TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Virumaa kolledž

Rakenduse loomine vanuse, soo ja emotsioonide tuvastamiseks reaalajas tehtud näopiltide põhjal

Creating an application for age, gender and emotion detection from real-time facial images

Arukad süsteemid ja rakendusinfotehnoloogia ÕPPEKAVA LÕPUTÖÖ

Üliõpilane: Anton Štunder
Üliõpilaskood: 193049EDTR
Juhendaja: Natalja Ivleva, lektor

Kohtla-Järve, 2024

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.

Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

LIHTLITSENTS LÕPUTÖÖ ÜLDSUSELE KÄTTESAADAVAKS TEGEMISEKS JA REPRODUTSEERIMISEKS

Mina Anton Štunder (sünnikuupäev: 28.01.1996)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

Rakenduse loomine vanuse, soo ja emotsioonide tuvastamiseks reaalajas tehtud näopiltide põhjal, mille juhendaja on Natalja Ivleva,

1.1. reprodutseerimiseks säilitamise ja elektroonilise avaldamise eesmärgil, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta kolmandate isikute intellektuaalomandi ega isikuandmete kaitse seadusest ja teistest õigusaktidest tulenevaid õigusi.

SISUKORD

SISSEJUHATUS

Informatsioonitehnoloogia areng on avanud meile palju uusi võimalusi. Üheks neist on arvutinägemine. Kaasaegses maailmas suudavad arvutid, mis on õpetatud keerukate algoritmide abil suure hulga andmete põhjal, tuvastada, kes või mis on pildil. Samuti suudavad nad tuvastada inimese vanuse, soo ja emotsiooni tema näo põhjal. Just see saab olema selle projekti peamine teema.

Luuakse rakendus puutetundliku ekraani ja kaameraga arvuti jaoks. See rakendus suudab tuvastada inimese isiku tema näo järgi ja teda tervitada. Kasutaja saab end rakendusse lihtsalt ja kiiresti lisada. Samuti saab ta rakenduselt paluda tuvastada tema sugu, vanus või emotsioon tema näo järgi. Kõik need võimalused on kättesaadavad mugava ja intuitiivse graafilise kasutajaliidese abil. Rakendus suhtleb nii teksti kui ka illustratsioonide abil. Samuti toetatakse mitmekeelsust, nimelt eesti, vene ja inglise keelt.

Projekti eesmärk on luua huvitav interaktiivne rakendus. Antud projekt võib suurendada huvi selle eriala vastu ning samuti pakkuda kasutajale lõbu. Lisaks tõstab rakenduse loomise protsess autori oskuste taset ja annab kasulikku praktilist kogemust.

Projekti peamiseks uurimisobjektiks on arvutinägemine, eriti näotuvastus. Selleks on välja töötatud mitmeid teeke. Kõige tuntum neist on OpenCV. Kuid selles projektis kasutatakse spetsiifilise ülesande jaoks sobivamat teeki nimega DeepFace. Sellel on arusaadav kasutajaliides ja see on üsna kasutajasõbralik. Teek sisaldab kogu vajalikku funktsionaalsust arvutinägemise rakenduses realiseerimiseks. See teek on kirjutatud Pythoni programmeerimiskeele jaoks.

Teine suur osa projektist on graafilise kasutajaliidese (GUI) loomine. Kuna DeepFace teek on kirjutatud Pythoni jaoks, oleks mõistlik kogu projekti arendada Pythonis. GUI loomiseks on mitmeid teekisid, kuid autori valik langes Tkinterile. See on kõige populaarsem teek GUI loomiseks Pythonis. See töötab kõigil platvormidel, on kergekaaluline ja hästi dokumenteeritud. Lisaks sellele, kuna see on väga populaarne, on sellele internetis saadaval palju abistavat infot ja näiteid.

Võtmesõnad: näotuvastus, rakendus, DeepFace, Tkinter, bakalaureusetöö

1. ARVUTI NÄGEMINE JA NÄGUDE TUVASTUS

1.1 Arvuti nägemine

Arvutinägemine on tehisintelekti (AI) valdkond mis lubab arvutitele ja süsteemidele mõista mis on pildil või video peal kujundatud ja selle arusaamisega siis midagi teostada või mingi info kasutajale anda. Üldiselt AI annab arvutile võimalust mõelda ja arvutinägemine annab talle võimalust näha.

Arvutinägemine töötab suhtelist sarnaselt inimesta nägemisega, kuid inimestel on edumaa arvutite ees. Inimeste nägemisel on eelis, sest ta saab terve elu selleks et õppida kuidas objektid erinevad, kui kaugel nad asuvad, kas nad liikuvad ja aru saada kui pildiga on midagi valesti.

