Sustav za detekciju, prepoznavanje i interakciju s ljudima

Prpić, Rea

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni prijediplomski studij računarstva

Završni rad

Sustav za detekciju, prepoznavanje i interakciju s ljudima

Rijeka, rujan 2023.

Rea Prpić
0069082840

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni prijediplomski studij računarstva

Završni rad

Sustav za detekciju, prepoznavanje i interakciju s ljudima

Mentor: prof. dr. sc. Ivan Štajduhar

Rijeka, rujan 2023.

Rea Prpić
0069082840

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 10. ožujka 2021.

Zavod: **Zavod za računarstvo**
Predmet: **Uvod u umjetnu inteligenciju**
Grana: **2.09.04 umjetna inteligencija**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Rea Prpić (0069082840)**
Studij: Preddiplomski sveučilišni studij računarstva

Zadatak: **Sustav za detekciju, prepoznavanje i interakciju s ljudima / A system for detecting, recognising and interacting with people**

Opis zadatka:

Zadatak je napraviti prototip sustava za jednostavnu interakciju s pojedincima u neposrednoj blizini. Sustav putem jedne monokromatske kamere prepoznaje ljude u blizini te ih putem dodirnog ekrana pozdravlja i poziva na čavrljanje i interakciju. Razmotriti i opisati tehnike analize slike kojima se prepoznaju ljudi u kadru. Razmotriti i opisati tehnike analize slike kojima se prepoznaje lice pojedinca. Osmisliti i implementirati protokol za prepoznavanje i površnu interakciju sa zainteresiranim pojedincima. Implementirati odgovarajuće tehnike analize slike i pohrane podataka. Sustav izgraditi na ugradbenoj arhitekturi RaspberryPi. Testirati i vrednovati rješenje u realnim uvjetima.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 15. ožujka 2021.

Mentor:

Izv. prof. dr. sc. Ivan Štajduhar

Predsjednik povjerenstva za završni ispit:

Izv. prof. dr. sc. Kristijan Lenac

Izjava o samostalnoj izradi rada

Izjavljujem da sam završni rad pod naslovom „Sustav za detekciju, prepoznavanje i interakciju s ljudima“ izradila samostalno.

Rijeka, rujan 2023.

Rea Prpić

Sadržaj

1 Uvod

Tema ovog završnog rada bila je izraditi prototip sustava za jednostavnu interakciju s pojedincima u neposrednoj blizini. Sustav je implementiran na ugradbenoj arhitekturi Raspberry Pi 3 - Model B zajedno sa Raspberry Pi modulom kamere i Raspberry Pi ekranom osjetljivim na dodir.

Pojedincima u neposrednoj blizini kada su detektirani od strane sustava nastoji se zaokupiti pažnja zvučnim signalom kojeg emitira Piezo zvučnik i porukom ispisanom na grafičkom sučelju kojom se pozivaju da se približe sustavu kako bi im lice bilo detektirano. Detekcijom lica sustav ili prepoznaje pojedinca ili ukoliko ga ne prepoznaje nudi mogućnost upoznavanja prikazivanjem okvira za unos imena nepoznatog pojedinca. Kada je pojedinac poznat sustavu, sustav ga personalizirano pozdravlja i generira zabavnu činjenicu o korisničkom imenu.

U ovom završnom radu opisane su korištene tehnike za detekciju osoba i lica (Viola - Jones algoritam) te prepoznavanje (histogram lokalnih binarnih uzoraka) lica. Prikazana je struktura izrađenog projekta te su izdvojeni i opisani ključni koncepti i funkcije unutar svake datoteke. Cijeli sustav je implementiran koristeći programski jezik Python 3.9, korisničko sučelje je izrađeno s bibliotekom PyQt5, dok su funkcionalnosti računalnog vida ostvarene primjenom OpenCV (engl. *Open Source Computer Vision Library*) biblioteke.

Računalni vid je područje umjetne inteligencije koje omogućava računalima da interpretiraju i razumiju vizualne podatke iz okoline. Cilj računalnog vida je omogućiti računalnim sustavima da obavljaju zadatke koji zahtjevaju vizualnu spoznaju. Analizom slika i video snimki računala dobivaju vizualne informacije na temelju kojih donose odluke ili obavljaju određene radnje. Primjena računalnog vida se može

vidjeti u brojnim industrijama. To uključuje automobilsku industriju koja koristi ovu tehnologiju za autonomno vođenje vozila, zdravstvenu industriju gdje se koristi za rendgenske dijagnostičke postupke, kao i sigurnosne sustave koji implementiraju tehnologiju prepoznavanja lica te mnoge druge.

2 Detekcija

Detekcija objekata je tehnika računalnog vida koja omogućuje pronalaženje objekata na slikama ili videozapisima. Razvijeno je mnogo metoda i algoritama za detekciju objekata, svaka s vlastitim prednostima i nedostacima. Jedan od najpoznatijih algoritama u ovoj domeni je Viola-Jones algoritam, koji je posebno poznat po svojoj primjeni u detekciji lica.

2.1 Viola-Jones algoritam

Paul Viola i Michael Jones predstavili su 2001. godine rad pod nazivom "*Rapid Object Detection using a Boosted Cascade of Simple Features*"[1] u kojem opisuju inovativni pristup detekciji objekata s posebnim naglaskom na detekciju lica u stvarnom vremenu. Ovaj algoritam sposoban je izuzetno brzo obraditi slike u stvarnom vremenu i postići visoku preciznost u detektiranju objekata.

Viola-Jones algoritam za detekciju sastoji se od četiri glavna koraka:

1. Izračunavanje integralne slike
2. Haar-ove značajke
3. Adaboost učenje
4. Kaskadni klasifikatori

2.1.1 Integralna slika

Prvi korak u algoritmu je izračunavanje integralne slike. Integralna slika je posebna vrsta slike koja se koristi u Viola-Jones algoritmu za brže izračunavanje Haar-ovih značajki. Ova tehnika značajno ubrzava proces detekcije objekata na slici, uključujući lica.

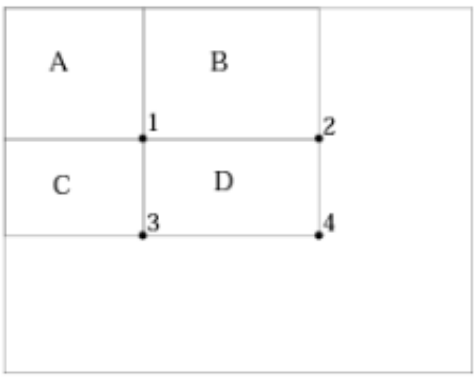
Integralna slika se generira iz izvorne slike tako da svaki piksel u integralnoj slici predstavlja sumu svih piksela iznad i lijevo od njega, uključujući i njega, u izvornoj slici. Prvi red i prvi stupac integralne slike sadrže kopirani piksel iz izvorne slike.

Ovakva reprezentacija slike omogućuje brzo izračunavanje suma piksela unutar bilo kojeg pravokutnog područja na slici. Umjesto da se svaki put iznova zbrajaju pikseli unutar određenog područja za svaku Haar-ovu značajku, Viola-Jones algoritam može jednostavno koristiti integralnu sliku za brzo izračunavanje tih suma. Kada se Haar-ove značajke primjenjuju na određeno područje slike, umjesto da se zbrajaju pikseli unutar tog područja, Viola-Jones algoritam jednostavno koristi četiri vrijednosti iz integralne slike kako bi brzo izračunao sume piksela. Ovakav način je znatno brži i učinkovitiji od izravnog zbrajanja piksela.

