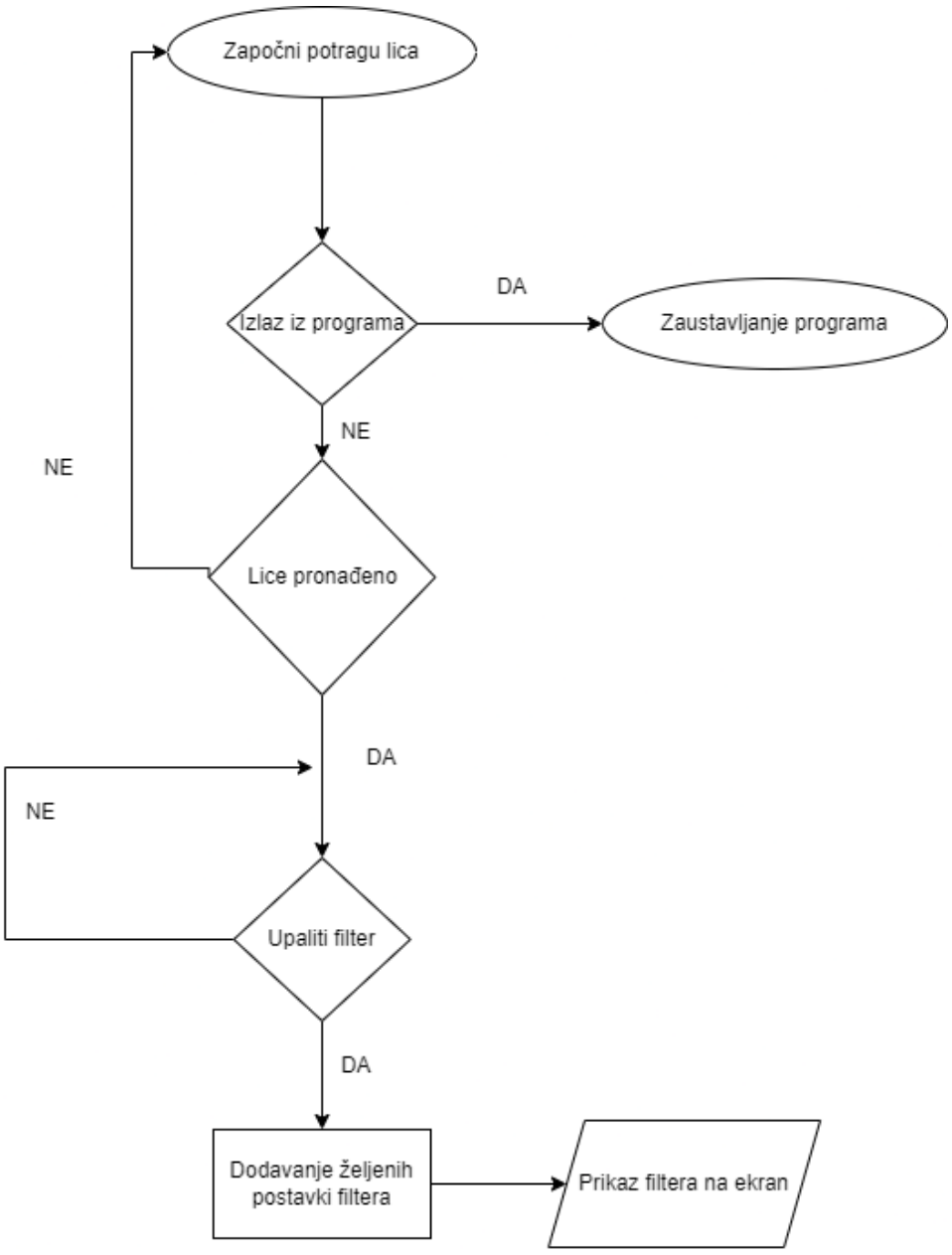
Unity je višeplatformski pogonski sklop za igre koji je razvio Unity Technologies, a koji se prvenstveno koristi za razvoj videoigara i simulacija za računala, konzole i mobilne uređaje, ali može se koristiti u svrhe pravljenja aplikacija za proširenu stvarnost. Na slici 2.6. prikazana je igrica napravljena koristeći *Unity*.



Slika 2.6. Primjer aplikacije napravljene koristeći Unity [8]

3. Razvoj aplikacije za virtualnu šminku

3.1. Dijagram toka



Slika 3.1. Dijagram toka programa

Zbog lakšeg razumijevanja zadatka poželjno je napraviti dijagram toka slika 3.1. kako program radi. Program na početku traži lice osobe koja se nalazi ispred kamere te konstantno gleda je li stisnut gumb za gašenje programa te ako nije nastavlja dalje. Ako je nije pronađeno vraća se na početak. Ako je pronađeno potrebno je postaviti pitanje je li filter uključen ili nije? Ako je dolazi

do postavka za uređivanja filtera te se taj filter prikazuje na ekranu. U sljedećim dijelovima rada će biti objašnjene biblioteke te njihove glave funkcije za izradu programa.