Arvutinägemine õpetab masinaid neid funktsioone teostada, kuid ta peab tegema seda palju lühema aja jooksul. Ta saavutab seda kaamerate, andmete ja algoritmide abil, kuid inimesel on selle asemel silmad, optiilised närvirakud ja visuaalne ajukoor. Kuna süsteem, mis on treenitud selleks, et inspekteerida tooteid või jälgida protsesse võib analiseerida tuhandeid asju või protsesse minutis, märkades raskesti nähtavaid defekte või probleeme, siis arvutid saavad kiiresti ületada inimeste nägemist erinevates valdkondades.

Arvutinägemine on kasutusel paljudes erinevates valdkondades, aslustades energeetikast ja tööstustes ja lõpetades nutitelefonidega ja autodega – ja arvutinägemise turg jätkab kasvu. 2023 aastal see turg on väärt ca 17.2 miljardit dollarit.

Mõned arvutinägemise kasutusalad on:

- Isejuhtivad autod
- Kaameratega varustatud auto saab ümbrusest pilte teha ja arvutinägemise algoritmile saada, mis analiseeriks neid pilte. Sellisel viisil autonoomsed autot võivad saada infot ümbrusest, nagu teede piiridest, liiklusmärkidest, jalakäijatest ja teistest autodest. Seda infot kasutades auto saab iseseisvalt liikuda.
- Liitreaalsus
- Liitreaalsus samuti kasutab arvutinägemist. Ta lubab seadmetele, näiteks nutitelefonidele, lisada erinevaid objekte kaamera pildile. Selleks, et seda korrektselt teha, arvutinägemine määrab pinnad, nagu seinad, lauad, põrand ja muu.
- Tervishoid

- Arvutinägemisel on ka tevishoiule abiks. Ta saab vaadata probleeme naha peal ja analiseerida keha skaneerimise tulemust kiiresti ja täpselt.
- Nägude tuvastamine
- Nägude tuvastamist kasutatavad rakendused tuginevad kõvasti arvutinägemise algoritmidele. Need algoritmid saavad identifitseerida erinevate inimeste näojooned ja nende abil aru saada kelle näo on pildi peal. Nägude tuvastamise rakendused on kasutusel nutitelefonidel, et lubada seda kasutamiseks õigele inimesele. Sotsiaalvõrkudes, et inimest nö foto peal tagida. Samuti seda kasutatakse selleks, et kriminaali tuvastada turvakamerate video pealt.

Peamised ülesanned, mis saab arvutinägemisega teostada:

- Pildi klassifitseerimine
- Pilt pannakse mingisse kategooriasse. Näiteks algoritm saab järeldada, et pildil on auto või koer. See sobib pildile, kus on üks peamine objekt.
- Objektide tuvastamine
- Pildi peal tuvastatakse erinevaid objekte. Kus nad asuvad ja mis nad ennast kujutavad. Näiteks selline ülesanne või tekitada tootmisliini peal, kus on vaja tuvastada ja eemaldada defektseid tooteid.
- Objekti jälgimine
- Video pealt jälgitakse mingi objekti liikumist. Selle näideks võib olla auto parkimisala, või mingisugune loom loomaaias.
- Sümbolite äratundmine
- Arvutinägemine saab tuvastada teksti pildi pealt. [1]

Selleks, et arvuti saaks aru mis on pildi peal, teda peab õpetama. Selle jaoks on loodud keerulised masinõpe algoritmid, mis on erinevate arvutinägemise ülesannete jaoks sobilikud. Samuti on vaja palju pilte, mis on inimeste poolt tuvastatud. Näiteks pildi klassifitseerimise jaoks, iga pildi jaoks on üks klass. Ehk kõik kassiga pildid on klassiga kass, koeraga koer. Objekti tuvastamise jaoks, inimene peab märkima kus ja mis asub. Siis need andmed on algoritmile pakutud ja ta kasutab suurema osa õppimiseks ja väiksema osa õpingu tulemuse kontrolliks. Õpingu tulemusena on loodud mudel, mis nüüd saab mingi arvutinägemise ülesanne lahendada.

1.2 Nägude leidmine ja tuvastus

Nägude leidmine on arvutinägemise ülesanne, mis kasutab tehisintelekti selleks, et leida näod pildi pealt. Kui näod on leitud, siis nendega saab veel palju asju teha. Kindlasti on

võimalik tuvastada, kellele need näod kuluvad. Kaamerad võivad kasutada seda, et fokuseerida peamise tegelase näole. Turvakaamerad võivad märkada, kui keegi tuleb sisse või lugeda kui plaju inimesi on tulnud või lahkunud hoonest. Võib ka määrata nende nägude emotsiooni, vanust, sugu või rassi näiteks.

