Programska aplikacija za pronalaženje karakterističnih točaka na licu

Pačić, Ira

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Ira Pačić

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Prof. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Ira Pačić

Zagreb, 2022.

Izjavljujem da sam ovaj rad izradila samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se profesoru Tomislavu Stipančiću za pruženu podršku te pomoć prilikom izrade ovog rada, te kolegama i prijateljima na korisnim savjetima.

Posebice se zahvaljujem kolegicama Korini i Heleni te mom Luki na neizmjernoj podršci tokom studiranja.

Na kraju, najveća zahvala mojoj obitelji na pruženoj podršci i povjerenju tokom cijelog studija.

Ira Pačić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Ira Pačić** JMBAG: **0035220496**

Naslov rada na hrvatskom jeziku: **Programska aplikacija za pronalaženje karakterističnih točaka na licu**

Naslov rada na engleskom jeziku: **Software application for finding characteristic points on the human face**

Opis zadatka:

Pronalaženje karakterističnih točaka na licu (eng. facial landmarks detection) je zadatak iz domene računalnog vida u kojem je cilj otkriti i pratiti ključne točke s ljudskog lica. Pronalaženje karakterističnih točaka na licu se koristi kao temelj za mnoge složene zadaće iz više područja, uključujući umjetnu inteligenciju, virtualnu stvarnost i računalni vid. Tako je omogućeno, na primjer, prenošenje točaka na licu stvarne osobe na lice virtualnog agenta unutar virtualne okoline, određivanje smjera pogleda osobe temeljem informacija o položaju lica, određivanje smještaja točaka na licu u svrhu donošenja zaključaka o trenutnom emocionalnom stanju promatrane osobe i sl.
U zadatku je potrebno napisati programsku podršku koristeći Python programski jezik i pripadajuće programske biblioteke strojnog učenja te računalnog vida, te ostvariti programsku aplikaciju za pronalaženje karakterističnih točaka na licu. Kao ulazni tok podataka programska aplikacija može koristiti video materijal koji se učita u aplikaciju ili podatke s kamere koja radi u realnom vremenu.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:
9. 5. 2022.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Opis
AI	Artificial intelligence (umjetna inteligencija)
RGB	Red, green, blue (sustav boja)
BGR	Blue, green red (sustav boja)
CNN	Konvolucijska neuronska mreža
RNN	Rekurentna neuronska mreža
OOP	Objektno orijentirano programiranje

SAŽETAK

Rad ukratko opisuje proces kreiranja programa za prepoznavanje točaka na licu. Prvotno se opisuju potrebni softveri za kreiranje programa te njihova primjena. Zatim slijedi detaljan opis postupka kreiranja koda te samo objašnjenje postupka. Naposlijetku slijedi opis objektno orijentiranog programiranja te njegova primjena u ovom radu. Također se opisuje dio programa te samog koda gdje je primijenjen ovakav način programiranja. Uz to je napravljena analiza prednosti i nedostataka svih biblioteka korištenih u ovom radu te kritički osvrt na rad konačne verzije programa.

Ključne riječi: prepoznavanje točaka, programiranje, računalni vid

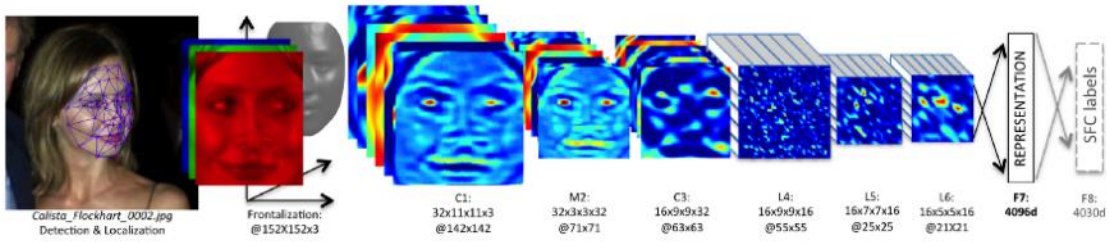
SUMMARY

The paper briefly describes the process of creating a program for facial landmark detection. Initially, the necessary software for program creation and their application are described. Then follows a detailed description of the code creation process and an explanation of the process itself. Finally, there is a description of object-oriented programming and its application in this paper. It also describes the part of the program and the code itself where this type of programming is applied. In addition, an analysis of the advantages and disadvantages of all the libraries used in this program was made, as well as a critical review of the work of the final version of the program.

Key words:facial landmark detection, programming, computer vision

1. UVOD

Kontinuiranim razvojem znanosti i tehnologija, pronalaze se novi načini olakšavanja različitih aspekata u životu, čime se može poboljšati, ali – po nekima – i pogoršati kvaliteta života te općenito stanje u društvu poput otvaranja/zatvaranja određenih radnih mjesta i sl. Prepoznavanje lica je posebno zanimljivo, jer suprotno pretpostavci da ljudsko znanje nije korisno kad je riječ o naprednim modelima - ljudi mogu uvelike pomoći modelu dubokog te strojnog učenja. Ljudi znaju puno o ljudskim licima pa iz tog razloga model može imati koristi od ljudske podrške. Jedan od prvih ovakvih programa, DeepFace je unatoč tome zamišljen rano u prvom usponu dubokog učenja. Prije su se koristili tradicionalniji algoritmi strojnog učenja, no DeepFace se približio ljudskim performansama prepoznavanja lica i postigao impresivna poboljšanja u točnosti računala. Godine 2015. , ubrzo nakon izlaska DeepFacea, Google je objavio svoj razvoj prepoznavanja lica, FaceNet. Bio je drugačiji od DeepFacea po tome što je korištena jedna neuronska mreža za cijeli proces umjesto njih nekoliko.[1] Današnji oblici sustava za prepoznavanje točaka na licu izgledaju nešto drugačije, a jedan od takvih primjera pokazan je u ovom radu. Korišteni su noviji sustavi poput OpenCv programa koji je unaprijed osmišljen za ovakve vrste zadataka te uvelike olakšava sam proces programiranja.