3.2. *GUI*

GUI aplikacije je moguće napraviti koristeći *OpenCV* biblioteku. Pomoću njegovih funkcija može se napraviti dva potrebna prozora koji će jedan biti za postavke filtera, a drugi prozor je zadužen za prikaz kamere u stvarnom vremenu. Funkcije su prikazane na slici 3.2.

```
5   title_window = 'Virtual Makeup'
6   trackbar_window = 'Settings'
7
8   cv.namedWindow(trackbar_window)
9   cv.resizeWindow(trackbar_window,350,250)
10  cv.namedWindow(title_window) #pravljenje novog prozora
```

Slika 3.2. Pravljenje 2 prozora

Također pomoću *OpenCV*-a snimat će se kamera pomoću funkcije na slici 3.3.

```
11  cap = cv.VideoCapture(0,cv.CAP_DSHOW)#pokrece se snimanje videa sa kamere
```

Slika. 3.3. Snimanje kamere

Za postavljanje postavki filtera potrebno je napraviti klizače koje također pravimo pomoću *OpenCV* biblioteke kao što je na slici 3.4.

```
28  cv.createTrackbar('R', trackbar_window, 0, 255, on_trackbar) #pravljenje 4 klizač za boje
29  cv.createTrackbar('G', trackbar_window, 0, 255, on_trackbar)
30  cv.createTrackbar('B', trackbar_window, 0, 255, on_trackbar)
31  cv.createTrackbar('Alpha',trackbar_window,0,100,on_trackbar)
32  switch = 'OFF|ON'
33  cv.createTrackbar(switch, trackbar_window,0,1,on_trackbar)#klizač za paljenje i gašenje filtera
34  mode = 'Mode'
35  cv.createTrackbar(mode, trackbar_window,0,3,nothing) #klizač za alpha vrijednosti
```

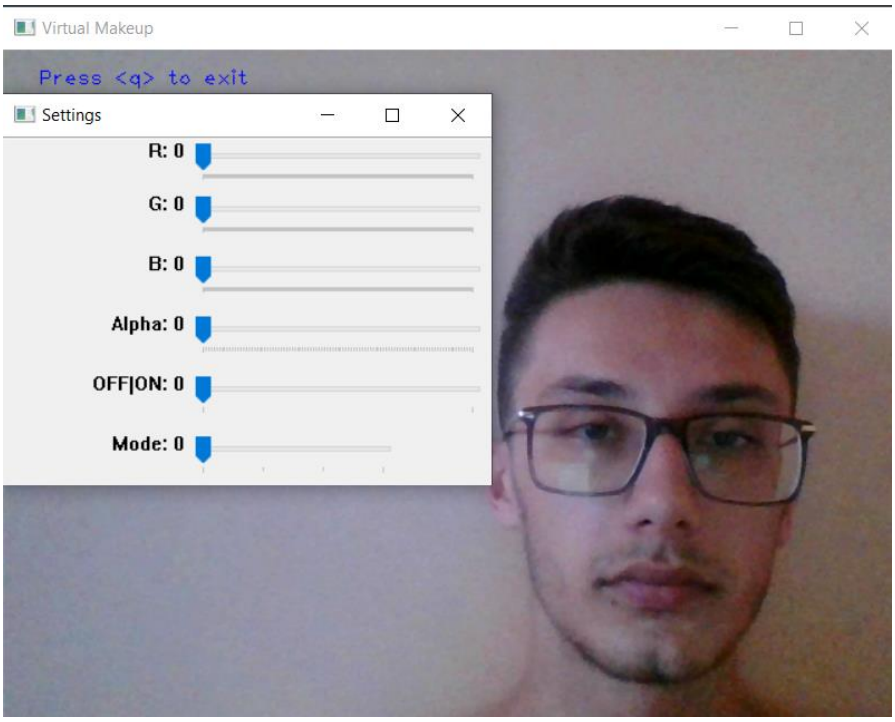
Slika 3.4. Pravljenje klizača za postavke filtera

Dodatno se može napraviti funkciju za izlazak iz aplikacije kojoj je zadatak gledati je li zadano slovo stisnuto dok god aplikacija radi. Ako je stisnuto slovo koje je u ovom slučaju slovo *Q* ostvaren je izlazak iz aplikacije. Kod je prikazan na slici 3.5.

```
43      cv.putText(originalFrame, "Press <q> to exit", (25, 25), cv.FONT_HERSHEY_PLAIN, fontScale = 1, color = (255, 0, 0), thickness = 1)
168     k = cv.waitKey(1)
169     if k & 0xff == ord('q'):
170         break
```

Slika 3.5. Funkcija za izlazak i ispis teksta na ekranu

Dobiveni rezultat bi trebao izgledati kao na slici 3.6. gdje su prikazani prozori za postavke i kameru.

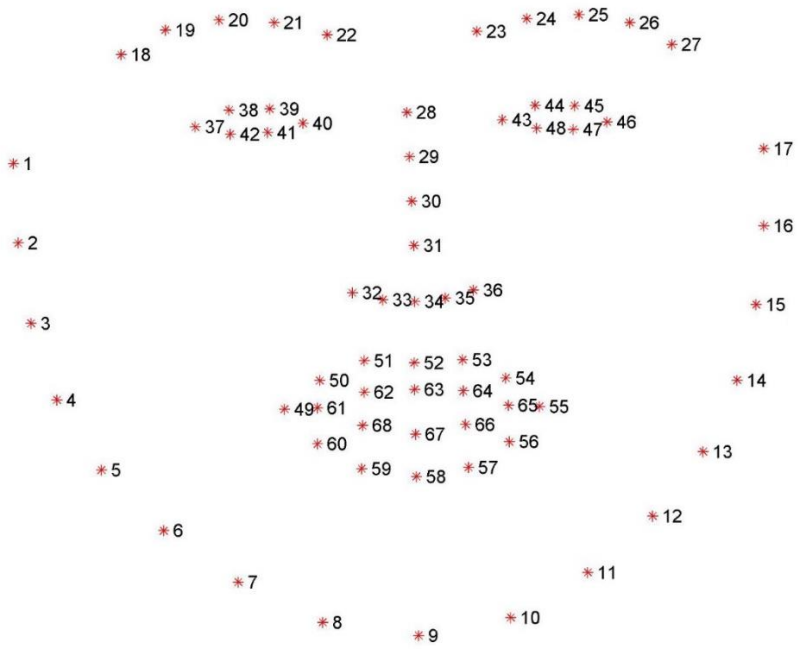


Slika 3.6. Postignuti *GUI* aplikacije *OpenCV* bibliotekom

.