Näo leidmine on objekti tuvastamise liige, sest on vaja eristada mingi objekti (näo) teistest objektidest pildi peal. Selleks on vaja trennida masinaõpe mudeli tohutu hulga määratud nägudega pilte kasutades. Sellised mudelid on juba loodud ja on pakutud arendajatele kasutamiseks. Nendel on erinevad plussid ja miinused: kiirus, täpsus, väljundite kogus. Mõned ei suuda leida näo, kui ta on nurga all. Mõned annavad teada nii, kus näo asub, kui ka silmade ja suu asukoha, ja lisaks sellele mille nurga all see nägu on. [2]

Näotuvastus on teine oluline arvutinägemise ülesanne. Seda kasutatakse näiteks selleks, et avada telefoni või kriminaali identifitseerida turvakamerate video pealt. See töötab nii, et on olemas identifitseeritava isiku pilt, siis algoritm skaneerib seda pilti, et luua isiku näo „allkirja". Kui on vaja inimese mingi teise pildi pealt identifitseerida, algoritm võrdleb näo „allkirja" pildist andmebaasis olevatega ja annab välja tulemuse. [3]

Selleks, et tuvastada näo vanust, emotsiooni või sugu on järelikult mudelid loodud. Üldiselt tuvastatavad emotsioonid on: õnne, kurbus, viha, hirm, üllatus, jälkustunne ja lisaks neutraalne emotsioon. Nende omaduste tuvastamise täpsus sõltub näo kaldenurgas, valgustusest jne.

1.3 Arvutinägemise teegid

Selles peatükis on välja toodud mõned näided teekidest, mis saab kasutada arvutinägemise ülesande lahendamiseks. Need teegid on avatud kasutamiseks ja on hästi dokumenteeritud.

1.3.1 ML Kit

Google poolt loodud. See teek võimaldab lahendada nii arvutinägemise, kui ka keeltega seotud ülesandeid. Tal on nägude leidmise funktsioon olemas, kuid puudub näotuvastus ja emotsiooni/vanuse/sugu määramine. Samuti ta on loodud nutitelefonide jaoks. Nendel põhjustel ei sobi ta projekti arendamiseks. [4]

1.3.2 OpenCV

Võimalikult kõige populaarsem arvutinägemise teek. Ta pakkub palju funktsioone, mis

aitavad arvutinägemise ülesandeid lahendada. Need funktsioonid pole ainult otseselt arvutinägemisega seotud. Ta võimaldab pilte töödelda, veebikaameraga töötada ja muu. Teda saab kasutada kõikides populaarsetes platvormides ja erinevates keeltes. Kahjuks tal pole head lahendust näotuvastamise ja emotsiooni/vanuse/sugu määramise jaoks. Kuid ta võimaldab autorile veebikaameraga töötada ja selleks autor kasutabki seda. [5]

1.3.3 Deepface

Arvutinägemise teek, mis spetsialiseerub nägudele. See omab funktsioone nii nägude leidmiseks, tuvastamiseks, kui ka näo emotsiooni/vanuse/sugu määramiseks. See on loodud pythoni keele jaoks ja tal on mugav kasutajaliides. See teek sobib projekti realiseerimiseks kõige paremini. Tagataustas ta kasutab erinevad mudelid nägude leidmiseks/tuvastamiseks ja annab arendajale võimalust neid realiseerida. Kuna nägude leidmisel on oluline rool projektis, siis oleks mõistlik need mudelid testida ja võrrelda. Siis saab valida sellist, mis töötaks kiiresti ja hea täpsusega. Mudeli saab valida nii:

```
DeepFace.extract_faces(img_path = "img.jpg",
detector_backend = "opencv"/"ssd"/...)
```

[6]

1.4 Uuring

Selleks, et võrrelda erinevad nägude leidmise mudelid, on loodud pythoni skripti. Internetist on leitud 10 fotot inimestega ja nende failide nimi sees on lisatud nägude number. Siis saab võrrelda kui palju näod oli leitud iga mudeli poolt (Tabel 1.1) ja kui palju aega möödus (Tabel 1.2). Samuti on loodud uued pildid, kus oleks näha kus need näod olid leitud. Nende tulemuste abil saab valida mudeli, millega oleks mõistlikum näo leidmise realiseerida. Pildid on võetud Google otsingust „people" päringuga ja piltide nimed on koostatud järgmisel viisil: pildi number_nägude arv.