Slika 1. DeepFace način rada

2. TEORIJSKA PODLOGA

2.1. Umjetna inteligencija

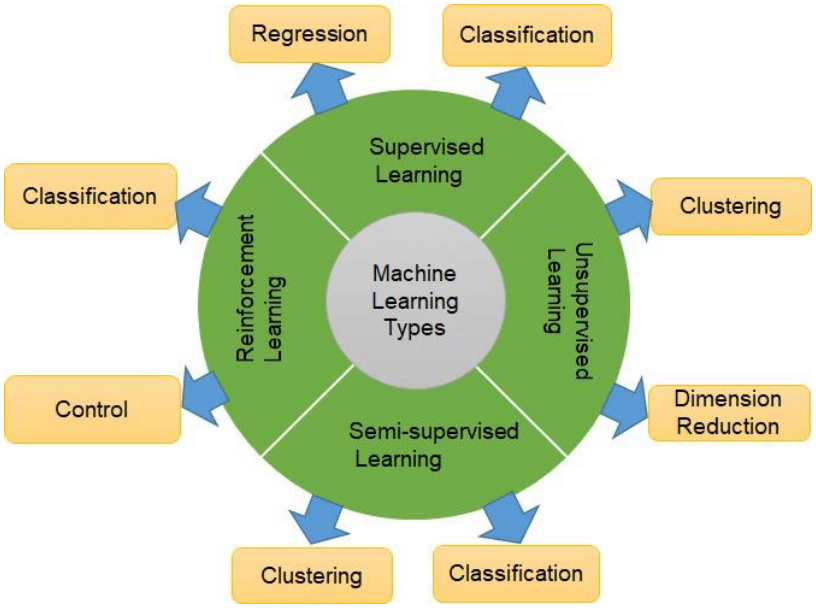
Umjetna inteligencija je simulacija procesa ljudske inteligencije od strane strojeva, posebno računalnih sustava. Neke od primjena umjetne inteligencije uključuju ekspertne sustave, obradu prirodnog jezika, prepoznavanje govora i strojni vid. Za razvoj AI potrebni su temelji specijaliziranog hardvera i softvera za pisanje i obuku algoritama strojnog učenja. Općenito, sustavi umjetne inteligencije funkcioniraju tako da unose velike količine označenih podataka o obuci, analiziraju podatke radi kreiranja uzoraka i koriste te obrasce za predviđanje budućih stanja. Na ovaj način alat za prepoznavanje slika može naučiti identificirati i opisati objekte na slikama pregledom milijuna primjera.[11] Neke od prednosti umjetne inteligencije su te da je ona dobra za poslove usmjerene na detalje, potrebno je puno manje vremena za napraviti neke zadatke koji uključuju velik broj podataka te daje dosljedne rezultate. Sa druge strane ovakav način rješavanja zadataka zahtjeva duboku tehničku stručnost, AI zna sano ono što je pokazano te ne može generalizirati jedan zadatak na drugi.

2.2. Strojno učenje

Grana umjetne inteligencije i računalne znanosti koja se fokusira na upotrebu podataka i algoritama za oponašanje načina na koji ljudi uče te postupno poboljšava točnost tog algoritma naziva se strojno učenje. Uz strojno učenje, bitno je spomenuti i duboko učenje koje je također bitna grana umjetne inteligencije. Budući da se duboko učenje i strojno učenje koriste naizmjenično, treba uočiti razlike između njih. Strojno učenje, dubinsko učenje i neuronske mreže sve su potpodručja umjetne inteligencije. Međutim, neuronske mreže zapravo su grana strojnog učenja, a dubinsko učenje je potpodručje neuronskih mreža. Način na koji se duboko učenje i strojno učenje razlikuju je način na koji svaki algoritam uči. Duboko strojno učenje može koristiti, ali ne zahtijeva nužno, označene skupove podataka za informiranje svog algoritma. Duboko učenje može unijeti nestrukturirane podatke u svom osnovnom obliku (npr. tekst ili slike) i može odrediti skup značajki koje razlikuju različite kategorije podataka jedne od drugih. Time se smanjuje potreba za ljudskom pomoći i omogućuje korištenje većih skupova podataka za razliku od strojnog učenja koje u većoj mjeri ovisi o ljudskoj pomoći u učenju. Stručnjaci određuju skup značajki kako bi razumjeli razlike između unosa podataka, obično zahtijevajući više strukturiranih podataka za učenje. Duboko učenje i neuronske mreže zaslužni su za ubrzani napredak u područjima kao što su računalni vid, obrada prirodnog jezika i

prepoznavanje govora. [2] Sustav učenja algoritma strojnog učenja rastavlja se na tri glavna dijela prikazana i objašnjena u tablici 1., dok su na slici 2. prikazane različite metode strojnog učenja kao što su nadzirano strojno učenje, učenje bez nadzora, te polu-nadzirano učenje.

Nadzirano učenje definirano je upotrebom označenih skupova podataka za obuku algoritama za klasificiranje podataka ili točno predviđanje ishoda. Neke metode koje se koriste u nadziranom učenju uključuju neuronske mreže, Bayes, linearnu regresiju, logističku regresiju itd. Nenadzirano učenje koristi algoritme strojnog učenja za analizu i grupiranje neoznačenih skupova podataka. Ovi algoritmi otkrivaju skrivene uzorke ili grupiranja podataka bez potrebe za ljudskom intervencijom. Neki od algoritama koji se koriste u nenadziranom učenju uključuju neuronske mreže, model k-srednjih vrijednosti i dr. Polu nadzirano učenje nudi sredinu između nadziranog i nenadziranog učenja. Tijekom obuke koristi manji označeni skup podataka za usmjeravanje klasifikacije i izdvajanje značajki iz većeg, neoznačenog skupa podataka. Polu-nadzirano učenje može riješiti problem nedostatka označenih podataka za algoritam nadziranog učenja. Također pomaže ako je preskupo označiti dovoljno podataka.[2]



Slika 2. Metode strojnog učenja

Proces odlučivanja	Algoritmi strojnog učenja koriste se za predviđanje ili klasifikaciju. Na temelju nekih ulaznih podataka (označenih ili neoznačenih) algoritam će provesti procjenu uzorka u podacima.
Funkcija pogreške	Funkcija pogreške procjenjuje predviđanje modela. Ako postoje poznati primjeri, funkcija pogreške može napraviti usporedbu za procjenu točnosti modela.
Proces optimizacije modela	Ako se model može bolje uklopiti u točke unutar podataka koji se nalaze u skupu za obuku, tada se težine prilagođavaju kako bi se smanjila razlika između poznatog primjera i procjene modela. Algoritam će ponoviti ovaj proces te ažurirati težine sve dok se ne postigne određeni prag točnosti.