Integralna slika omogućuje Viola-Jones algoritmu da ubrza proces detekcije objekata, uključujući lica, jer Haar-ove značajke mogu biti izračunate mnogo brže koristeći ovakvu reprezentaciju. To čini algoritam prikladnim za korištenje u stvarnom vremenu i učinkovitim u raznim aplikacijama detekcije objekata na slikama.

Primjer izračuna integralne slike

- Vrijednost integralne slike na lokaciji 1 je zbroj piksela u pravokutniku A
- Vrijednost integralne slike na lokaciji 2 jednaka je zbroju pravokutnika A i B
- Vrijednost integralne slike na lokaciji 3 jednaka je zbroju pravokutnika A i C
- Vrijednost integralne slike na lokaciji 4 jednaka je zbroju pravokutnika A, B, C i D
- Zbroj unutar pravokutnika D može se izračunati kao vrijednost u točki 4 + vrijednost u točki 1 - (vrijednost u točki 2 + vrijednost u točki 3)

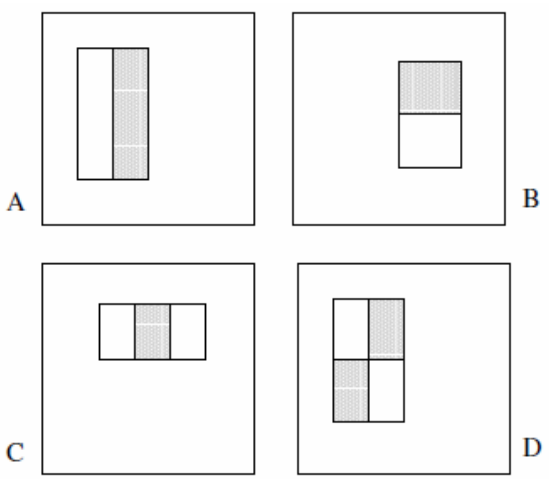


Slika 2.1 Primjer izračuna integralne slike[1]

2.1.2 Haar-ove značajke

Haar-ove značajke su kvadratni ili pravokutni obrasci koji se koriste za detekciju različitih oblika ili struktura na slici, poput lokalnih promjena u intezitetu svjetlosti.

Ovaj korak uključuje odabir i izračunavanje Haar-ovih značajki koje se koriste za prepoznavanje različitih dijelova lica i drugih objekata na slici.



Slika 2.2 Haar-ove značajke[1]

Kada se ovi obrasci primijene na sliku, izračunava se razlika u intezitetu svijetlosti između svjetlih i tamnih dijelova unutar tih oblika.

Ovi osnovni oblici Haar-ovih značajki često se upotrebljavaju kao temelji za izgradnju složenijih značajki koje se koriste u detekciji lica i drugih objekata. Kombinacijom različitih oblika Haar-ovih značajki, moguće je generirati mnoštvo raznolikih oblika i uzoraka koji doprinose uspješnijem detektiranju objekata na slici.

2.1.3 Adaboost učenje

AdaBoost (Adaptive Boosting) je algoritam strojnog učenja koji se koristi za odabir najvažnijih značajki iz velikog broja potencijalnih značajki i izradu klasifikatora za detekciju lica. Počinje treniranjem slabog klasifikatora na cjelokupnom skupu podataka. Nakon što izračuna pogrešku, AdaBoost algoritam trenira sljedeći klasifikator fokusirajući se više na primjere koji su prvotno pogrešno klasificirani.

Nakon što se identificiraju najvažnije značajke, AdaBoost koristi te odabrane značajke za složeniju analizu svake slike. Koristeći odabrane značajke, algoritam procjenjuje je li lice prisutno u svakom potprozoru slike.

Posebna vrijednost algoritma AdaBoost leži u njegovoj sposobnosti usmjeravanja pažnje na one primjere koji su bili zahtjevniji za klasifikaciju u prethodnim iteracijama, odnosno na one potprozore koji su prvo pogrešno klasificirani kao sadrže lice ili ne sadrže lice. Time se poboljšava ukupna učinkovitost i preciznost detekcije lica. Kroz ovaj proces, AdaBoost stvara snažan klasifikator za detekciju lica, koristeći grupu slabih klasifikatora, pri čemu svaki slijedeći klasifikator optimizira ispravljanje pogrešaka svog prethodnika.

2.1.4 Kaskadni klasifikatori

Posljednja faza u postupku detekcije lica koju provodi Viola-Jones algoritam uključuje upotrebu kaskadnih klasifikatora. Ovi klasifikatori su poput serije strogih kontrola koje slika mora proći. Svaki klasifikator njihovog niza posebno je iztreniran da prepozna određene značajke lica koristeći Haar-ove značajke, koje se kreću od najosnovnijih do najdetaljnijih.

Početni klasifikatori u nizu identificiraju vrlo osnovne značajke, poput lokalnih promjena intenziteta svjetlosti, što bi moglo ukazivati na prisustvo očiju ili nosa na slici. Ukoliko slika uspješno prođe ove uvodne faze, ona se dalje šalje složenijim klasifikatorima. Napredniji klasifikatori traže detaljnije značajke lica, što rezultira preciznijim prepoznavanjem.

Ključni aspekt ovog procesa je to što čim slika ne prođe bilo koji od klasifikatora, odmah se odbacuje kao "negativna", tj. označava se kao slika koja ne sadrži lice. Ovakav pristup značajno ubrzava proces detekcije lica, jer slike koje očito ne sadrže lica ne moraju proći kroz cijelu kaskadu klasifikatora.

3 Prepoznavanje

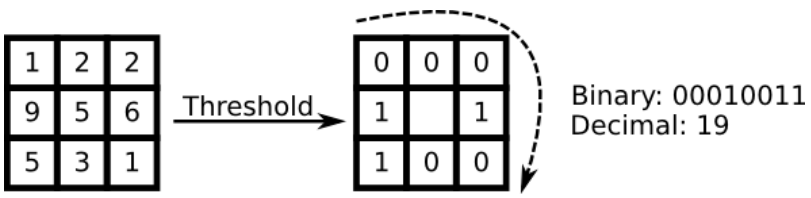
Prepoznavanje lica je ključna značajka računalnog vida koja omogućuje identifikaciju osobe na temelju značajki njenog lica. Ova tehnologija nalazi svoju primjenu u različitim područjima - od sigurnosnih sustava do raznih interakcija s korisnicima. Postoje brojni pristupi za prepoznavanje lica, jedan od njih je algoritam temeljen na histogramu lokalnih binarnih uzoraka (engl. *Local Binary Pattern Histogram*, LBPH)[2].

3.1 Histogrami lokalnih binarnih uzoraka

Histogram lokalnih binarnih uzoraka predstavlja jednostavan no učinkovit pristup u prepoznavanju lica. Zasniva se na tehnici lokalnih binarnih uzoraka (LBP) koja analizira lokalne regije lica.