Tabel 1.1 mudelite poolt leitud näode arv

Pilt	Näod	Opencv	Ssd	Mtcnn	Retinaface	mediapipe
1_1.jpg	1	1	1	1	1	1
2_7.jpg	7	4	5	7	7	1
3_8.jpg	8	5	6	8	8	5
4_15.jpg	15	8	7	15	15	0
5_2.jpg	2	1	0	1	2	1
6_8.jpg	8	7	0	9	8	0
7_9.jpg	9	9	9	9	9	0
8_3.jpg	3	2	2	3	3	3
9_5.jpg	5	0	1	4	5	1
10_3.jpg	3	1	0	3	3	0

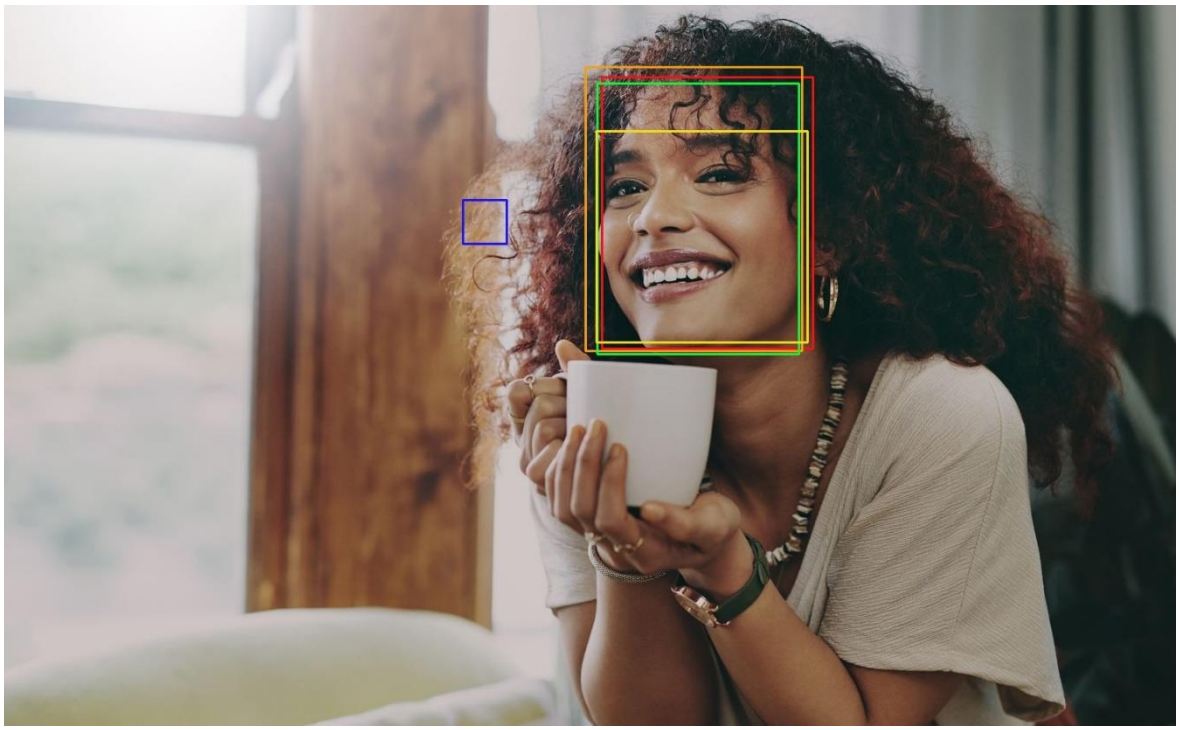
Tabel 1.2 täitmisele kasutatud aeg sekundited

Mudel	Aeg sekundited
Opencv	1,52
Ssd	0,62
Mtcnn	11,98
Retinaface	19,68
Mediapipe	0,05

Mtcnn ja Retinaface on kõige suurema täpsusega. Retinaface leidis absoluutselt kõik näod piltidest. Kuid nad on liiga aeglased minu ülesanne jaoks. Meil jäävad siis OpenCV, Ssd ja Mediapipe. Mediapipe on väga kiire, kuid tema täpsus on kehv. Selleks, et valida OpenCV ja Ssd vahel, autor vaatab tulemused pildi pealt. OpenCV on sinise ja Ssd punase värviga. Esimene pilt, mis pakkub huvi on 6_8.jpg (Joonis 1.1), kus OpenCV leidis 7, kuid Ssd 0 näod.

Joonis 1.1 Pilt 8 nägudega [9]