Tablica 1. Tri dijela sustava strojnog učenja[2]

2.3. Računalni vid

Računalni vid je polje umjetne inteligencije koje omogućuje računalima i sustavima da izvuku bitne informacije iz digitalnih slika i videa te poduzmu radnje ili daju preporuke na temelju tih informacija. Ako AI omogućuje računalima da razmišljaju, računalni vid im omogućuje da vide, promatraju i razumiju. Unatoč sličnom načinu funkcioniranja, ljudski vid za razliku od računalnog ima prednost u treniranju razlikovanja objekata,udaljenosti između njih, kreću li se i postoji li nešto pogrešno na slici unutar puno većeg vremenskog perioda. Računalni vid trenira

strojeve za obavljanje ovih funkcija, ali to mora učiniti u puno kraćem vremenu s kamerama, podacima i algoritmima. Računalni vid treba mnogo podataka te konstantno iznova provodi analize podataka dok ne uoči razlike i konačno prepozna slike. Da bi se to postiglo koriste se dvije osnovne tehnologije: ranije spomenuto duboko učenje i konvolucijska neuronska mreža (CNN). Strojno učenje koristi algoritme koji omogućuju računalu da uči o kontekstu vizualnih podataka. Ako se kroz model unese dovoljno podataka, računalo će "pogledati" podatke i naučiti razlikovati jednu sliku od druge. Algoritmi omogućuju stroju samostalno učenje, umjesto da ga netko programira da prepozna sliku. Neuronska mreža pokreće konvolucije (matematička operacija na dvjema funkcijama za proizvodnju treće funkcije) i provjerava točnost svojih predviđanja u nizu ponavljanja sve dok se predviđanja ne počnu ostvarivati. To je onda prepoznavanje ili gledanje slika na način sličan ljudima.

Slično kao što čovjek stvara sliku na daljinu, CNN prvo prepoznaje oštre rubove i jednostavne oblike, a zatim popunjava informacije te u isto vrijeme provodi ponavljanja svojih predviđanja. CNN se koristi za razumijevanje pojedinačnih slika. Rekurentna neuronska mreža (RNN) koristi se na sličan način za videe kako bi pomogla računalima razumjeti kako su slike u nizu međusobno povezane.[3]

2.4. Facial Landmark Detection - Definicija

Prepoznavanje točaka na licu zadatak je računalnog vida u kojem model treba predvidjeti ključne točke koje predstavljaju određena područja na ljudskom licu – oči, nos, usne i druge. Prepoznavanje točaka na licu osnovni je zadatak koji se može koristiti za izvođenje drugih zadataka računalnog vida, uključujući procjenu položaja glave, prepoznavanje smjera pogleda, otkrivanje gestikulacija lica i zamjenu lica.[4]

2.5. Način rada

Za prepoznavanje točaka na licu koriste se dvije ključne biblioteke pod nazivom OpenCv i Mediapipe. Proces se sastoji od dva glavna koraka. Prvo, model otkriva jedno ili više lica na slici. Drugo, slika detektira oko 468 ključnih točaka lica pomoću regresije. Ovaj model detektira 3D koordinate. Te x i y koordinate normalizirane su iz skale slike. Z koordinata se dohvaća uzimanjem relativnog izračuna između zaslona i x koordinata modela.

2.6. Programski jezik Python

Python je računalni programski jezik koji se često koristi za izradu web stranica i softvera, automatizaciju zadataka i provođenje analize podataka. Python je općenit programski jezik, što znači da se može koristiti za stvaranje niza različitih programa i nije specijaliziran za neke specifične probleme.[13]Ovaj programski jezik je orijentiran na objektno programiranje koje je i korišteno u programskom kodu u nastavku ovog rada.

2.7. OpenCv biblioteka

OpenCV (Open Source Computer Vision Library) biblioteka je softvera za računalni vid i strojno učenje otvorenog koda. OpenCV je napravljen kako bi osigurao zajedničku podlogu za aplikacije računalnog vida i ubrzao korištenje strojne percepcije u komercijalnim proizvodima. Postoji mnogo problema koji se rješavaju pomoću OpenCV-a, a neke primjena su navedene u nastavku:[5]

1. Prepoznavanje lica
2. Automatizirani pregled i nadzor
3. Broj ljudi – brojanje (pješački promet u trgovačkom centru, itd.)
4. Brojanje vozila na autocestama zajedno s njihovim brzinama
5. Interaktivne umjetničke instalacije
6. Otkrivanje anomalije (defekata) u proizvodnom procesu (čudni neispravni proizvodi)
7. Pretraživanje i dohvaćanje videa/slike
8. Navigacija i kontrola automobila bez robota i vozača
9. Prepoznavanje predmeta
10. Analiza medicinske slike
11. Filmovi – 3D struktura iz pokreta
12. Prepoznavanje reklama TV kanala

Ovdje se koristi programski jezik Python pa je potrebno instalirati biblioteku pod nazivom „opencv-python“. U ovom slučaju, ova biblioteka služi za prepoznavanje lica koja se nalaze na kameri, broj osoba koje se pojavljuju, te također ima unaprijed ugrađene naredbe za otvaranje web kamere, promjenu boje slike i mnoge druge, no ovdje je većinom bila korištena samo za već navedene opcije.

2.8. Mediapipe biblioteka

MediaPipe je biblioteka koja koristi princip strojnog učenja za obradu vremenskih serija podataka kao što su video, audio i dr. Namijenjena je za više platformi te radi na stolnom računalu, Androidu, iOS-u i ugrađenim uređajima kao što su Raspberry Pi i Jetson Nano. MediaPipe tok percepcije naziva se Graph. Graf se sastoji od čvorova povezanih rubovima. Unutar MediaPipe grafa, čvorovi se nazivaju kalkulatori, a rubovi se zovu tokovi. Rješenja ove biblioteke su unaprijed izgrađeni primjeri otvorenog koda koji se temelje na određenom unaprijed obučenom modelu. Trenutno se nudi šesnaest rješenja kako je navedeno u nastavku:[6]

1. Face Detection
2. Face Mesh
3. Iris
4. Hands
5. Pose
6. Holistic
7. Selfie Segmentation
8. Hair Segmentation
9. Object Detection
10. Box Tracking
11. Instant Motion Tracking
12. Objectron
13. KNIFT
14. AutoFlip

15. MediaSequence
16. YouTube 8M

MediaPipe Face Mesh je rješenje koje procjenjuje 468 3D točaka lica u stvarnom vremenu čak i na mobilnim uređajima. Koristi strojno učenje za određivanje 3D površine lica, zahtijevajući samo jedan ulaz kamere bez potrebe za senzorom dubine. Ovo rješenje pruža izvedbe u stvarnom vremenu ključne za iskustva uživo.[6] U ovom konkretnom slučaju se koristilo točno to rješenje uz pomoću kojeg se detektira 468 točaka na licu/licima te se prikazuju te točke i poveznice između njih. Također su se funkcije unutar ove biblioteke koristile za ispisivanje vrijednosti svih točaka.