Prvi korak je podjela slike lica na male kvadratne djelove. Na svakom od tih djelova implementira se tehnika za ekstrakciju značajki na slikama - lokalni binarni uzorak (LBP). Svaki piksel na slici uspoređuje se sa susjednim pikselima. Ukoliko je vrijednost središnjeg piksela manja od vrijednosti susjeda, susjed se označava se kao svijetliji sa vrijednošću 1, a ukoliko je vrijednost središnjeg piksela veća od vrijednosti susjeda, susjed se označava kao tamniji sa vrijednošću 0.

Nakon generiranja binarnog uzorka za svaki piksel, binarni uzorak svakog piksela se koristi za stvaranje binarne slike. Svaki piksel u ovoj binarnoj slici sadržava binarni uzorak koji je kreiran uspoređivanjem intenziteta svjetlosti tog piksela s intenzitetom svjetlosti njegovih susjednih piksela. Zatim se svaki binarni uzorak, koji je predstavljen kao binarni broj, pretvara u decimalni ekvivalent.



Slika 3.1 Tehnika lokalnih binarnih uzoraka[3]

Nadalje, za svaku se regiju kreira histogram koji pokazuje raspodjelu tih binarnih uzoraka. Histogram predstavlja broj puta koliko se svaki binarni uzorak odnosno koliko se često određeni teksturni uzorak pojavljuje unutar regije. Na taj način, svaka regija dobiva svoj jedinstveni "otisak" u obliku histograma.

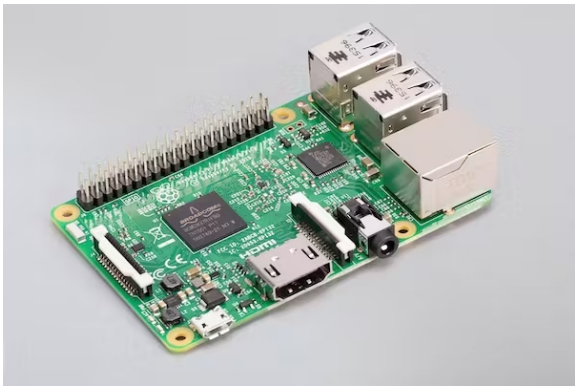
Na kraju se histogrami svih regija kombiniraju u jedan konačni histogram koji predstavlja cjelokupno lice. Ovaj konačni histogram sadrži ključne informacije o teksturi i strukturi lica koje su bitne za njegovo prepoznavanje.

Algoritam histograma lokalnih binarnih uzoraka je učinkovit i stabilan, jer analizira lokalne značajke slike, a ne globalne. To znači da male promjene u izgledu lica (npr. promjene u osvjetljenju, izrazima lica itd.) neće bitno utjecati na rezultate prepoznavanja. Također zahtijeva manje resursa, jer nije toliko računski zahtjevan kao neke druge tehnike za prepoznavanje lica.

4 Razvojna oprema

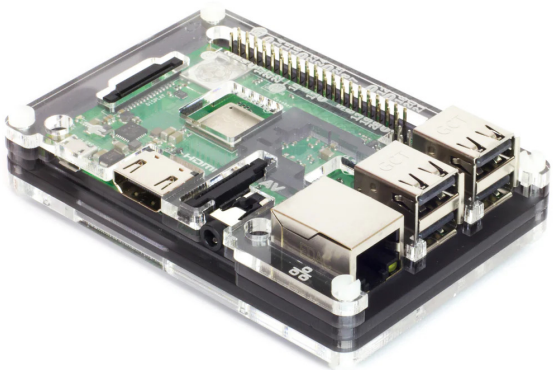
Ovaj završni rad izrađen je na ugradbenoj arhitekturi Raspberry Pi koja je popularan izbor za širok raspon aplikacija zbog svoje kompaktne veličine, pristupačnosti i niskoj potrošnji energije. Specifičan model Raspberry Pi ugradbene arhitekture koji se koristio za realizaciju ovog završnog rada je Raspberry Pi 3 Model B[4]. To je najraniji model treće generacije Raspberry Pi te je zamijenio Raspberry Pi 2 Model B u veljači 2016. Uređaj je temeljen na Broadcom BCM2837 64-bitnom četverojezgrenom procesoru takta 1,2 GHz što uvelike poboljšava njegovu izvedbu u odnosu na prethodne modele te također posjeduje 1 GB RAM (engl. *Random Access Memory*) memorije. Sadrži četiri USB (engl. *Universal Serial Bus*) 2.0 priključka i HDMI (engl. *High-Definition Multimedia Interface*) 1.3/1.4 video izlaz te ugrađene mogućnosti povezivanja kao što su Ethernet, Wi-Fi i Bluetooth, a GPIO (engl. *General-purpose input/output*) pinovi koji se nalaze na uređaju omogućuju povezivanje s drugom elektronikom,

U razvojnu opremu također je uključeno službeno Raspberry Pi napajanje (5,1 V/2,5 A), Ninja Coupé PiBow Raspberry Pi kućište, Raspberry Pi modul kamere (Rev 1.3), Raspberry Pi 7" dodirni zaslon zajedno s Pimoroni - Royale 7" okvirom te Piezo zvučnik.



Slika 4.1 Raspberry Pi 3 Model B[5]

Slika 4.2 Raspberry Pi službeno napajanje[6]



Slika 4.3 Ninja Coupé PiBow Raspberry Pi kućište[7]

Slika 4.4 Raspberry Pi modul kamere (Rev 1.3)[8]

Slika 4.5 Raspberry Pi 7" dodirni zaslon[9]

Slika 4.6 Pimoroni - Royale 7" okvir za dodirni zaslon[10]

Slika 4.7 Piezo zvučnik[11]

5 Postavljanje i konfiguracija sustava

Prije svega potrebno je povezati sve dijelove opreme koja će se koristiti za razvoj te instalirati odgovarajući operacijski sustav. Za ovaj projekt biti će korišten službeni operacijski sustav Raspberry Pi OS[12] instaliran koristeći Raspberry Pi Imager. Raspberry Pi Imager je besplatni uslužni alat otvorenog koda koji se koristi za instaliranje operacijskih sustava na microSD karticu za korištenje s Raspberry Pi računalom.

Nakon instalicije operacijskog sustava na microSD karticu, promijenjene su postavke unutar konfiguracijske datoteke Raspberry Pi-a. Konfiguracijska datoteka `config.txt` Rapberry Pi-a koristi se za pohranu konfiguracijskih parametara koji se čitaju prilikom pokretanja uređaja. Unutar `config.txt` datoteke koja se nalazi u `boot` direktoriju uređaja podešena je orijentacija dodirnog zaslona. Naredba `lcd_rotate = 2` označava okretanje dodirnog zaslona za 180 stupnjeva.

Raspberry Pi koristi zajednički memorijski sustav gdje i centralna procesorska jedinica i grafički procesor dijele memoriju. Podjela je definirana na temelju postavki u konfiguracijskoj datoteci `config.txt`. Definiranjem `gpu_mem = 128` dodijeljujemo 128 MB ukupne memorije sustava grafičkom procesoru što je 64 MB više od zadane vrijednosti. Količina memorije koja je dodijeljena grafičkom procesoru može utjecati na izvedbu grafičkih zadataka, a s obzirom da su i kamera i dodirni zaslon oboje grafički intenzivni odnosno zahtijevaju značajnu količinu grafičke procesorske snage za rad, povećanjem memorije koju dodijeljujemo grafičkom procesoru poboljšavaju se i performanse cjelokupnog sustava. Također i biblioteka OpenCV ima koristi od

povećanja dodijeljene memorije grafičkom procesoru s obzirom da grafički procesor može ubrzati izvođenje određenih operacija koje uključuju obradu vizualnih podataka te i na taj način unaprijediti učinkovitost sustava.