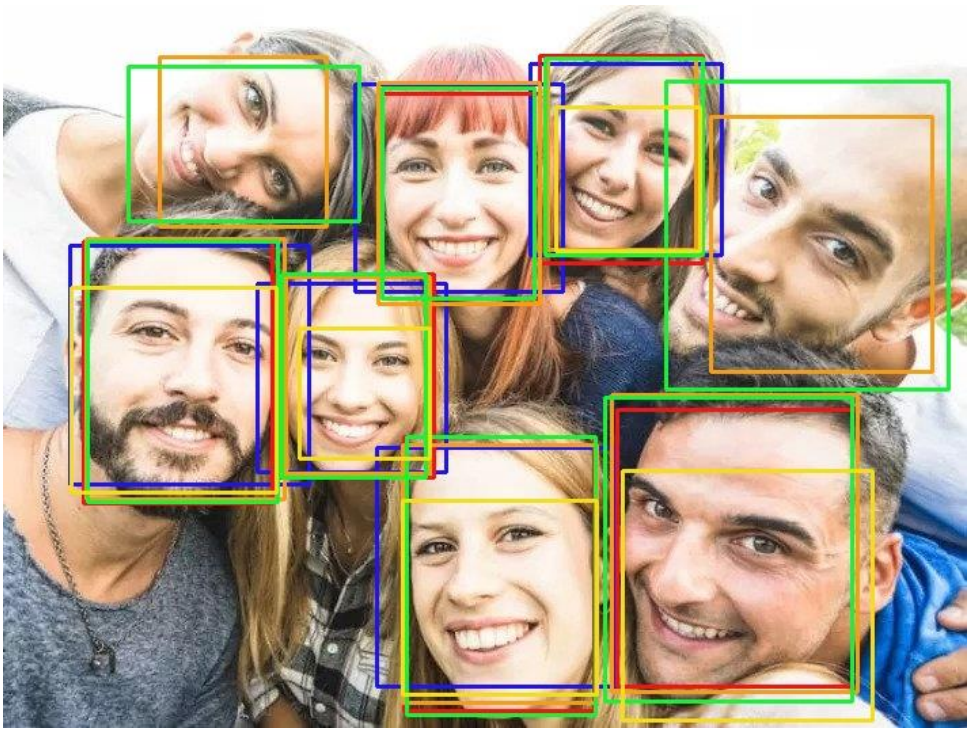
Nagu on näha see pilt ei sarnane sellega, mis veebikaamera hakkab nägema. Vaatan edasi 1_1.jpg (Joonis 1.2) pilti.



Joonis 1.2 Pilt 1 näoga [10]

Siis OpenCV näitas ennast halvemalt poolt. Selline pilt on sarnasem sellega, mis hakkab veebikaamera ees tulema. Lõpuks vaatan pildid 2_7.jpg (Joonis 1.3) ja 3_8.jpg (Joonis 1.4), sest nad on ka sobilikumad.

Joonis 1.3 Pilt 7 nägudega [11]



Joonis 1.4 Pilt 8 nägudega [12]

Ssd töötas nende peal natuke paremini ja võtis vähema aega. Sellepärast projektis

nägude leidmiseks hakatakse kasutama Ssd standartse OpenCV mudeli asemel.

1.5 Kokkuvõte

Olid vaadeldud erinevad arvutinägemise teegid ja otsustus langes DeepFace teegile. Samuti olid võrreldud erinevad mudelid nägude leidmiseks, mis on DeepFace poolt pakutud ja oli kehtestatud et Ssd sobis kõige paremini. Programmeerimiskeeleks on Python. Kasutajaliidese loomiseks on Tkinter teek. Tkinter on populaarne ja kergekaaluline. Ta on hästi dokumenteeritud ja kuna ta on populaarne, siis internetis leidub palju õpimismaterjaali ja näideid.

2. RAKENDUSE ÜLEVAADE

2.1 Teegid

- Deepface

Selle teegi abil on realiseeritud kõik arvutinägemise funktsioonid rakenduses, nimelt: nägude leidmine, nägude tuvastus, sugu/vanuse/emotsiooni määramine.

- OpenCV

OpenCV on kasutatud veebikaamerast piltide lugemiseks, piltide salvestamiseks ja manipuleerimiseks.

- Tkinter

Tkinteri abil on realiseeritud kasutajaliides. Tkiteris on vanamoodsed nupud ja sellepärast koos temaga on kasutusel ka CustomTkinter, mis on varastatud parema välimusega nuppudega.

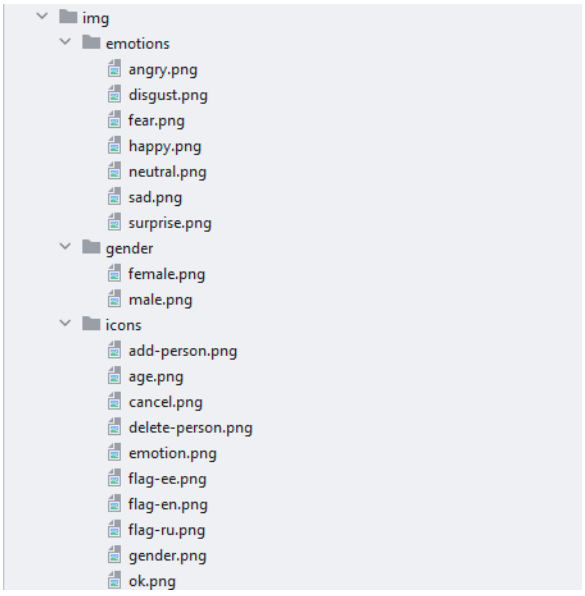
- Pillow

See on peamine piltidega töötamise teek. Tema laeb pilte ja joonistab veebikaamera pildi peal. Samuti Pillowi abil on genereeritud teksti, näo ovaali ja progressiriba pildid. Sellega autor manipuleerib ka piltide suurust ja läbipaistavust.