3. PROGRAMSKI KOD

3.1. Tijek koda - sažeto

Prvi korak pisanja koda sastoji se od instaliranja OpenCv i Mediapipe biblioteke. Nakon instalacije slijedi uvođenje web kamere računala u program kako bi se mogla otvoriti pomoću programa i koristiti za daljnji tijek programskog koda. Nakon uvođenja kamere slijedi stvaranje dijela koda za kreiranje „face mesh-a“ odnosno mreže na licu koja sadrži 468 točaka te linije koje ih povezuju. Zatim se kreira dio programa za ispisivanje koordinata tih točaka ili čak vrijednosti svake točke.

3.2. Uvođenje kamere

Prvo se trebaju instalirati gore navedene biblioteke te se otvara novi Python projekt u kojeg se onda uvode te biblioteke na sljedeći način:

```
import cv2
import mediapipe as mp
```

Slika 3. Import biblioteka

Za uvođenje web kamere računala, ključni dio koda je prikazan na slici 4. Koristi se „cv2.VideoCapture(0)“ kako bi se mogla pokrenuti kamera uživo.U zagradi se piše nula kojom se određuje da je riječ o web kameri računala, a ako bi trebalo učitati video može se staviti putanja do tog videa.

```
cap = cv2.VideoCapture(0)
```

Slika 4. Uvođenje kamere

Također je bitan idući dio koji služi za čitanje onog što se prikazuje na kameri te prikaz istoga:

```
while cap.isOpened():
  success, image = cap.read()

cv2.imshow("kamera", image)
```

Slika 5. Prikaz slike

Dakle, dok je kamera otvorena, čita se dobivena slika te prikazuje na kameri. Konačno, slijedi dio za isključivanje kamere pritiskom tipke „q“:

```
if cv2.waitKey(1) & 0xFF == ord("q"):
    break
```

Slika 6. Isključivanje kamere

3.3. Uvođenje potrebnih alata za crtanje mreže

Sljedeća slika prikazuje uvođenje potrebnih atributa za crtanje. Dakle, iz unaprijed napravljenih rješenja za kreiranje crteža izvlači se ono što je nama potrebno. Jedno od tih rješenja je prikazano na slici 7. u prvom redu. Crteži se mogu nacrtati i ručno, no ovo rješenje se koristi zbog pojednostavljenja crtanja jer se ručnim crtanjem puno teže naprave linije između točaka koje su potrebne za željeni prikaz mreže. Iz istog razloga postoji i iduća po redu funkcija koja služi za stil crtanja. Nakon nje slijedi rješenje za kreiranje face mesh-a te naposlijetku specifikacije linija i čvorova koji će biti nacrtani. Ovdje je odabrana roza boja,te su debljina linije i radijus definirani brojem 1.

```
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_face_mesh = mp.solutions.face_mesh
drawing_spec = mp_drawing.DrawingSpec(color=(245,66,230),thickness=1, circle_radius=1)
```

Slika 7. Funkcije za crtanje

3.4. Glavni dio koda – kreiranje mreže na licu

Za glavni dio koristi se funkcija „mp_face_mesh.FaceMesh()“ pomoću koje se pronalazi mreža na licu. Unutar nje se određuje maksimalni broj lica koje će program prepoznati,te točnost detekcije i praćenja točaka na licu. Ako je šansa pronalaženja lica veća od 50 posto kako je ovdje namješteno, program pronalazi to lice te zatim započinje praćenje tog lica za koje također vjerojatnost praćenja treba biti iznad 50 posto. Zatim slijedi uvođenje kamere te prepoznavanje i kreiranje slike koje je ranije objašnjeno. Također se slika treba pretvoriti iz BGR u RGB format jer klasa „facemesh“ prihvaća samo RGB slike. Do sada objašnjen dio koda prikazan je na slici 8.

```
with mp_face_mesh.FaceMesh(
    max_num_faces=3,
    refine_landmarks=True,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as face_mesh:
  while cap.isOpened():
    success, image = cap.read()
    image.flags.writeable = False
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = face_mesh.process(image)
```

Slika 8. Glavni dio koda – prvi dio

Nakon toga se pomoću „if“ funkcije i „for“ petlje crtaju točke na licu. Dakle, ako je nešto detektirano se kreće crtati, no obzirom da program radi i za više lica, potrebna je i for petlja kako bi se prošlo kroz sva lica i tek onda crtala mreža. Određuje se nekoliko ulaznih varijabli kao što je prikazano na slici 9. Kao slika se stavlja ranije učitana slika koja se prikazuje na kameri, listu točaka predstavljaju unaprijed određene točke na licu, crtanje tih točaka i linija između njih je također ranije određeno pomoću „drawing_spec“ funkcije. Za prikaz točaka odabire se opcija „facemesh tesselation“ koja stvara poligone od točaka na licu. Iste korake potrebno je ponoviti sa opcijom „facemesh contours“ kako bi se prikazale spojnice između točaka.

```
with mp_face_mesh.FaceMesh(
    max_num_faces=3,
    refine_landmarks=True,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as face_mesh:
  while cap.isOpened():
    success, image = cap.read()
    image.flags.writeable = False
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = face_mesh.process(image)

    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    if results.multi_face_landmarks:
        for face_landmarks in results.multi_face_landmarks:
            mp_drawing.draw_landmarks(
                image=image,
                landmark_list=face_landmarks,
                connections=mp_face_mesh.FACEMESH_TESSELATION,
                landmark_drawing_spec=drawing_spec,
                connection_drawing_spec=drawing_spec
                )
```

```
mp_drawing.draw_landmarks(
    image=image,
    landmark_list=face_landmarks,
    connections=mp_face_mesh.FACEMESH_CONTOURS,
    landmark_drawing_spec=drawing_spec,
    connection_drawing_spec=drawing_spec
    )
mp_drawing.draw_landmarks(
    image=image,
    landmark_list=face_landmarks,
    connections=mp_face_mesh.FACEMESH_IRISES,
    landmark_drawing_spec=drawing_spec,
    connection_drawing_spec=drawing_spec
    )
```

Slika 9. Glavni dio koda - potpuni

Ovime je napravljena osnovna verzija koda koju ćemo pretvoriti u modul pomoću objektno orijentiranog programiranja objašnenog u nastavku.