Nakon povezivanja Raspberry Pi modula kamere sa sustavom potrebno ju je aktivirati u postavkama operacijskog sustava kako bi se korsitila.

OpenAI API (engl. *Application Programming Interface*) ključ koji će biti korišten za generiranje zanimljivosti o imenu pojedinaca, definiran je unutar `.bashrc` datoteke. Datoteka `.bashrc` je konfiguracijska datoteka koja omogućuje korisnicima konfiguracije poput postavljanja varijabli okruženja. Kada korisnik otvori novu sesiju terminala, datoteka `.bashrc` se automatski izvršava.

Sljedećom naredbom otvara se `.bashrc` datoteka i uređuje pomoću uređivača teksta nano kako bi OpenAI API ključ bio dopisan:

```
nano .bashrc
```

Nakon konfiguracije uređaja potrebno je instalirati potrebnu programsku podršku. U nastavku su navedene ključne naredbe tog procesa i opis istih.

Provodi se provjera postoji li potreba za ažuriranjem postojećih paketa te ukoliko postoji isti se ažuriraju:

```
sudo apt update && sudo apt upgrade -y
```

Zatim se instalira pip alat za instaliranje Python paketa:

```
sudo apt install python3-pip
```

Instalacija biblioteka numpy i pillow (Python Imaging Library) koje se koriste za numeričku obradu i obradu slika:

```
pip3 install opencv-contrib-python numpy pillow
```

OpenCV[13] je biblioteka otvorenog koda za računalni vid i strojno učenje. Nastala je s ciljem pružanja raznovrsnih funkcionalnosti za aplikacije računalnog vida, olakšavajući time rad s vizualnim podacima.

Instalacija biblioteke OpenCV:

```
pip3 install opencv-contrib-python==4.5.3.56
```

PyQt5[14] je popularni razvojni okvir za razvoj aplikacija koji omogućuje stvara-

nje grafičkih korisničkih sučelja.

Instalacija PyQt5 koji će se koristiti za izradu korisničkog sučelja te paketa koji se koriste za manipulaciju slikovnim podacima i numeričke izračune vrši se naredbom:

```
sudo apt-get install -y libhdf5-dev libhdf5-serial-dev python3-pyqt5
libatlas-base-dev libjasper-dev
```

Provjera statusa modula kamere spojenog na Raspberry Pi:

```
vcgencmd get_camera
```

6 Struktura projekta

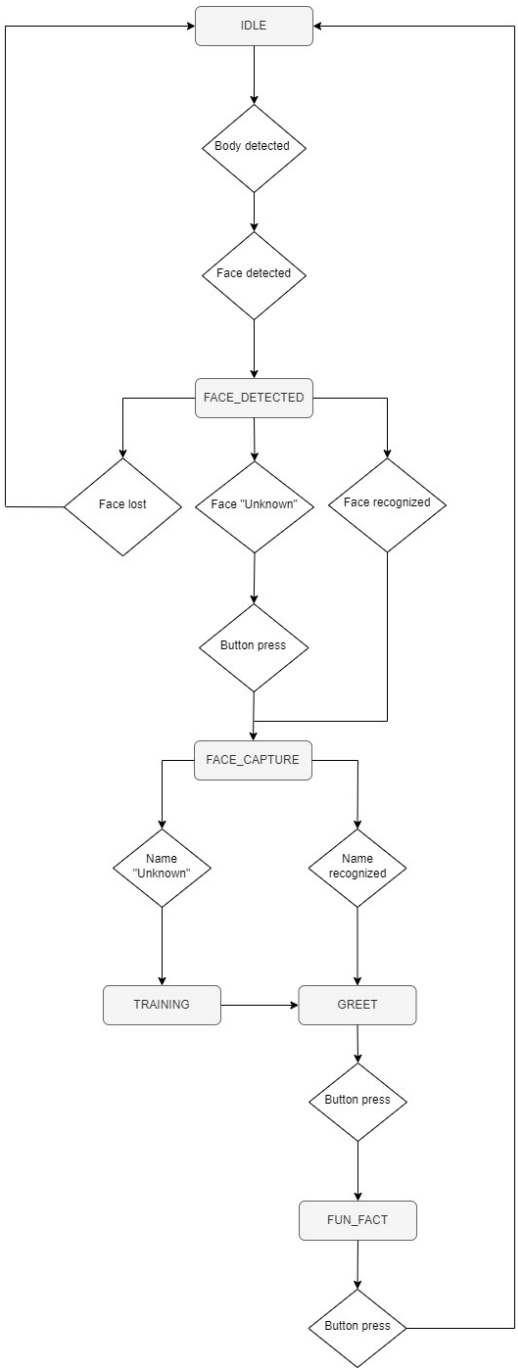
Projekt je podijeljen u 7 datoteka:

- `fsm.py` – datoteka unutar koje se nalazi logika za upravljanjem stanjem aplikacije i njenim kontrolnim tokom
- `main.py` – temeljna datoteka projekta odgovorna za koordinaciju i kontrolu funkcija
- `helpers.py` – datoteka unutar koje su definirane pomoćne funkcije
- `ui.py` – datoteka unutar koje je definirano korisničko sučelje
- `vision.py` – datoteka unutar koje su definirane funkcije vezane uz računalni vid
- `mapping.json` – JSON datoteka unutar koje je pohranjeno mapiranje između oznaka i imena potrebnih za prepoznavanje lica
- `model.yml` – datoteka koja se koristi za pohranjivanje istreniranog modela za prepoznavanje lica

6.1 Automat konačnih stanja

Automat konačnih stanja (engl. *Finite State Machine*, FSM) kontrolira cjelokupni tok aplikacije upravljajući interakcijama različitih podsustava. Njegov dizajn pruža jasan, strukturiran način za upravljanje ponašanjima koja ovise o stanju u procesu. Svako stanje predstavlja određenu fazu u tijeku aplikacije te se u svakom stanju izvodi drugačiji skup operacija. Implementacija automata konačnih stanja olakšava

održavanje koda, jer su stanja jasno definirana te promjene u jednom stanju ne utječu na ostatak sustava.



Slika 6.1 Automat konačnih stanja

6.2 main.py

Datoteka `main.py` je osnovna datoteka koja usklađuje i nadzire funkcije sustava.

Na početku se inicijaliziraju klase `Vision`, `SimpleUI` i `FSM`. Zatim se sučelje maksimizira kako bi prekrilo cijeli ekran. Sustav učitava model za prepoznavanje lica i `mapping.json` datoteku koja sadrži mapiranje korisničkih imena na oznake.

U glavnoj petlji, sustav kontinuirano obrađuje *frame-ove*, detektira lica i osobe te komunicira s automatom konačnih stanja kako bi odlučio o odgovarajućim akcijama ili promjenama stanja.

Ukoliko sustav prepozna lice, informacije o prepoznatom licu pohranjuju se u `fsm`, a *frame* s prepoznatim licima prikazuje se u korisničkom sučelju.