2.2 Failid

2.2.1 Ressursid

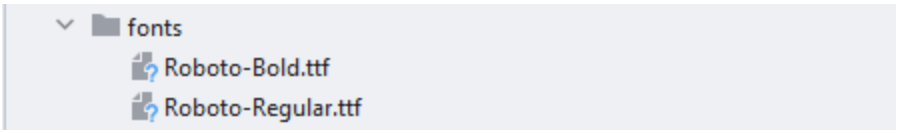
1) Pildid



Joonis 2.1 Piltide failid

Pildid on leitud Google otsingu abil ja vajadusel on tehtud muudatused GIMP tarkvaraga.

2) Trükkiri



Joonis 2.2 Trükkiride failid

Võetud Google fontist ja on kasutusel teksti piltide genereerimiseks.

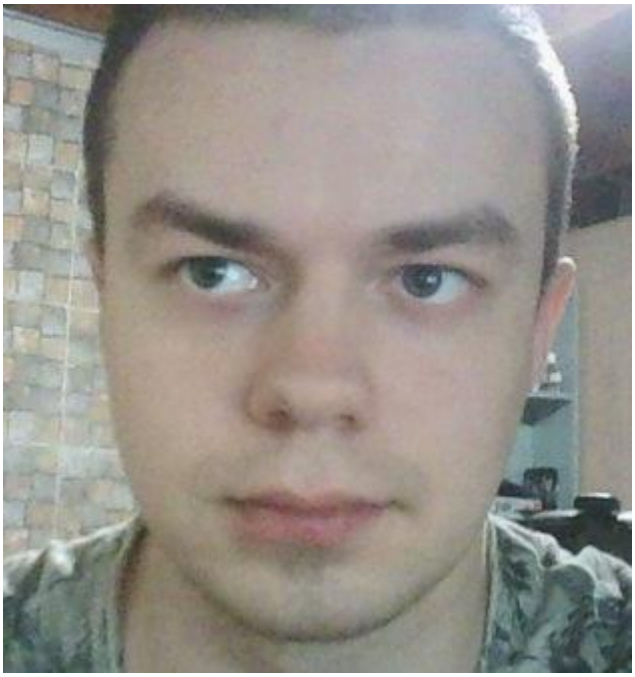
3) text.csv

Sisaldab rakenduse teksti kolmes keeles.

```
hello;hello;tere;привет
good_morning;good morning;tere hommikust;доброе утро
good_afternoon;good afternoon;tere päevast;добрый день
good_evening;good evening;tere õhtust;добрый вечер
angry;you are angry;te olete kuri;вы злы
```

Joonis 2.3 text.csv

4) Nägude pildid



Joonis 2.4 Näo pilt

Rakenduse poolt tehtud nägude pildid selleks, et hiljem aru saada kes on veebikaamera ees. Nimi on {id}.jpg.

5) names_db.csv

```
24,Leonardo Dicaprio
25,Антон Штундер
```

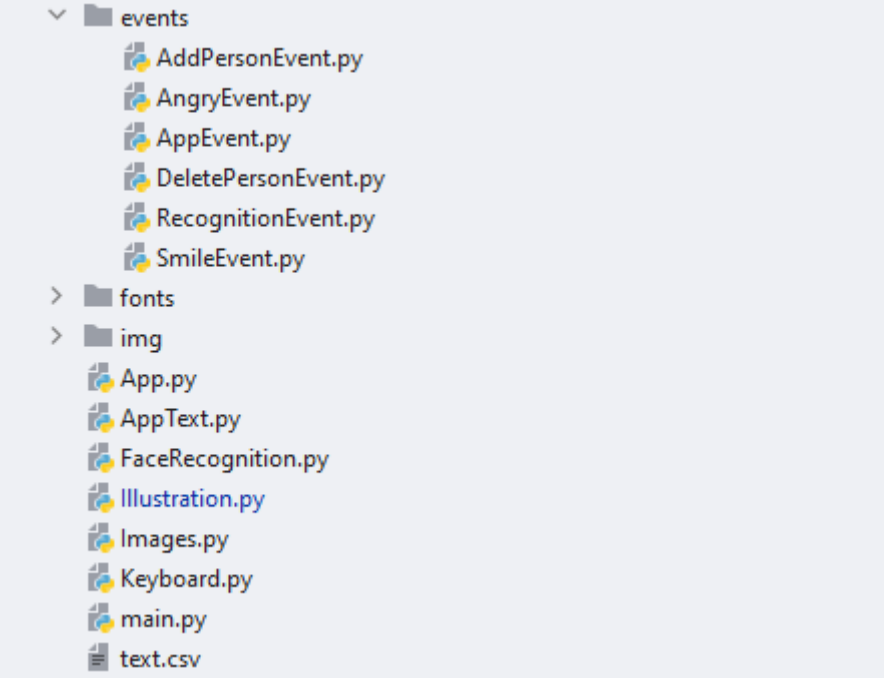
Joonis 2.5 names_db.csv

Csv fail, kus on seotud pildi id inimese nimega.