3.3. Prepoznavanje lica

Dlib biblioteka će biti potrebna za prepoznavanje lica osobe ispred kamere i za dobivanje koordinata određenih orijentira. Ovo će se postići koristeći već gotovi model koji sadrži 68 orijentira lica. Na slici 3.7. su prikazani gdje se sve nalaze orijentiri lica.



Slika 3.7. Dlib shape_predictor_68_face_landmarks model lica s orijentirima

Pomoću *Dlib*-a napravit će se detektor lica koristeći već takozvanu istreniranu funkciju za pronalazak lica osobe. Također je potrebno učitati orijentire lica. Kod je na slici 3.8.

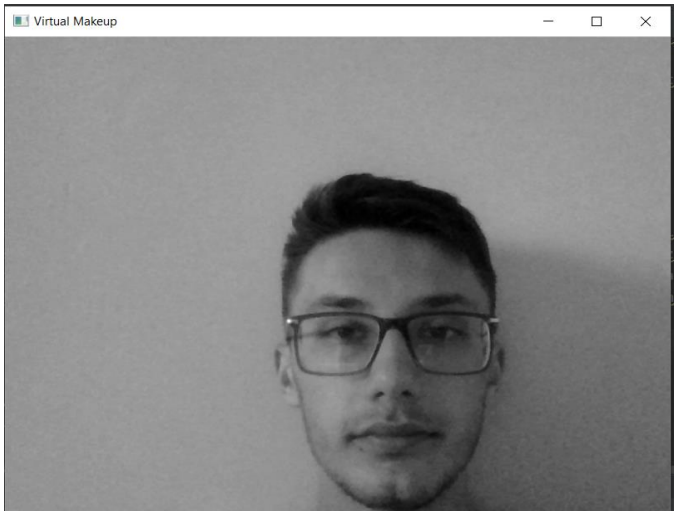
```
13    detector = dlib.get_frontal_face_detector()
14    predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

Slika 3.8. Pravljenje detektora i prediktora

Ovo je sve potrebno staviti u petlju koja će se zauvijek izvršavati dok se ne stisne slovo za izlazak iz aplikacije. Ovim će se dobiti pronalaženje lica u stvarnom vremenu jer će program u svakom trenutku (engl. frame) *frameu*[2] tražiti poziciju lica te pomicati masku kako se pomiče lice osobe ispred kamere. Izgled koda vidi se na slici 3.9.

```
37  while (1):
38
39      ret, frame = cap.read()  # dekodira i vraca frame videa koji je u ovom slucaju web kamera
40      gray_frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)  # pretvara boje tog framea u crno-bijelu boju
41      originalFrame = frame.copy()
42      faces = detector(frame)  # frame predajemo detectoru koji ce pornalaziti lice na tom frameu
```

Slika 3.9. Petlja za pronalazak lica



Slika 3.10. Sivi frame

Također se mijenja boja *framea* u crno-bijelu radi lakšeg pronalaženja dijelova lica kao na slici 3.10 te pravimo kopiju *framea* u boji zbog kasnije upotrebe koja će biti objašnjena u poglavlju 3.6.

[2] Frame – hrv. Okvir – jedna slika videa

Sljedeće što je potrebno napraviti je spremiti sve koordinate orijentira u jedan niz kao što je na slici 3.11.