U slučaju bilo kakvih prekida, poput prekida s tipkovnice ili zatvaranja PyQt5 prozora, program se zaustavlja, prekidajući video snimanje i zatvarajući sve OpenCV prozore.

6.3 ui.py

Korisničko sučelje projekta kreirano je koristeći PyQt5, biblioteku za izradu grafičkih korisničkih sučelja. Glavni prozor sučelja, klasa `SimpleUI`, prikazuje video zapis u stvarnom vremenu s kamere, a također ima i statusnu poruku koja korisnicima pruža povratne informacije. Tu je i akcijski gumb koji korisnicima omogućava interakciju sa sustavom, npr. da započnu postupak snimanja lica. Kada je potrebno unijeti ime novog korisnika, pojavljuje se poseban dijaloški prozor `NameInputDialog`. Ovaj dijaloški prozor omogućava unos teksta i sadrži opciju za prikaz virtualne tipkovnice.

6.4 helpers.py

Datoteka `helpers.py` sadrži pomoćne funkcije za kontrolu Piezo zvučnika spojenog na Raspberry Pi, za pozdravljanje prepoznatog korisnika i interakciju s OpenAI GPT-3.5 modelom za generiranje činjenica o imenu prepoznatog korisnika.

6.4.1 Funkcija `beep()`

Funkcija `beep()` sadrži logiku za oglašavanje Piezo zvučnika spojenog na Raspberry Pi GPIO pin 26, dva puta uzastopno s kratkom odgodom između. Funkcija prvo bilježi trenutno vrijeme i uspoređuje ga s vremenom kada se zvučnik zadnji put oglasio. Ukoliko je od zadnjeg zvučnog signala proteklo manje od 10 sekundi, zvučnik se ne oglašava.

```
def beep(frequency=1000, first_duration=0.05, second_duration
   =0.1):
    global last_beep_time
    current_time = time.time()

    if current_time - last_beep_time < 10:
        return

    pwm = GPIO.PWM(BUZZER_PIN, frequency)
    pwm.start(100)
    time.sleep(first_duration)
    pwm.stop()

    time.sleep(0.1)

    pwm.start(100)
    time.sleep(second_duration)
    pwm.stop()
    last_beep_time = current_time
```

Kodni isječak 6.1 Funkcija `beep()`

6.4.2 Funkcija `get_fun_fact()`

U sljedećem isječku koda prikazana je interakcija sa OpenAI API-jem za dohvaćanje činjenica o određenom imenu. Na početku se dohvaća OpenAI API ključ. Ukoliko ključ nije pronađen, javlja se `ValueError` koji označava da OpenAI API ključ nije

ispravno postavljen kao varijabla okruženja.

Funkcija `get_fun_fact()` namijenjena je generiranju činjenice o određenom imenu koristeći GPT-3.5 model. Prilikom pozivanja funkcije kreira se nova `ChatCompletion` instanca s nekoliko definiranih parametara: `model` je postavljen kao "gpt-3.5-turbo", `max_tokens` je ograničen na 100 za kontrolu duljine odgovora, a `messages` je lista dva *dictionary-a* od kojih svaki ima dva ključ-vrijednost para. `role` ključ može biti `system`, `user` ili `assistant` i označava tko upućuje poruku, a `content` ključ sadrži sami tekst poruke. Prva poruka je iz sustava i daje upute GPT-3.5 modelu o generiranju činjenice o određenom imenu, dok je druga poruka od korisnika i sadrži korisnički unos, odnosno ime o kojem GPT-3.5 model treba izgenerirati činjenicu. `Try` blok hvata sadržaj odgovora prvog izbora i vraća ga. U slučaju bilo kakvih iznimaka ili pogrešaka tijekom izvođenja `try` bloka (poput greške API-ja), `except` blok hvata iznimku i ispisuje poruku o pogrešci. Zatim vraća `None`, što pokazuje da nijedna činjenica nije mogla biti dohvaćena zbog greške.

```
OPENAI_API_KEY = os.environ.get('OPENAI_API_KEY')
if not OPENAI_API_KEY:
    raise ValueError("OpenAI API key not set as environment variable")
openai.api_key = OPENAI_API_KEY

def get_fun_fact(name):
    try:
        completion = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            max_tokens=100,
            messages=[
                {"role": "system", "content": "\
                You provide very brief, concise, and fun facts about\
                the user's name. You do not output anything but the fun fact."},
                {"role": "user", "content": "{}".format(name)}
            ])
```

```
17          return completion.choices[0].message.content
18
19      except Exception as e:
20          print(f"Error fetching fun fact: {e}")
21          return None
```

Kodni isječak 6.2 Prikaz interakcije sa OpenAI API-jem

6.5 fsm.py

Unutar `fsm.py` datoteke implementiran je automat konačnih stanja.

Na početku je definirana enumeracija naziva `State` unutar koje su definirana različita stanja automata konačnih stanja. Stanja unutar kojih se sustav može nalaziti su:

- `IDLE` – početno stanje sustava
- `FACE_DETECTED` – stanje u koje se prelazi nakon što je detektirano lice
- `FACE_CAPTURE` – stanje u koje se prelazi ukoliko je potrebno snimiti podatke o licu
- `TRAINING` – stanje u kojem se trenira model nakon što je prikupljeno dovoljno podataka o licu
- `GREET` – stanje u kojem se pozdravlja prepoznati korisnik
- `FUN_FACT` – stanje unutar kojeg sustav generira zabavne činjenice o imenu prepoznatog korisnika

Automat konačnih stanja implementiran unutar `FSM` klase koristi višedretvenost za postavljanje vremenskih ograničenja za prijelaze stanja s ciljem uvođenja odgode prije promjene stanja.

6.5.1 Funkcija `start_timer()`

```
def start_timer(self, seconds=2, state=State.IDLE):
        def on_timer_expire():
            print("Timer expired!")
            self.transition(state)

        if self.timer_thread is not None and self.timer_thread.
   is_alive():
            return

        if self.timer_thread is not None:
            self.timer_thread.cancel()

        self.timer_thread = threading.Timer(seconds,
   on_timer_expire)
        self.timer_thread.start()
```

Kodni isječak 6.3 Funkcija `start_timer()` unutar datoteke `fsm.py`

Unutar funkcije `start_timer()` u datoteci `fsm.py` implementirano je pokretanje odbrojavanja od dvije sekunde prije nego što sustav prijeđe u drugo stanje. Koristi se Pythonov `threading` modul kojim se stvara nova dretva odvojena od glavne dretve, za implementaciju funkcije mjerača vremena. Stvara se nova dretva koja čeka dvije sekunde, a nakon tog perioda aktivira se funckija `on_timer_expire()` definirana unutar `start_timer()` funkcije koja prebacuje sustav iz jednog stanja u drugi. Omogućuje sustavu prijelaz u novo stanje nakon vremenske odgode bez zaustavljanja izvršavanja programa tijekom perioda čekanja. Prednost korištenja višedretvenosti ovdje je kontrola prijelaza stanja. Sustav sa razdobljem čekanja dobiva dovoljno vremena za izvršavanje zadataka i neometani prijelaz stanja.