6) representations_facenet512.pkl

Fail, mis DeepFace näotuvastamise funktsioon loob. Seda on vaja selleks, et näotuvastus töötaks kiiresti. See on nägude piltide näojoonte kompaktne kirjeldus. Funktsioon loob seda selle puudumisel ja kui pildid on muudetud, siis peab seda kustutama. Seda peab kustutama sellepärast, et kui see fail on loodud, siis funktsioon enam pilte ei puuduta ja järgnevalt ei tea muudatustest. Ilma selle failita näotuvastus võttaks vähemalt sekundi, mis absoluutselt ei sobiks video jaoks.

2.2.2 Kood



Joonis 2.6 Koodide failid

1) main.py

Standartne fail mis loob IDE, kui alustada uue projekti. Ta ainult käivitab rakenduse.

2) App.py

See on peamine rakenduse kood, seal on loodud kasutajaliides ja on initsialiseeritud kõik komponendit. Ta sisaldab peamise funktsiooni, mis käib pidevalt.

3) AppText.py

See kood vastutab teksti eest, ta loeb teksti failist text.csv ja laseb teksti teistele komponentidele küsida.

4) FaceRecognition.py

Sisaldab kõik arvutinägamise funktsioonid.

5) Illustration.py

Kujutab ennast klass, mis on pilt kordinaatidega. Eksisteerib selleks, et joonistada pilte kaamera pildi peal õiges kohas. Samuti toetab läbipaistavust.

6) Images.py

See klass vastutab pilte laadimise ja genereerimise eest.

7) Keyboard.py

Põhimõtteliselt virtuaalne klaviatuur, mis on kolmes keeles ja seda saab joonistada kaamera pildi peal. On vaja selleks, et inimene saaks oma nimi kirjutada.

8) AppEvent.py

See on super klass event ehk sündmuse klassidele. Sündmuste objektid töötavad „paralleelselt". Need on töödeldud peamises funktsioonis, nad jälgivad aega ja omavad sissepääsu kõikidele Appis sisaldatavatele objektidele.

9) AddPersonEvent.py

See sündmus on loodud siis, kui inimene pole veel andmebaasis ja ta vajutab vasakpoolse nuppu. Siis temast pilt on hoolikalt tehtud ja ta saab enda nimi trükkida ja ennast salvestada.

10) DeletePersonEvent.py

See sündmus on loodud siis, kui inimene on andmebaasis ja ta vajutab vasakpoolse nuppu. Siis tal on võimalus ennast kustutada, vajutades linnukese.

11) RecognitionEvent.py

See sündmus toimub pidevalt, ta skaneerib kaamerapildi iga 750ms ja tuvastab kes on ja kui palju inimesi on kaamera ees. Samuti ta annab teada, kui inimene polnud veel tervitatud. Ta tuvastab näo ainult juhul, kui kaamera ees on üks inimene.