```
51      for face in faces:
52          landmarks = predictor(gray_frame,
53                                face)  # trazimo orijentire pomocu predictora te mu predajemo sivi frame i koordinate lica
54          LandMarkPoints = []  # polje kojoj cemo predati koordinate svih landmarkova
55      for n in range(68):
56          x = landmarks.part(n).x
57          y = landmarks.part(n).y
58          coordinates = [x, y]
59          LandMarkPoints.append(coordinates)
60
61      npLandMarkPoints = np.array(LandMarkPoints)
```

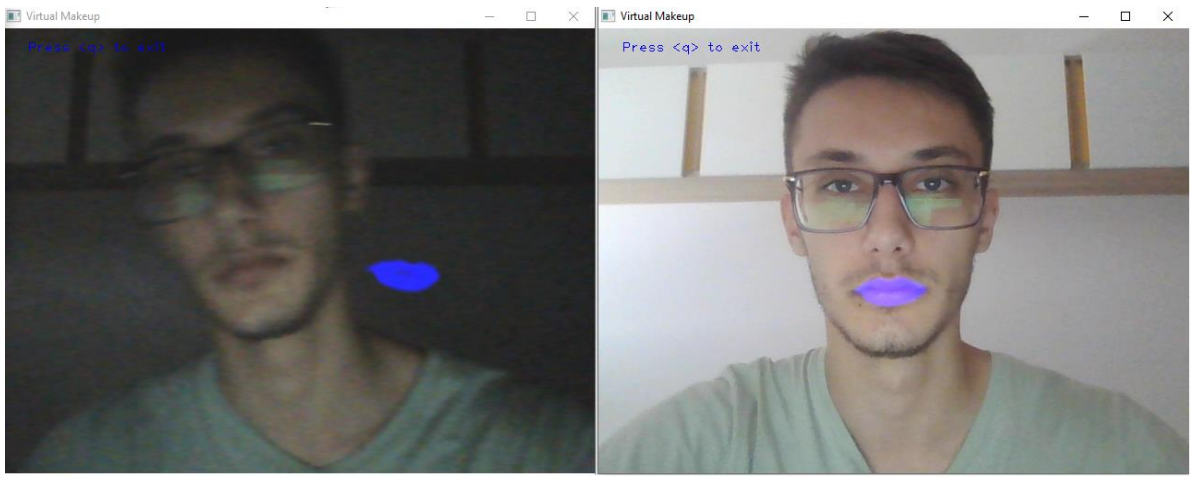
Slika 3.11: Petlja za spremanje koordinata orijentira

To se radi na sljedeći način za svaki dio lica koje prediktor nađe njegove koordinate pohraniti u *landmarks.* U njemu se pohranjuju koordinate u heksadekadskom zapisu. U sljedećoj *for* petlji svakom orijentiru ćemo predati x i y koordinatu i pohraniti taj cijeli niz u *LandMarkPoints.* Funkcija za pravljenje maske ne prima takav format niza te je potrebno koristiti funkciju iz biblioteke *Numpy*.

Nakon pronalaženja svih koordinata orijentira moguće je napraviti masku.

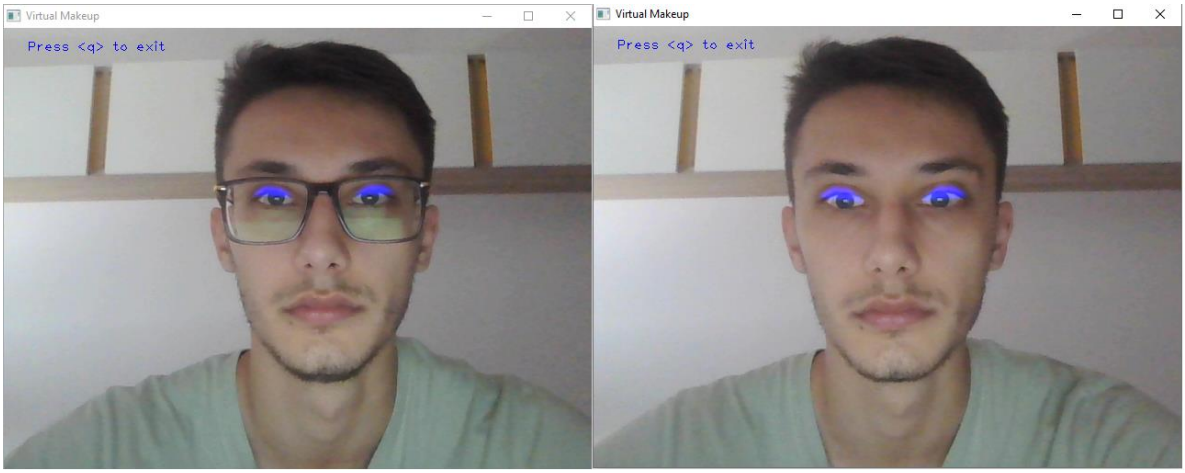
3.4. Izazovi prilikom pronalaska lica

Jedan od izazova prilikom pronalaska lica je nedovoljno osvjetljenje lica gdje program nije sposoban pronaći orijentire lica te maska će ostati lebdjeti u zraku kao na slici 3.12. Lijeva strana slike pokazuje problem s lošim osvjetljenjem, a desna strana s dobrim osvjetljenjem.



Slika 3.12: Pronalazak maske u različitim osvjetljenima

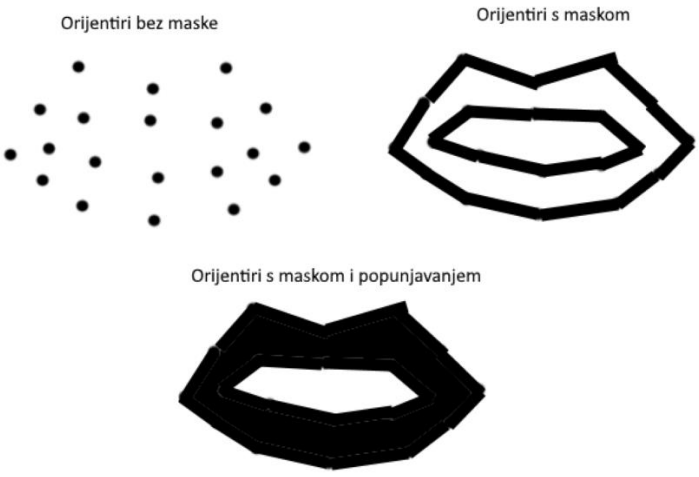
Također jedan od problema koji se može dogoditi je neuspješno pronalaženje očiju zbog naočala. U ovom slučaju je program dovoljno sposoban i dalje prepoznati dijelove lica čak i ako korisnik nosi naočale kao što se može vidjeti na slici 3.13.



Slika 3.13: Pronalazak trepavica s naočalama i bez njih

3.5. Izrada maske

Maska se pravi tako što se uzmu koordinate orijentira te se oni onda spajaju da naprave željeni oblik. Pravljenje maske je prikazan na primjeru usana. Orijentiri usana kao što se može vidjeti na slici 3.7. kreću se od 49 do 68. Funkcija maske uzima koordinate tih orijentira i pravi vanjski oblik maske. Ako želimo ispuniti taj oblik koristi se funkcija *fillPoly*. Na slici 3.14. vidimo kako radi funkcija mask i *fillPoly*.



Slika 3.14: Maska i *fillPoly* funkcija

Boja maske se postavlja na bijelu koristeći aditivni model RGB[3] koji zbraja boje i kao izlaz daje boju ovisno o kombinaciji crvene, zelene i plave. Njihova vrijednost se kreće od 0 do 255. Znači da bi za izlaz dobili bijelu boju koristimo kombinaciju [255,255,255]. Ovo se može vidjeti na slici 3.15.