4. OBJEKTNO ORIJENTIRANO PROGRAMIRANJE

Objektno orijentirano programiranje (OOP) je oblik programiranja koji se oslanja na koncept klasa i objekata. Koristi se za podjelu softverskog programa u jednostavne dijelove nacrta koda koji se mogu ponovno koristiti (obično se nazivaju klasama), koji se koriste za stvaranje pojedinačnih instanci objekata. OOP jezik omogućuje rastavljanje programa na probleme veličine bita koji se mogu lako riješiti (jedan po jedan objekt). Prednosti ovakvog programiranja su veća produktivnost programera, bolja kvaliteta softvera i manji troškovi održavanja. OOP sustavi mogu se lako nadograditi iz malih u velike sustave. [14]Ovakav koncept se koristio u ovom radu zbog pojednostavljenja pozivanja programa. Napravi se modul koji se kasnije može samo pozvati unutar drugih programa kojima je potreban te se čitav kod pokrene samo preko te jedne linije koda.

4.1. Tijek koda OOP – sažeto

Početni dio koda je u potpunosti isti te se započinje uvođenjem potrebnih biblioteka. Nakon toga slijedi definiranje glavne funkcije odnosno kreiranje modula te zatim slijedi uvođenje web kamere računala u program kako bi se mogla otvoriti pomoću programa i koristiti za daljnji tijek programskog koda. Nakon toga se kreira glavni dio koda koji se malo razlikuje od osnovnog koda opisanog na početku. Te razlike biti će objašnjene u nastavku.

4.2. Pokretanje modula

Prvo što se treba napraviti je definirati što će se prikazati ako se pokreće sami modul, a to se radi na sljedeći način:

```
if __name__ == "__main__":
    main()
```

Slika 10. Pokretanje modula

Dakle, ako je ime jednako main, pokreće se glavna funkcija koja se mora definirati. Definira se pomoću funkcije „def“ te se unutar toga stavlja dio koda za učitavanje kamere, čitanje slike te prikaz iste. Taj dio jednak je kao i u osnovnom kodu, samo što se ovdje treba smjestiti unutar te glavne funkcije, što je prikazano na slici 11.

```
def main():

    cap = cv2.VideoCapture(0)
    detector = FaceMeshDetector()
    while cap.isOpened():
            success, image = cap.read()
            image = detector.findFaceMesh(image)



            image.flags.writeable = False
            cv2.imshow("kamera", image)
            if cv2.waitKey(1) & 0xFF == ord("q"):
                break
```

Slika 11. Definiranje glavne funkcije

4.3. Kreiranje klase

Na početku se određuje ime klase te se zatim definira metoda za inicijalizaciju. Ovdje se definiraju određeni parametri koji već postoje unutar FaceMesh-a, ali ih ovdje treba također definirati. Prvi je namješten na False što znači da služi prvo za detekciju pa onda za praćenje, te slijede isti parametri kao u osnovnom kodu, a to su: maksimalni broj lica koja se detektiraju te uspješnost detekcije i praćenja. Ti svi parametri se pridodaju objektu te se definiraju njihove vrijednosti koje su jednake tim parametrima.

```
import cv2
import mediapipe as mp
import time

class FaceMeshDetector():

    def __init__(self, StaticMode = False, max_num_faces = 2
                 , refineLandmarks = False, minDetectionCon = 0.5, minTrackCon = 0.5):
        self.StaticMode = StaticMode
        self.max_num_faces = max_num_faces
        self.refineLandmarks = refineLandmarks
        self.minDetectionCon = minDetectionCon
        self.minTrackCon = minTrackCon
```

Slika 12. Inicijalizacija

4.4. Uvođenje potrebnih alata za crtanje mreže

Za ovaj korak koriste se svi jednaki alati kao u osnovnom kodu, samo ih se uvodi pomoću funkcije „self“ te se unutar tih alata kopiraju ranije definirani parametri.

```
self.mp_drawing = mp.solutions.drawing_utils
self.mp_drawing_styles = mp.solutions.drawing_styles
self.mp_face_mesh = mp.solutions.face_mesh
self.drawing_spec = self.mp_drawing.DrawingSpec(color=(245, 66, 230), thickness=1, circle_radius=1)
self.face_mesh = self.mp_face_mesh.FaceMesh()
self.mp_face_mesh.FaceMesh(self.StaticMode,
  self.max_num_faces,
  self.refineLandmarks,
  self.minDetectionCon,
  self.minTrackCon)
```

Slika 13. Alati za crtanje mreže

4.5. Glavni dio koda – kreiranje mreže na licu

Za glavni dio se definira metoda za pronalaženje mreže na licu te se unutar nje stavlja glavni dio koda koji je jednak osnovnom kodu osim što se kao i ranije, dodaje funkcija „self“ na definirane funkcije unutar koda. Također je potrebno definirati detektor mreže na licu unutar glavne funkcije kako bi sve radilo. Konačno, taj dio izgleda ovako:

```
def findFaceMesh(self, image, draw = True):
    image.flags.writeable = True
    self.image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    self.results = self.face_mesh.process(self.image)
    faces = []
    if self.results.multi_face_landmarks:
        for face_landmarks in self.results.multi_face_landmarks:
            if draw:
                self.mp_drawing.draw_landmarks(
                  image=image,
                  landmark_list=face_landmarks,
                  connections=self.mp_face_mesh.FACEMESH_TESSELATION,
                  landmark_drawing_spec=self.drawing_spec,
                  connection_drawing_spec=self.drawing_spec
                 )

                self.mp_drawing.draw_landmarks(
                  image=image,
                  landmark_list=face_landmarks,
                  connections=self.mp_face_mesh.FACEMESH_CONTOURS,
                  landmark_drawing_spec=self.drawing_spec,
                  connection_drawing_spec=self.drawing_spec
                 )
```

Slika 14. Glavni dio koda

4.6. Ispisivanje vrijednosti točaka

Ovo je dodatan korak kojim se postiže ispisivanje vrijednosti točaka u prostoru. Tijek koda je sljedeći: za svako lice će se proći kroz svaku točku te će se svaka točka pretvoriti u x i y koordinatu te zatim u piksele te će se one zatim pohraniti. Iz tog razloga se kreira nova varijabla u obliku liste u koju će se pohraniti te koordinate i zatim pomoću opcije „print“ prikazati na ekranu. Obzirom da se može prikazati više lica, kreira se još jedna lista izvan petlje kako bi se u nju pohranile sve točke za sva lica.