6.5.2 Funkcija `update()`

U sljedećem isječku koda prikazana je funkcija `update()` koja upravlja različitim stanjima u kojima se sustav može nalaziti.

```
def update(self, frame=None):
     if self.state == State.IDLE:
         self.ui.update_button(False)
         self.ui.update_status("")
         if len(self.detected_bodies) > 0 and not self.
body_in_frame:
             print("Body detected!")
             self.ui.update_status("Hi there! Please come
closer so I can take a look at you.")
             beep()
             self.body_in_frame = True
         elif len(self.detected_faces) > 0 and self.
body_in_frame:
             print("Face detected!")
             self.ui.update_status("I see you! Please wait
while I recognize you.")
             self.body_in_frame = False
             self.transition(State.FACE_DETECTED)

     elif self.state == State.FACE_DETECTED:
         if self.recognized_faces:
                 self.closest_face = max(self.
recognized_faces, key=lambda face: face[2] * face[3])
                 if self.timer_thread is None or not self.
timer_thread.is_alive() :
                     self.start_timer(2, State.FACE_CAPTURE)
                 name = self.closest_face[4]
         if len(self.detected_faces) == 0:
             print("Face lost!")
             if self.timer_thread is not None:
                 self.timer_thread.cancel()
```

```
26                 self.transition(State.IDLE)
27             else:
28                 if 'name' not in locals() or not name or name
   == "Unknown":
29                     self.ui.update_status("I'm trying to
   recognize you, please wait a moment or press the button
   below if we haven't met before.")
30                     self.ui.update_button(True, "I'm new!")
31                     if self.timer_thread is not None:
32                         self.timer_thread.cancel()
33                 else:
34                     pass
35
36         elif self.state == State.FACE_CAPTURE:
37             self.ui.update_status("Nice to meet you! Please
   enter your name and look at the camera while I memorize your
    face.")
38             self.ui.update_button(False)
39
40             recognized_faces = self.recognized_faces
41
42             if recognized_faces:
43                 closest_face = max(recognized_faces, key=lambda
    face: face[2] * face[3])
44                 x, y, w, h, name = closest_face
45
46                 if name == "Unknown":
47                     if self.current_name is None:
48                         self.current_name = self.ui.
   get_name_from_user()
49
50                     if self.capture_count < 30 and self.
   current_name and closest_face:
51                         self.vision.capture_faces(closest_face,
    self.current_name, frame, self.capture_count)
```

```
52                          self.capture_count += 1
53                      elif self.capture_count >= 30:
54                          self.capture_count = 0
55                          self.transition(State.TRAINING)
56                  else:
57                      self.current_name = name
58                      self.transition(State.GREET)
59          elif self.state == State.TRAINING:
60                  self.ui.update_status("I am training my
    recognizer, please wait...")
61                  images, labels = self.vision.process_faces()
62                  self.vision.train_model(images, labels)
63                  self.vision.save_face_model('model.yml')
64                  print("Model trained and saved!")
65                  self.vision.save_name_label_mapping('mapping.
    json')
66
67                  self.transition(State.GREET)
68          elif self.state == State.GREET:
69              if not self.next_state_ready:
70                  self.ui.update_status(get_greeting(self.
    current_name))
71                  self.ui.update_button(True, "Fun Fact")
72                  self.next_state_ready = True
73          elif self.state == State.FUN_FACT:
74              if not self.next_state_ready:
75                  self.ui.update_status(get_fun_fact(self.
    current_name))
76                  self.ui.update_button(True, "Goodbye")
77                  self.current_name = None
78                  self.next_state_ready = True
```

Kodni isječak 6.4 Funkcija `update()` unutar datoteke `fsm.py`

IDLE

Ukoliko je osoba detektirana u *frame-u*, status na korisničkom sučelju se ažurira te se poziva korisnika da priđe bliže kako bi ga se prepoznalo, a pomoću funkcije `beep()` emitira se zvučni signal. Varijabla `self.body_in_frame` koja je u funkciji privremenog *flag-a*, označava da li je osoba detektirana u kadru postavlja se na vrijednost `True`. Ukoliko je detektirano lice (a osoba je već detektirana u kadru), status na korisničkom sučelju se ažurira kako bi obavijestio korisnika da sustav sada pokušava prepoznati lice. Varijabla `self.body_in_frame` sada se postavlja na početnu vrijednost `False`. Nakon toga trenutno stanje sustava prelazi u `FACE_DETECTED` odnosno u stanje u koje se prelazi nakon što je lice detektirano.

FACE_DETECTED

Sustav u stanju `FACE_DETECTED` provjerava postoje li prethodno prepoznata lica iz glavne dretve te ukoliko postoje pronalazi lice sa najvećim dimenzijama odnosno najbliže lice. Zatim provjerava da li je *timer* dretva aktivna te ukoliko nije, inicijalizira se nova *timer* dretva koja na dvije sekunde odgađa prijelazak u sljedeće stanje, a to je stanje `FACE_CAPTURE` te unutar `name` varijable sprema ime prepoznatog korisnika. Ukoliko lice izađe iz *frame-a*, *timer* dretva se zaustavlja i stanje se ne mijenja u `FACE_CAPTURE` nego sustav prelazi ponovno u stanje `IDLE`. Ukoliko je prepoznati korisnik "*Unknown*" odnosno nepoznat, u grafičkom sučelju pojavljuje se gumb "*I'm new!*" te se prethodna *time*r dretva zaustavlja i ne prelazi se u novo stanje dok korisnik ne pritisne gumb za nastavak.

FACE_CAPTURE

U stanju `FACE_CAPTURE`, ažurira se statusna poruka u grafičkom sučelju koja poziva korisnika na upis imena.

Ukoliko je prepoznato lice nepoznato *recognizeru*, varijabla `current_name` pokreće funkciju `get_name_from_user Vision` klase i dohvaća ime korisnika. Zatim sustav snima različite fotografije nepoznatog lica. Svaka fotografija prikupljena iz

različitih perspektiva služi kao zapis specifičnih obilježja lica, bilo da se radi o osvjetljenju, izrazu ili kutu snimanja.

Varijabla `capture_count` služi kao brojač koji prati broj snimljenih slika dok ne dođe do određenog broja. Kroz ovaj proces, sustav sprema slike u odgovarajući direktorij (`faces/name`) za daljnju obradu. Sustav tada prebacuje stanje u `TRAINING`.

U slučaju da je prepoznato lice već poznato *recognizeru*, stanje sustava prebacuje se u stanje `GREET`.

TRAINING

Unutar stanja `TRAINING`, sustav prvo pokreće obradu novozabilježenih slika funkcijom `process_faces` `Vision` klase. Korisničko sučelje se ažurira s porukom o statusu, obavještavajući korisnika da trenutačno trenira *recognizer*. Sustav obrađuje zabilježena lica i prosljeđuje ih u funkciju `train_model` `Vision` klase. Nakon što je model istreniran, model se sprema za buduću upotrebu pod nazivom `model.yml`. S fazom treniranja modela završenom, sustav prelazi na stanje `GREET`, spreman za interakciju s novoprepoznatom osobom.

GREET

U stanju `GREET` sustav ažurira statusnu poruku funkcijom `update_status` `SimpleUI` klase te upućuje personalizirani pozdrav korisniku dohvaćajući korisničko ime funkcijom `get_greeting` koja se nalazi unutar `helpers.py` datoteke. Također se prikazuje gumb pomoću `update_button` funkcije `SimpleUI` klase te je korisnička interakcija potrebna za nastavak u sljedeće stanje `FUN_FACT`.