```
22  def createMask(frame,points,offset): #funkcija za pravljenje maske od predanih točaka
23
24          mask = np.zeros_like(frame)
25          mask = cv.fillPoly(mask,[points],(255,255,255),offset=offset)
```

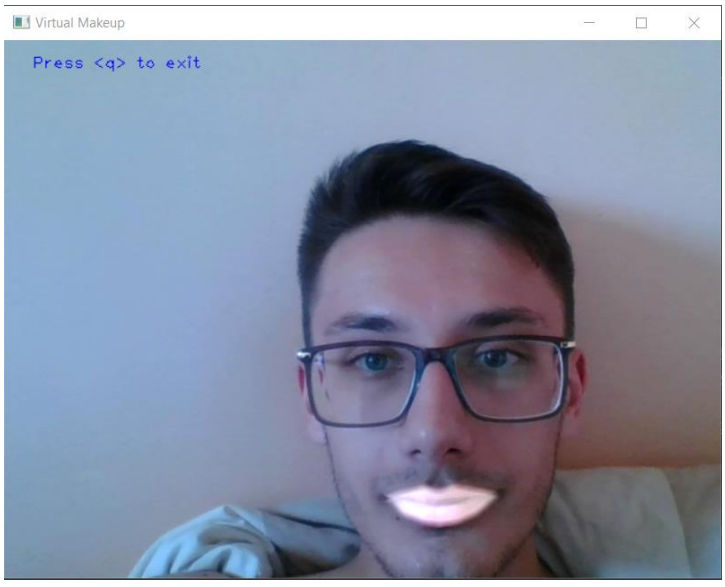
Slika 3.15. Funkcija za pravljenje boje maske

Nakon izrade funkcije za pravljenje maske možemo je pozvati za primjer usana kao na slici 3.16.

```
65  if(trac_mode == 0):
66
67      lipsMask = createMask(frame, LandMarkPoints[48:68],None) #zovemo funkciju za pravljenje maske te predajemo landmarkove namjenjene za usne
68      lipsColor = np.zeros_like(lipsMask) #vraca polje nula ali sa istim tipom i oblikom maske, samo su postavke maske vrijednosti nula
69
```

Slika 3.16. Pozivanje funkcije za izradu maske namijenjene za primjer usana

Varijabla *trac_mode* će biti objašnjena u sljedećim poglavljima.



Slika 3.17. Maska za usne

[3] RGB – engleska kratica za „Red Green Blue“ – aditivni model kod kojeg se zbrajanjem osnovnih boja dobivaju ostale boje

Rezultat pravljena maske možemo vidjeti na slici 3.17.

3.6. Bojanje maske

Masku možemo bojati pomoću klizača. Može se mijenjati crvena, zelena i plava boja i njihove vrijednosti se kreću od 0 do 255. Prvo je potrebno dobavljati stanja vrijednosti klizača, a to je moguće pomoću funkcije *getTrackbarPos* te ta funkcija vraća vrijednost klizača i pohranjuje u varijablu tipa *int*[4]. Ovo se radi za sve 3 boje, transparentnost, uključivanje i isključivanje filtera te način rada filtera. Kod se nalazi na slici 3.18.

```
44  trac_mode = cv.getTrackbarPos(mode, trackbar_window)
45  r = cv.getTrackbarPos('R', trackbar_window)
46  g = cv.getTrackbarPos('G', trackbar_window)
47  b = cv.getTrackbarPos('B', trackbar_window)
48  s = cv.getTrackbarPos(switch, trackbar_window)
49  a = cv.getTrackbarPos('Alpha',trackbar_window)
```

Slika 3.18. Dobavljanje vrijednosti s klizača

Sada se postavlja pitanje kako primijeniti vrijednosti postavki maske na samu masku? Prvo je potrebno napraviti kopiju maske kojoj će se sve vrijednosti postaviti na nulu pomoću funkcije *zeroes_like* iz biblioteke *NumPy*. Pošto imamo opciju uključivanja i isključivanja filtera prvo moramo postaviti dva upita.

– Ako je vrijednost klizača za filter na jedinici – filter je uključen i poprima vrijednosti s ostalih klizača.

- Ako je vrijednost klizača za filter na nuli – filter je isključen i sve vrijednosti na klizačima su nula.

Kako je to postignuto je prikazano na slici 3.19.

[4] Int – cijelo brojni tip podatka