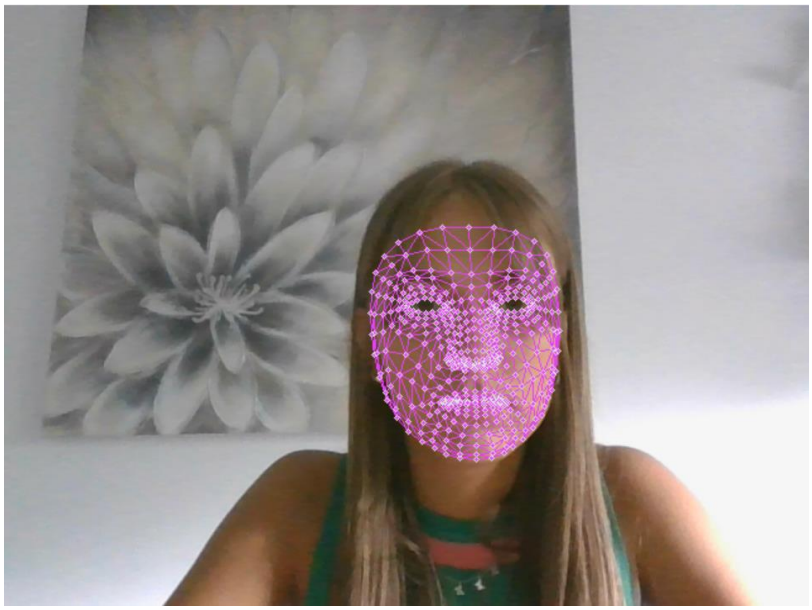
```
            face = []
            for id, landmark in enumerate(face_landmarks.landmark):
                    ih, iw, ic = image.shape
                    x, y = int(landmark.x*iw), int(landmark.y*ih)

                    print(id, x, y)
                    face.append([x, y])
            faces.append(face)
    return image
```

Slika 15. Ispisivanje vrijednosti točaka

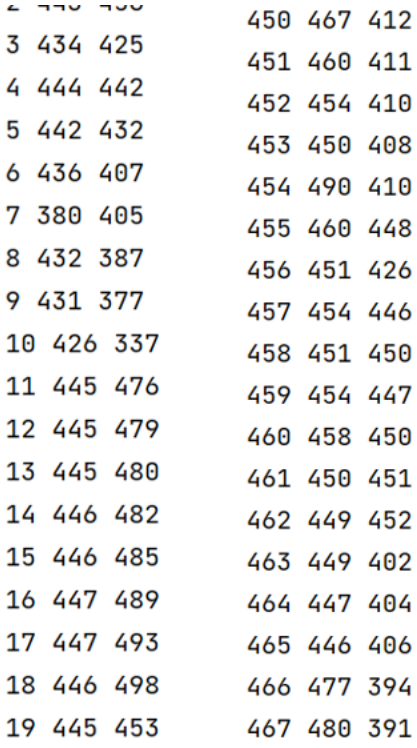
5. REZULTAT I ANALIZA

Tijek događaja nakon pisanja koda je sljedeći: pokrene se odabrani kod, početno se odabere osnovni kod te se pokrene. Nakon pokretanja se uključuje kamera na kojoj se vidi osoba te specifične točke na licu te osobe. Okretanjem glave su i dalje prisutne te točke te zapravo ta mreža prati kretnje lica te osobe. Zatim se program može ugasiti pritiskom tipke q. Drugi programski kod ima isti tijek događaja, no dodatno se na dnu ekrana prikazuju brojevi kojima su definirane te točke (od 0 do 467) te pored njih vrijednosti piksela tih točaka.

Konačni rezultat bi trebao biti prikaz mreže na licu te ispisivanje vrijednosti piksela svih točaka sadržanih u toj mreži. To je prikazano na slikama 15. i 16.



Slika 16. Konačna mreža



Slika 17. Dio ispisa točaka

Konačnom analizom svih korištenih biblioteka te programa, i samim promatranjem rada dobivenog koda, uočene su određene prednosti i nedostaci prikazani u tablici 2.

	PREDNOSTI	NEDOSTACI
Python	• Sveukupno odličan programski jezik, lako se shvati korištenje dosta funkcija unutar jezika te ima puno dostupnih objašnjenja • Velika zajednica • Širok spektar dostupnih biblioteka • besplatan	• Prilično osjetljiv na različite razmake u kodu • Velika potrošnja memorije

OpenCv	• Besplatna biblioteka • Budući da je biblioteka OpenCV napisana u C/C++, prilično je brza • Mala upotreba memorije • Prenosiv je jer može raditi na skoro bilo kojm uređaju • Puno dostupne dokumentacije na web stranici	• Nisu uočeni veći nedostaci
Mediapipe	• Besplatna biblioteka • Unaprijed izrađena rješenja za puno različitih primjena	• Nisu uočeni veći nedostaci

Tablica 2. Analiza prednosti i nedostaci

5.1. Usporedba sa postojećim rješenjima

Jedno od mogućih rješenja je korištenjem dlib biblioteke umjesto Mediapipe biblioteke. Sa takvim rješenjem se može detektirati 68 točaka na licu, dakle znatno manje te se može namjestiti da se detektira samo određenih 5 točaka koje se odaberu. Prvo se te točke moraju definirati uz pomoću naziva koji im pripada te pripadajućih koordinata. Zatim se definiraju funkcije pomoću kojih će se crtati te točke, slično kao i u Mediapipe-u, samo se funkcije drugačije zovu te je dosta duži proces definiranja istih. Zatim se inicijalizira osnovni model te se pozove funkcija unutar dlib-a koja se zove „predictor“ koja služi za detekciju točaka. Nakon toga se pokrene kamera i program kreće detektirati točke. Ovakav način je malo kraći, no detektira se i manje točaka te se ne kreira mreža između njih kako bi se bolje prikazala struktura lica već su samo

prikazane točke. Dio takvog programskog kod prikazan je na slici 18. kako bi se bolje predočile razlike između navedenih biblioteka.

```
# Detect faces:
rects = detector(gray, 0)

# For each detected face, find the landmark.
for (i, rect) in enumerate(rects):
    # Draw a box around the face:
    cv2.rectangle(frame, (rect.left(), rect.top()), (rect.right(), rect.bottom()), (0, 255, 0)

    # Get the shape using the predictor:
    shape = predictor(gray, rect)

    # Convert the shape to numpy array:
    shape = shape_to_np(shape)

    # Draw all lines connecting the different face parts:
    #draw_shape_lines_all(shape, frame)

    # Draw jaw line:
    #draw_shape_lines_range(shape, frame, JAWLINE_POINTS)

    # Draw all points and their position:
    # draw_shape_points_pos(shape, frame)
    # You can also use:
    # draw_shape_points_pos_range(shape, frame, ALL_POINTS)
```