FUN_FACT

U stanju `FUN_FACT` sustav ažurira statusnu poruku prikazom zabavne činjenice o korisničkom imenu. Zabavnu činjenicu dohvaća funkcijom `get_fun_fact` koja se nalazi unutar `helpers.py` datoteke. Na kraju se prikazuje gumb i korisničkom interakcijom prelazi se u sljedeće stanje, a to je ponovno `IDLE`. Varijabla `current_name` koja

označava ime korsinika prikupljeno u trenutnoj iteraciji programa te ju postavlja na `None` kako bi se mogla ponovno koristiti u sljedećoj iteraciji.

6.6 vision.py

Unutar `vision.py` datoteke nalazi se klasa `Vision` unutar koje su definirane funkcije koje obavljaju zadatke vezane uz operacije kamere i zadatke obrade slika.

6.6.1 Funkcija `capture_frames()`

```
def capture_frames(self):
        local_frame_count = 0
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, self.
   RESOLUTION_WIDTH)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, self.
   RESOLUTION_HEIGHT)
        try:
            while self.running:
                ret, frame = self.cap.read()
                if local_frame_count % self.SKIP_FRAMES == 0:
                    with self.lock:
                        self.ret, self.frame = ret, frame
                    self.new_frame_event.set()

                local_frame_count += 1
                sleep(0.01)
        finally:
            self.cap.release()
```

Kodni isječak 6.5 Funkcija `capture_frames()` unutar datoteke `vision.py`

Funkcija `capture_frames()` kontinuirano snima *frame-ove* sa kamere u rezoluciji 320 x 240, sve dok se *flag* `running` ne postavi na vrijednost `False`. U varijable `ret` i `frame` spremamo vrijednosti koje vraća funkcija `cap.read()`. `ret` je boolean

vrijednost koja označava da li je *frame* uspješno učitan, dok varijabla `frame` predstavlja sljedeći *frame* koji je spreman za obradu. Kako sustav ne bi obrađivao svaki pojedinačni *frame*, svaki treći *frame* odnosno broj *frame-ova* definiran konstantom `SKIP_FRAMES` se pohranjuje za daljnju obradu. Varijabla `local_frame_count` prati koliko je *frame-ova* prošlo kako bi se mogla koristiti za izračunavanje svakon n-tog *frame-a*. S obzirom da se funckija `capture_frames` pokreće unutar dretve, dobra je praksa definirati sigurnost dretvenosti koja se postiže korištenjem lock objekta. Na taj način sprječavaju se situacije gdje dvije dretve istovremeno pristupaju ili mijenjaju dijeljene varijable. Posljednji snimljeni frame sprema se u `self.frame` i postavlja se događaj koji signalizira da je novi *frame* spreman za glavnu dretvu (`main` funkcija). Na kraju funkcije varijabla `local_frame_count` se inkrementira i koristi se `sleep()` funkcija sa kratkim trajanjem radi dodatne stabilnosti sustava. Funkcijom `start_capture` pokreće se funkcija `capture_frames` unutar nove dretve, te se dretva zaustavlja funkcijom `stop_capture` pri izlasku iz programa. Također unutar funkcije `stop_capture` prekida se dohvaćanje *frame-ova* sa kamere i briše se dretva. Na kraju `try` bloka korsiti se ključna riječ `finally` u slučaju da dođe do prekida programa te otpušta kameru.

6.6.2 Funkcija `capture_faces()`

```
1    def capture_faces(self, closest_face, name, frame, i):
2        if not os.path.exists('faces/' + name):
3            os.makedirs('faces/' + name)
4
5        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
6        print("Capturing face " + str(i) + " for " + name)
7
8        x, y, w, h, _ = closest_face
9        print(cv2.imwrite(f"faces/" + name + "/" + str(i) + ".
   jpg", gray[y:y+h, x:x+w]))
10       cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0),
   2)
```

Kodni isječak 6.6 Funkcija `capture_faces()` unutar datoteke `vision.py`

Funkcija `capture_face()` snima slike iz *frame-a* i sprema ih u direktorij nazvan imenom prepoznate osobe. Započinje provjerom postoji li već direktorij za određeno ime te ukoliko ne postoji kreira ga. Zatim pretvara frame u *grayscale* format potreban za OpenCV obradu. Funkcija dohvaća položaj i veličinu najbližeg lica u *frame-u* (parametar `closest_face`) te sprema dio *grayscale frame-a* koji sadrži lice kao JPG datoteku unutar pripadajućeg direktorija. Na kraju u izvornom *frame-u* se ocrtava lice plavim kvadratom kako bi se označilo pronađeno lice.

6.6.3 Funkcija `process_faces()`

```
1     def process_faces(self, faces_dir="faces"):
2         images, labels, name_label_mapping, id = [], [], {}, 0
3         for subdir, dirs, files in os.walk(faces_dir):
4             for file in files:
5                 filepath = subdir + os.sep + file
6                 name = os.path.basename(subdir)
7                 if name not in name_label_mapping:
8                     name_label_mapping[name] = id
9                     id += 1
10                label = name_label_mapping[name]
11                images.append(cv2.imread(filepath, cv2.
    IMREAD_GRAYSCALE))
12                labels.append(int(label))
13
14        self.name_label_mapping = name_label_mapping
15
16        return images, labels
```

Kodni isječak 6.7 Funkcija `process_faces()` unutar datoteke `vision.py`

Funkcija `process_faces()` dizajnirana je za dohvaćanje i obradu slika lica iz određenog direktorija. Osnovna svrha je da se svaka slika poveže sa odgovarajućom oznakom baziranom na imenu poddirektorija iz kojeg slika dolazi. Kako bi to postigla, funkcija prolazi kroz sve poddirektorije navedenog direktorija, čita svaku sliku koristeći OpenCV biblioteku i mapira je na odgovarajuću oznaku. Na kraju, funkcija vraća dve liste: jednu sa svim pročitanim slikama i drugu sa pripadajućim oznakama.

Oznaka odnosno varijabla `label` predstavlja svojevrsni ID spremljenih lica koja će biti upamćena. Liste se zatim koriste za treniranje modela pomoću `train_model` funkcije.

6.6.4 Funkcija `train_model()`

```
1     def train_model(self, images, labels):
2         self.recognizer.train(images, np.array(labels))
```

Kodni isječak 6.8 Funkcija `train_model()` unutar datoteke `vision.py`

Funkcija `train_model()` poziva OpenCV `train` funkciju za *recognizera* klase `Vision` sa parametrima `images` i `labels` dobivenih iz `process_faces()` funkcije.

6.6.5 Funkcija `detect_faces()`

```
1    def detect_faces(self, frame):
2         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
3         faces = self.face_cascade.detectMultiScale(gray,
    scaleFactor=1.05, minNeighbors=5)
4         return faces
```

Kodni isječak 6.9 Funkcija `detect_faces()` unutar datoteke `vision.py`

Funkcija `detect_faces()` prima *frame* kao parametar zatim ga pretvara u *grayscale* format potreban za OpenCV obradu. Nadalje, `detectMultiScale()` funkcija učitanog OpenCV kaskadnog klasifikatora za prepoznavanje lica, dodjeljuje `faces` varijabli listu *tuples-a* koji sadržavaju koordinate i dimenzije detektiranih lica. Funkcija `detect_bodies()` koja detektira osobe u *frame-u* funkcionira na identičan način, razlikujući se u kaskadnom klasifikatoru kojeg koristi.