```
65      lipsColor = np.zeros_like(lipsMask) #vraca polje nula ali sa istim tipom i oblikom maske, samo su postavke maske vrijednosti nula
66
67      if s==0: #ako je switch iskljucen boju maske postavlja na nule, mijenjanjem R,G,B slidera nece promjeniti boju filtera
68          lipsColor[:]=0
69          a = 0
70      else: #ako je switch upaljen mijenjat ce se boja filtera
71          lipsColor[:] = b, g, r
```

Slika 3.19. Upiti za postavljanje boja maske

Vrijednost transparentnosti je jako osjetljiva što znači ako stavimo vrijednost 100 neće biti transparentno. Pošto je poželjno da ju možemo namjestiti što preciznije potrebno ju je podijeliti s nekim brojem, u ovom slučaju koristit će se broj 50. Vrijednost klizača za transparentnost se kreće od 0 do 100, ali ovako vrijednost koju će maska imati je 50 puta manja, što daje opciju za preciznije postavljanje transparentnosti.

Također je potrebno spojiti boje i masku pošto su one istog formata, a to se postiže funkcijom *bitwise_and* koja spaja boju i masku u jedan niz. U ovom slučaju boje s klizača će se primijeniti na masku.

Sljedeće što je potrebno je spojiti originalni frame i masku. To se postiže korištenjem funkcije *addWeighted* koja prima originalni frame te njegovu vrijednost transparentnosti te masku i njezinu vrijednost transparentnosti. Ova funkcija spaja tj. miješa originalni frame i masku te vraća novi frame koji će se prikazivati na prozoru aplikacije. Gotov frame se prikazuje pomoću funkcije *imshow* koja prima ime prozora na kojem će se prikazivati i ime frame-a koji će biti prikazan. Kod se nalazi na slici 3.20.

```
76          lipsColor = cv.addWeighted(originalFrame, 1, lipsColor, a, 0) # funkcija koja stvara takozvani "blend" alpha
77
78          cv.imshow(title_window, lipsColor) #prikazujemo obojanu masku na prozoru
```

Slika 3.20. Funkcije miješanja frame-ova i prikazivanja na prozoru

4. Problemi pri realizaciji aplikacije

4.1. Izrada maske za obrve, trepavice i kapke

Problem na koji nailazimo je izrada maski za dijelove lica kojih ima dva ili više, te kako na te dijelove lica primijeniti masku. Ako bi izrada maske bila na isti način kao za usne, npr. maska za obrve bi bila spojena i imali bi dijelove maske koje ne bi trebalo biti kao na slici 4.1.

Napravljeno kao dvije odvojene maske – točno rješenje problema

Napravljeno kao jedna maska – netočno rješenje problema

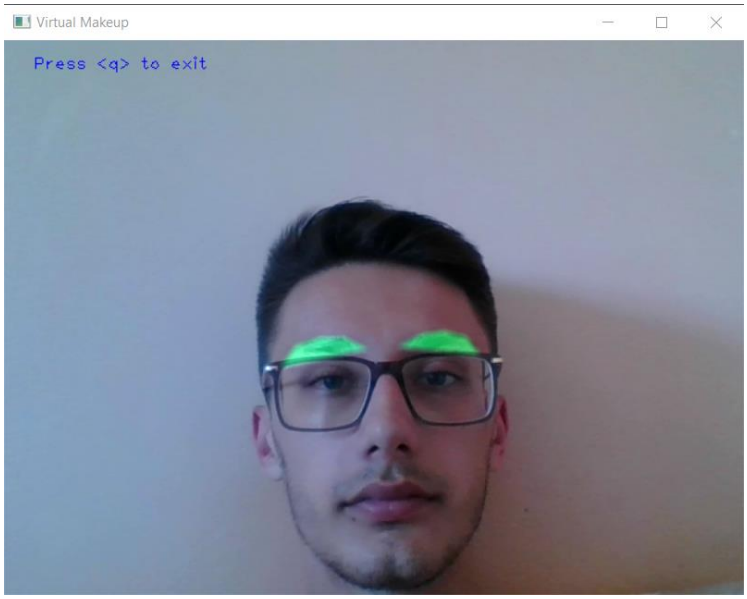
Slika 4.1. Maska za obrve

Ovakav problem se rješava tako da se naprave posebno maske za lijevu i desnu stranu lica tj. u ovom slučaju lijeve i desne obrve. Maska se pravi na isti način kao i za usne, ali je na kraju potrebno spojiti te dvije maske tako da kada se budu prikazivale na prozoru budu i dalje odvojene. Ovo omogućuje jednostavna funkcija iz biblioteke *OpenCV* i ona glasi *add* koja je prikazana na slici 4.2.