Slika 18. Kod pomoću dlib biblioteke [10]

6. PRIMJENA

Obzirom da se pomoću prepoznavanja točaka na licu mogu određivati i emocije te određena stanja, ovakav sustav imao bi koristi pri prepoznavanju umora kod vozača te tako služio za sprječavanje prometnih nesreća uzrokovanih umorom. Na primjer, model računalnog vida mogao bi obraditi video izvore s kamere u vozilu koja otkriva znakove umora na licu vozača. Model bi mogao pokrenuti upozorenje ako vozač ne obrati dovoljno pozornosti na cestu. Također, jedna od zanimljivih primjena je ona za kreiranje raznih likova u crtićima, filmovima i igricama. U ovom smislu se može koristiti i za promjene kretnji lica kako bi odgovarale zvuku koji se govori u određenom filmu, crtiću itd. Još jedna moguća primjena je za potrebe prepoznavanja lica. Ovakav slučaj upotrebe uključuje algoritme koji izvode zadatke provjere lica, prepoznavanja i grupiranja sličnih lica. Najbolji algoritmi koriste pretprocesiranje lica zajedno s poravnavanjem lica za poboljšanje prepoznavanja lica. Emocionalni izrazi vidljivi su pokretima usana, očiju i obrva. Prepoznavanje orijentira na licu može pomoći u prepoznavanju emocija što je povezano sa prvom navedenom primjenom.

7. KRITIČKI OSVRT

Prilikom rada na programu opisanog u ovom radu te kreiranja konačne verzije programa, trebalo bi se osvrnuti i na sam rad programa. Jedna od pozitivnih strana je ta što osnovni model programa funkcionira za prikaz „Face Mesh-a“ na više osoba (koliko god se namjesti unutar koda) što ga čini naprednijim i kvalitetnijim. Također, treba istaknuti zadovoljstvo samim prikazom mreže i točaka na licu koje su vrlo detaljno i čisto prikazane te jako dobro prate kretnje lica te bilo kakve manje pomake. Također, postignuta je i mogućnost prikazivanja zjenica oka pomoću posebnih točaka. Što se tiče modula u kojem je korišteno objektno orijentirano programiranje, uočen je nedostatak koji se tiče prepoznavanja više lica odjednom. Uočeno je da ne može prepoznati više lica odjednom zbog dodatnog dijela napravljenog u tom kodu,a taj dio je ispis vrijednosti piksela. Zbog toga može prepoznati te ispisivati vrijednosti samo za jedno lice. No, ispisivanje vrijednosti točaka svakako se može uzeti kao dodatan plus radu programa. Ovakav nedostatak može se popraviti ukoliko bismo maknuli/promjenili dio koda za ispisivanje vrijednosti točaka ili ga dodatno unaprijedili kompliciranijim kodom. Ostale funkcije unutar programa i kod daju rezultate i bolje od očekivanih.

8. ZAKLJUČAK

U ovome radu napravljen je program za detekciju i praćenje točaka na licu te ispisivanje vrijednosti piksela istih. Kao teorijska podloga dan je opis strojnog učenja, računalnog vida te programskih paketa i biblioteka potrebnih za kreiranje programa. Primjena prepoznavanja točaka na licu je u svakom slučaju široka te je to upravo čini i zanimljivom. Jedna od prednosti je svakako u jednostavnosti primjene ovakvog programa na različite aspekte života kao na primjer za prijašnje navedeno prepoznavanje emocija te stanja vozača u prometu. Uz kreiranje ovakvog programa se može puno naučiti o različitim načinima programiranja te različitim sustavima koji već postoje kako bi pomogli u stvaranju programa. Također je vrlo korisno za početnike u programiranju zbog velike količine dostupnih informacija na raznim web stranicama i drugim izvorima. No, uz izuzetno zanimljiv postupak kreiranja ovakvog programa došle su i razne prepreke s kojima se trebalo suočiti, od instaliranja samih biblioteka, pa sve do brojnih prepreka u pisanju koda. Obzirom da se ovakvi programi konstantno poboljšavaju, bilo je potrebno provesti puno vremena istraživajući te puno pokušaja i ponovnog pisanja koda kako bi se uspješno kreirao ovakav program. Moguća unaprjeđenja ovog programa su raznovrsna. Moguće je odrediti smjer pogleda osobe ili pak odrediti emocije na temelju položaja određenih dijelova lica. Tu bi se trebali ukomponirati dodatni paketi i biblioteke te je također moguć prelazak na duboko učenje ili pak učenje preko neuronskih mreža za puno složenije probleme.

LITERATURA

[1] >>Towards data science<<, 23. ožujak 2021. [Mrežno] Available: https://towardsdatascience.com/the-evolution-of-facial-recognition-a-case-study-in-the-transformation-of-deep-learning-73a738fcce67. [Pokušaj pristupa 28. lipanj 2022.]

[2] >>IBM<<, 15. srpanj 2021. [Mrežno] Available: https://www.ibm.com/cloud/learn/machine-learning. [Pokušaj pristupa 2. rujna 2022.]

[3] >>IBM<<, 15. srpanj 2021. [Mrežno] Available: https://www.ibm.com/topics/computer-vision#:~:text=Resources-,What%20is%20computer%20vision%3F,recommendations%20based%20on%20that%20information. [Pokušaj pristupa 3. rujna 2022.]

[4] >>Datagen<<, [Mrežno] Available: https://datagen.tech/guides/face-recognition/facial-landmarks/#:~:text=Facial%20landmark%20detection%20is%20a,nose%2C%20lips%2C%20and%20others. [Pokušaj pristupa 10. srpanj 2022.]