6.6.6 Funkcija `recognize_faces()`

```
1 def recognize_faces(self, frame, faces):
2         recognized_faces = []
3         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
4
5         for (x, y, w, h) in faces:
```

```
6             face_roi = gray[y:y+h, x:x+w]
7             label, confidence = self.recognizer.predict(
  face_roi)
8             if confidence < 110:
9                 detected_name = self.get_name_from_label(label)
10            else:
11                detected_name = "Unknown"
12            recognized_faces.append((x, y, w, h, detected_name)
  )
13
14        return recognized_faces
```

Kodni isječak 6.10 Funkcija `recognize_faces()` unutar datoteke `vision.py`

Funkcija `recognize_faces` koja sadrži funkcionalnost prepoznavanja lica pomoću OpenCV LBPH (engl. *Local Binary Pattern Histogram*) *recognizera*. Funkcija kao parametre prima `frame` i `faces` (lista *tuples-a*) te na početku deklarira praznu listu naziva `recognized_faces` koja će sadržavati koordinate, dimenzije i imena prepoznatih lica. Nadalje funkcija pretvara *frame* u *grayscale* za daljnju OpenCV obradu te petljom prolazi kroz koordinate i dimenzije svih lica spremljenih u `faces` varijabli. Deklaria se varijabla `face_roi` koja specificira područje interesa u *frame-u*. Zatim *recognizer* pomoću LBPH algoritma predviđa oznaku (`label`) pronađenog lica. Osim oznake vraća i varijablu `confidence` koja označava koliko je *recognizer* siguran u svoju predikciju. Manje vrijednosti označuju veću sigurnost. Zatim, ukoliko je `confidence` varijabla manja od 110 poziva funkciju `Vision` klase `get_name_from_label` koja mapira oznaku prepoznatog lica sa povezanim imenom iz `mapping.json` datoteke. Ukoliko je detektirano nepoznato lice, ime se postavlja kao "*Unknown*"). Na kraju funkcije u `recognized_faces` listu dodaju se koordinate, dimenzije i ime prepoznatog lica. Lista se zatim vraća.

7 Zaključak

Glavni cilj ovog završnog rada bila je realizacija interaktivnog sustava sposobnog za detekciju i prepoznavanje. Središnji element sustava su tehnike računalnog vida implementirane kroz OpenCV biblioteku. Sustav omogućuje personaliziranu interakciju korisnicima koje poznaje, a novim korisnicima omogućuje upoznavanje.

Sustav implementira niz značajnih funkcionalnosti, uključujući strukturiranu kontrolu nad različitim stanjima u kojima se može nalaziti, koristeći automat konačnih stanja, kao i paralelno upravljanje logikom obrade i prikazom videozapisa u stvarnom vremenu putem višedretvenog pristupa.

S obzirom da je sustav zasnovan na ugradbenoj arhitekturi, to mu daje svestranost potrebnu za različite stvarne primjene, bilo kao neovisna jedinica ili kao element nekog drugog sustava.

Potencijal za buduće unaprjeđenje sustava je prisutan, posebno u području personalizirane i intuitivne interakcije korisnika sa sustavom.

Popis slika

Popis kodnih isječaka

Literatura

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *International Journal of Computer Vision*, 2001.

[2] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," 1996.

[3] Face recognition with opencv. , s Interneta, https://docs.opencv.org/3.4/da/d60/tutorial_face_main.html , rujan 2023.

[4] Raspberry pi 3 model b. , s Interneta, https://www.raspberrypi.com/products/raspberry-pi-3-model-b/ , rujan 2023.

[5] Raspberry pi 3 model b. , s Interneta, https://images.prismic.io/rpf-products/877fb653-7b43-4931-9cee-977a22571f65_3b+Angle+2+refresh.jpg?auto=compress%2Cformat&fit=max&w=600&h=400&fm=webp , rujan 2023.

[6] Raspberry pi official universal power supply. , s Interneta, https://cdn1.smartmedia.is/computer/skrar/nyjar-vorumyndir/8721-1.jpg , rujan 2023.

[7] Ninja coupé pibow raspberry pi kućište. , s Interneta, https://shop.pimoroni.com/cdn/shop/products/Variant_4_of_4.JPG?v=1526041002&width=1024 , rujan 2023.

[8] Raspberry pi modul kamere. , s Interneta, https://ram-e-shop.com/wp-content/uploads/2019/10/rpi_camera_5mp.jpg , rujan 2023.

[9] Raspberry pi 7" dodirni zaslon. , s Interneta, https://images.prismic.io/rpf-products/432cfd901a27d4bbbddb761269547cedf4aa4c1b_display-kit-1-1510x1080.jpg?auto=compress%2Cformat&fit=max&w=600&h=400&fm=webp , rujan 2023.

[10] Pimoroni - royale 7" okvir za dodirni zaslon. , s Interneta, https://i.ebayimg.com/images/g/20MAAOSwpRRWpmy6/s-l1200.webp , rujan 2023.

[11] Piezo zvučnik. , s Interneta, https://m.media-amazon.com/images/I/51-zSo0wB8L._AC_SL1080_.jpg , rujan 2023.

[12] Raspberry pi os. , s Interneta, https://www.raspberrypi.com/software/ , rujan 2023.

[13] Opencv. , s Interneta, https://opencv.org/ , rujan 2023.

[14] Pyqt5. , s Interneta, https://doc.qt.io/qtforpython-6/index.html , rujan 2023.

Pojmovnik

API engl. *Application Programming Interface*. 16, 21

GPIO engl. *General-purpose input/output*. 10, 21

HDMI engl. *High-Definition Multimedia Interface*. 10

LBPH engl. *Local Binary Pattern Histogram*. 34

OpenCV engl. *Open Source Computer Vision Library*. 1, 15, 16, 20, 32–35, 41

RAM engl. *Random Access Memory*. 10

USB engl. *Universal Serial Bus*. 10

Sažetak

U ovome završnom radu predstavljen je sustav implementiran na ugradbenoj arhitekturi Raspberry Pi 3 - Model B, koji koristi funkcionalnosti OpenCV (engl. *Open Source Computer Vision Library*) biblioteke za detekciju i prepoznavanje lica. Korištenjem automata konačnih stanja postignuta je strukturirana kontrola nad različitim funkcijama i interakcijama unutar aplikacije. Zahvaljujući višedretvenom pristupu, aplikacija može paralelno upravljati logikom obrade i prikazivanjem video zapisa u stvarnom vremenu. Korisničko sučelje koje omogućuje korisnicima personaliziranu interakciju sa sustavom implementirano je kroz PyQt5 biblioteku.

Ključne riječi **— detekcija, prepoznavanje, automat konačnih stanja, višedretvenost, OpenCV, PyQt5**

Abstract

In this final paper, a system is introduced that is implemented on the embedded architecture Raspberry Pi 3 - Model B and utilizes the functionalities of the OpenCV library for face detection and recognition. Using a finite state machine, structured control over various functions and interactions within the application has been achieved. Thanks to the multithreaded approach, the application can concurrently manage processing logic and display UI elements, such as the video feed, in real-time. The user interface, which allows users personalized interaction with the system, is implemented through the PyQt5 library.

Keywords **— detection, recognition, finite state machine, multithreading, OpenCV, PyQt5**