```
104        both_masks = cv.add(eyebrow_L_Color, eyebrow_R_Color)
```

Slika 4.2. Funkcija add za spajanje dviju obrva

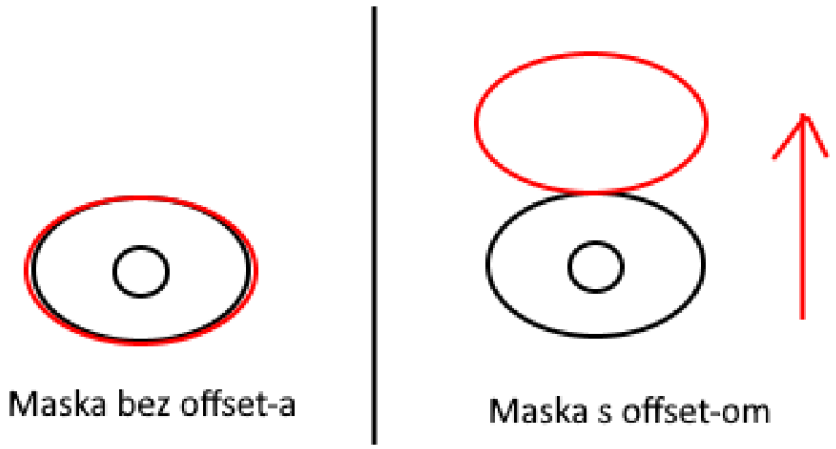
Nakon zvanja funkcije *add* možemo prikazati obje maske odjednom kao na slici 4.3.



Slika 4.3. Rezultat funkcije add za spajanje dviju obrva

4.2. Nepostojeći orijentiri

Drugi problem je taj što ne postoje orijentiri za trepavice i kapke pa je potrebno razmišljati nekonvencionalno. Problem za kapke se rješava tako da se uzmu orijentiri oka i primjeni *offset*[5]. *Offset* radi na način tako što koordinatama maske dodaje ili oduzima x i y vrijednosti. Funkcionalnost *offseta* prikazan je na slici 4.4.



.

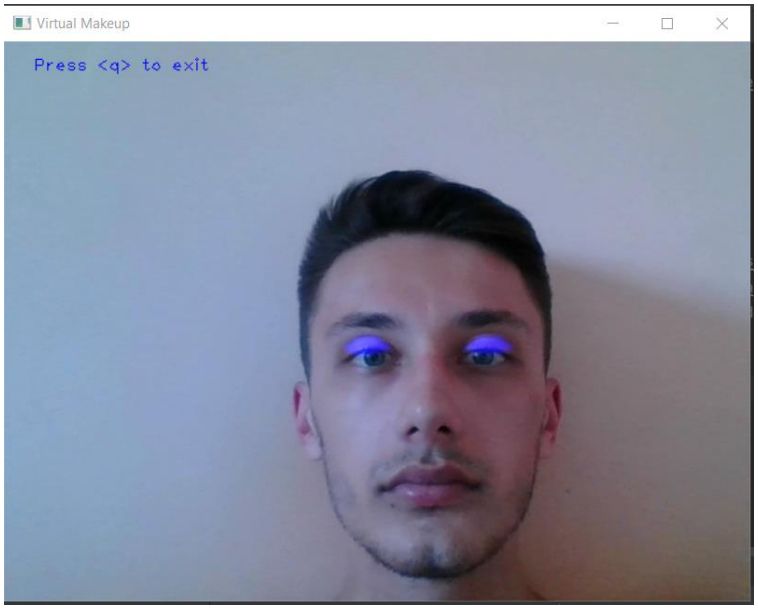
Slika 4.4. Maska s offset-om

[5] Eng. offset – kompenzacijski ekvivalent

Ovo je potrebno napraviti prije nego se maske naprave jer funkcija za pravljenje maske može poprimiti veličinu *offset*a koji se zadaje u formatu [x,y]. Na slici 4.5. je prikazan kod .

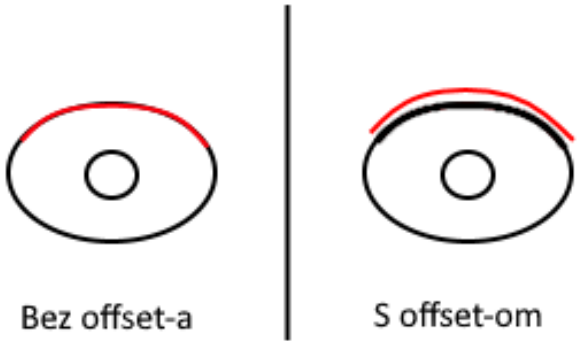
```
137         offset_left_lid = [5,-12] #offset za masku
138         offset_right_lid = [-5,-12]
139
140         eyelid_R_Mask = createMask(frame,npLandMarkPoints[36:42],offset_right_lid)
141         eyelid_L_Mask = createMask(frame,npLandMarkPoints[42:48],offset_left_lid)
```

Slika 4.5. Funkcija za pravljenje maske s offsetom



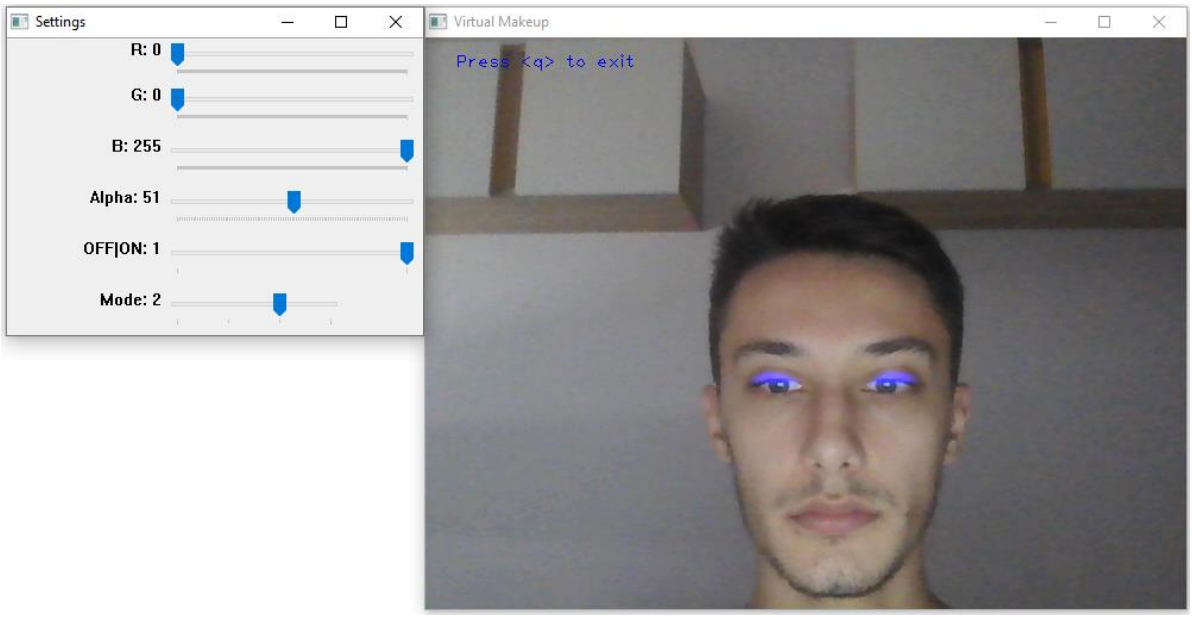
Slika 4.6. Maska za kapke

Na slici 4.6. prikazan je dobiveni rezultat maske za kapke.



Slika 4.7. Maska za trepavice s i bez offseta

Maska za trepavice se pravi na isti način s *offset*om, ali se uzimaju samo orijentiri vrha oka kao na slici 4.7.



Slika 4.8. Maska za trepavice

Rezultat maske za trepavice s postavljenim filterima je prikazan na slici 4.8.

5. Poboljšanje aplikacije

5.1. Izgled maske

Način na koji možemo poboljšati izgled maske je pomoću funkcije *Gaussian Blur* koja će zamutiti masku i dodati doživljaj da su vrhovi gdje se spajaju orijentiri glađi umjesto špicasti što je moguće vidjeti na slici 4.1. Ovaj primjer poboljšanja izgleda maske je korišten u ovom radu kako bi se prirodnije prikazala šminka.

Sa *Gaussian Blur* Bez *Gaussian Blur*

Slika 5.1. Funkcija *Gaussian Blur*

5.2. Bolji detektor lica

Način na koji se može poboljšati detektor lica je koristiti detektor s više orijentira ili koristeći strojno učenje kojim bi bilo moguće napraviti vlastiti detektor lica i vlastite orijentire. Ovime bi se postigla bolja preciznost za dijelova lica i poboljšanje izgleda maski za lice.

6. ZAKLJUČAK

Tehnologija *AR*-a sve se više razvija i sve se više koristi u svijetu. To su filteri za socijalne mreže, treniranje operacija u medicini, autoindustriji, vojnoj primjeni za pilote, tijekom televizijskih prijenosa i mnogo drugih primjena. Ovim završnim radom smo izradili aplikaciju za filter za šminku koristeći programski jezik Python i dodatne biblioteke koje on podržava. Izrađen je detektor lica koji se također može koristiti za otključavanje mobilnih uređaja. Uspješno je napravljena maska za lice kojoj je moguće