[5] >>Stack overflow<<, 24. ožujak 2021. [Mrežno] Available: https://stackoverflow.com/questions/67141844/python-how-to-get-face-mesh-landmarks-coordinates-in-mediapipe. [Pokušaj pristupa 17. srpanj 2022.]

[6] >>GeeksForGeeks<<, 5. kolovoz 2021. [Mrežno] Available: https://www.geeksforgeeks.org/opencv-overview/. [Pokušaj pristupa 10.rujan 2022.]

[7] >>GitHub<<, [Mrežno] Available: https://google.github.io/mediapipe/solutions/face_mesh.html. [Pokušaj pristupa 3.srpanj 2022.]

[8] >>LearnOpenCV<<, 19. ožujak 2018. [Mrežno] Available: https://learnopencv.com/facemark-facial-landmark-detection-using-opencv/. [Pokušaj pristupa 5. srpanj 2022.]

[9] Data science bloghaton, >>Facial Landmark detection simplified with OpenCV<<, 6. lipanj 2021. [Mrežno] Available: https://www.analyticsvidhya.com/blog/2021/07/facial-landmark-detection-simplified-with-opencv/#:~:text=OpenCV%20is%20the%20cross%2Dplatform,detection%2C%20face%20detection%2C%20etc. [Pokušaj pristupa 15. srpaj 2022.]

[10] Analitycs Vidhya, >>Facial Landmark detection in real time using OpenCV and dlib<<, 26. siječanj 2018. [Mrežno] Available: https://medium.com/analytics-vidhya/facial-landmark-detection-in-real-time-using-opencv-dlib-adbba9255329. [Pokušaj pristupa 10. kolovoz 2022.]

[11] >>TechTarget network<<, 1. veljača 2022. [Mrežno] Available: https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence. [Pokušaj pristupa 3. rujan 2022.]

[12] >>Coursera<< 10. kolovoz 2022. [Mrežno] Available: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python. [Pokušaj pristupa 5. rujan 2022.]

[13] >>Educative<<, 15. travanj 2020. [Mrežno] Available: https://www.educative.io/blog/object-oriented-programming. [Pokušaj pristupa 20.srpanj 2022.]

PRILOZI

I. Programski kod – osnovni

II. Programski kod – OOP

I. Programski kod - osnovni

```
import cv2
import mediapipe as mp
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_face_mesh = mp.solutions.face_mesh
drawing_spec = mp_drawing.DrawingSpec(color=(245,66,230),thickness=1,
circle_radius=1)
cap = cv2.VideoCapture(0)

with mp_face_mesh.FaceMesh(
    max_num_faces=3,
    refine_landmarks=True,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as face_mesh:
  while cap.isOpened():
    success, image = cap.read()
    image.flags.writeable = False
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = face_mesh.process(image)

    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    if results.multi_face_landmarks:
        for face_landmarks in results.multi_face_landmarks:
            mp_drawing.draw_landmarks(
                image=image,
                landmark_list=face_landmarks,
                connections=mp_face_mesh.FACEMESH_TESSELATION,
                landmark_drawing_spec=drawing_spec,
                connection_drawing_spec=drawing_spec
                )

            mp_drawing.draw_landmarks(
                image=image,
                landmark_list=face_landmarks,
                connections=mp_face_mesh.FACEMESH_CONTOURS,
                landmark_drawing_spec=drawing_spec,
                connection_drawing_spec=drawing_spec
```

```
                )
            mp_drawing.draw_landmarks(
                image=image,
                landmark_list=face_landmarks,
                connections=mp_face_mesh.FACEMESH_IRISES,
                landmark_drawing_spec=drawing_spec,
                connection_drawing_spec=drawing_spec
                )



    cv2.imshow("kamera", image)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

cap.release()
cv2.destroyAllWindows()
```

II. Programski kod - OOP

```
import cv2
import mediapipe as mp
import time

class FaceMeshDetector():

    def __init__(self, StaticMode = False, max_num_faces = 2
                 , refineLandmarks = False, minDetectionCon = 0.5,
minTrackCon = 0.5):
        self.StaticMode = StaticMode
        self.max_num_faces = max_num_faces
        self.refineLandmarks = refineLandmarks
        self.minDetectionCon = minDetectionCon
        self.minTrackCon = minTrackCon




        self.mp_drawing = mp.solutions.drawing_utils
```

```
        self.mp_drawing_styles = mp.solutions.drawing_styles
        self.mp_face_mesh = mp.solutions.face_mesh
        self.drawing_spec = self.mp_drawing.DrawingSpec(color=(245, 66,
230), thickness=1, circle_radius=1)
        self.face_mesh = self.mp_face_mesh.FaceMesh()
        self.mp_face_mesh.FaceMesh(self.StaticMode,
          self.max_num_faces,
          self.refineLandmarks,
          self.minDetectionCon,
          self.minTrackCon)


    def findFaceMesh(self, image, draw = True):
        image.flags.writeable = True
        self.image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        self.results = self.face_mesh.process(self.image)
        faces = []
        if self.results.multi_face_landmarks:
            for face_landmarks in self.results.multi_face_landmarks:
                if draw:
                  self.mp_drawing.draw_landmarks(
                    image=image,
                    landmark_list=face_landmarks,
                    connections=self.mp_face_mesh.FACEMESH_TESSELATION,
                    landmark_drawing_spec=self.drawing_spec,
                    connection_drawing_spec=self.drawing_spec
                   )

                  self.mp_drawing.draw_landmarks(
                    image=image,
                    landmark_list=face_landmarks,
                    connections=self.mp_face_mesh.FACEMESH_CONTOURS,
                    landmark_drawing_spec=self.drawing_spec,
                    connection_drawing_spec=self.drawing_spec
                   )




                face = []
```

```
                for id, landmark in enumerate(face_landmarks.landmark):
                        ih, iw, ic = image.shape
                        x, y = int(landmark.x*iw), int(landmark.y*ih)

                        print(id, x, y)
                        face.append([x, y])
                faces.append(face)
        return image




def main():

    cap = cv2.VideoCapture(0)
    detector = FaceMeshDetector()
    while cap.isOpened():
            success, image = cap.read()
            image = detector.findFaceMesh(image)



            image.flags.writeable = False
            cv2.imshow("kamera", image)
            if cv2.waitKey(1) & 0xFF == ord("q"):
                break




if __name__ == "__main__":
    main()
```