primijeniti različite boje u širokom spektru. Za pokretanje aplikacije je potrebno samo računalo. Aplikaciju bi također bilo moguće napraviti je za mobitele, tablete i ostale uređaje koristeći se drugim programskim jezicima kao što su *Java* i *C++* ili koristeći se alternativnim programima kao što su *ARCore, Apple ARKit* te ostale navedene alternative u ovom radu. Jedan od načina poboljšanja aplikacije je moguće postići korištenjem naprednije biblioteke kao što je *Tkinter, PyQt5* i *Kivy* koje se stvorene specifično za pravljenje *GUI-a* aplikacija napravljenih u programskom jeziku *Python*. Također bi se mogao koristiti bolji detektor lica koji se služi većim brojem orijentira za precizniji pronalazak dijelova lica.

LITERATURA

[1] *PyCharm* internet stranica https://www.jetbrains.com/PyCharm/, [22.8.2022.]

[2] *Python* internet stranica https://www.Python.org/, [22.8.2022.]

[3] *OpenCV* internet stranica https://OpenCV.org/, [22.8.202.]

[4] *Numpy* internet stranica https://Numpy.org/, [22.8.2022.]

[5] *Dlib* internet stranicahttp://Dlib.net/, [22.8.2022.]

[6] *ARCore* internet stranica https://arstechnica.netblogpro.com/gadgets/2018/05/google-arcore-1-2-enables-shared-ar-experiences-across-android-and-ios, [22.8.2022.]

[7] *Apple ARKit* internet stranica https://www.apple.com/ne/newsroom/2018/06/apple-unveils-arkit-2, [22.8.2022.]

[8] *Unity* internet stranica https://unity.com/unity/features/ar, [22.8.2022.]

[9] Što je *offset* u proširenoj stvarnosti https://sparkar.facebook.com/ar-studio/learn/patch-editor/utility-patches/offset-patch/#input, [22.8.2022.]

[10] Što je *Gaussian blur* https://www.adobe.com/creativecloud/photography/discover/gaussian-blur.html, [22.8.2022.]

[11] C++ https://www.w3schools.com/cpp/cpp_intro.asp, [22.8.2022.]

[12] Java https://www.java.com/en/, [22.8.2022.]

SAŽETAK

U ovom radu je opisan postupak izrade programa za virtualnu šminku. Opisuje koje se integrirano razvojno okruženje koristilo te potrebne biblioteke za izradu rada. Korišten je samo programski jezik *Python*. Objašnjen je način izrade programa i tehnike kako je moguće postići što bolje rezultate. Postignuti rezultati su dobre kvalitete, ali moguće ih je poboljšati koristeći više različitih biblioteka za poboljšanje korisničkog sučelja i duže strojno učenje za prepoznavanje dijelova lica.

Ključne riječi: *OpenCV*, proširena stvarnost, *Python, PyCharm*, virtualna šminka

ABSTRACT

Virtual makeup

This paper describes the process of creating a program for virtual makeup. It describes which integrated development environment was used and the necessary libraries to create the work. Only the *Python* programming language was used. The method of creating the program and the technique to achieve the best possible results are explained. The results achieved are of good quality, but they can be improved by using several different libraries to improve the user interface and longer machine learning to recognize parts of the face.

Keywords: Augmented Reality, *OpenCV, Python*, *PyCharm*, Virtual Makeup