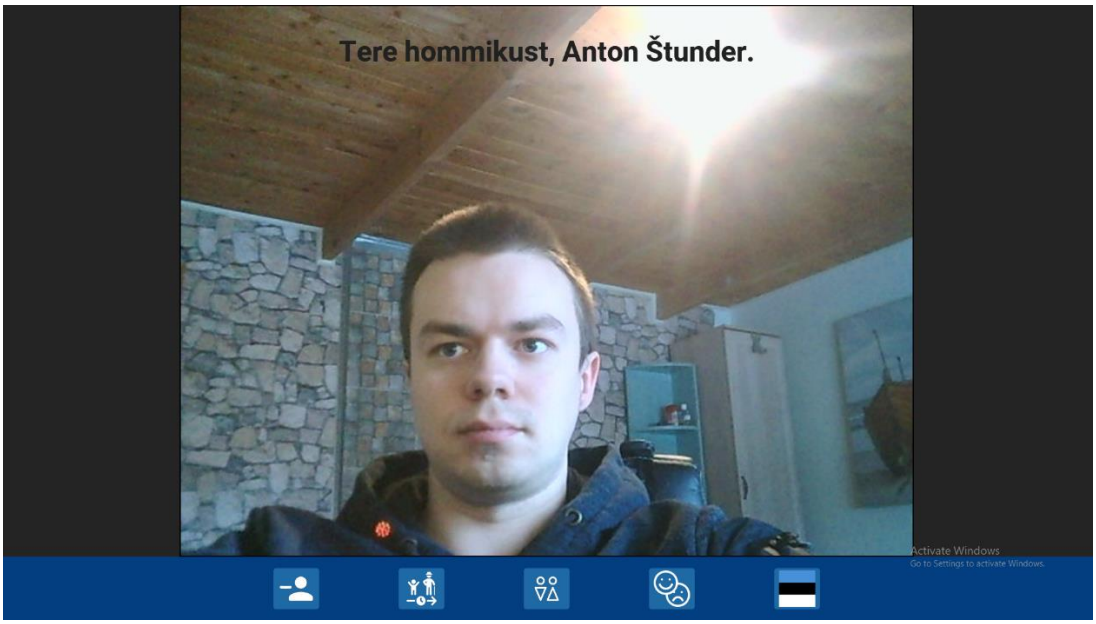
12) AngryEvent.py

Juhuslik sündmus, mis pakkub kasutajale näidata kurja näo ja siis reageerib sellele.

13) SmileEvent.py

Juhuslik sündmus, mis pakkub kasutajale naeratada ja siis reageerib sellele.

2.3 Töö lühikirjeldus



Joonis 2.7 Rakenduse pilt

Algusel programm kutsub App __init__ meetodit ehk konstruktorit, kus toimub seadistamine, kasutajaliidese loomine ja kõikide komponentide initsialiseerimine ja ressurside laadimine.

Pärast seda app_loop meetod on kutsutud, mis käib terve rakenduse elu ajal, kuna ta kutsub iseennast. Tema sees toimub pildi kaamerast võtmine ja selle pildi peal kõike vajalikke piltide ja teksti joonistamine. Samas meetodis toimub sündmuste töötlemine ja vajadusel inimeste tervitamine. Meetodi lõpus lõplik pilt on pandud label_widget kasutajalideese komponendile ja meetod kutsub iseennast.

Nupud on seadistatud vastavate App meetoditega, näiteks viimane nupp on keele muutmiseks ja ta kutsub change_language meetodi.

3. NÄOTUVASTUS RAKENDUSES

Kõik nägudega seotud funktsioonid on pakutud FaceRecognition klassi poolt.

- Algusel on konstandid, mis on laetud DeepFace mudelid ja nende tulemuste märgised (Joonis 3.1).

```
emotion_labels = ['angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral']
gender_labels = ['woman', 'man']
emotion_model = DeepFace.build_model('Emotion')
age_model = DeepFace.build_model('Age')
gender_model = DeepFace.build_model('Gender')
```

Joonis 3.1 DeepFace mudelid ja mudelite tulemuste märgised

- Konstruktor vajab teekonda kaustale, kus hakatakse nägude pildid ja names_db.csv faili salvestada. Pärast muutujate initsialiseerimist, meetod loeb andmed names_db.csv failist ja salvestab neid names_db muutujasse.

```
class FaceRecognition:
    def __init__(self, path):
        self.path = path
        self.names_db = dict()
        self.last_id = 0
        if self.path[:-1] != '/':
            self.path = self.path + '/'
        self.csv_path = f'{self.path}names_db.csv'
        if os.path.isfile(self.csv_path):
            with open(self.csv_path, 'r', newline='', encoding='utf-8') as csv_file:
                csv_reader = csv.reader(csv_file)
                for row in csv_reader:
                    id = row[0]
                    name = row[1]
                    self.names_db[id] = name
                    id = int(id)
                    if self.last_id < id:
                        self.last_id = id
```

Joonis 3.2 FaceRecognition konstruktor

- Get_faces meetod (Joonis 3.3) leiab näod pildi pealt DeepFace extract_faces funktsiooni abil. Isegi siis, kui extract funktsioon ei leia näo, ta ikka tagastab massiivi ühe elemendiga, kus x ja y on 0. Kuid üks kord ta tagastas tühja massiivi ja sellepärast meetodis on veel üks tingimuse kontroll len(faces) == 0. Samuti liiga väiksed näod on eemaldatud. Align on false sellepärast, et vastasel juhul see tekitas vea mõnedel juhtudel.

```python
def get_faces(self, image, backend='ssd'):
    try:
        faces = DeepFace.extract_faces(image, detector_backend=backend, enforce_detection=False, align=False)
    except cv2.error:
        print('cv2.error')
        return []
    if len(faces) == 0 or faces[0]['facial_area']['x'] == 0:
        return []
    result = []
    min_wh = len(image[0]) / 6
    for face in faces:
        if face['facial_area']['w'] >= min_wh or face['facial_area']['h'] >= min_wh:
            result.append(face)
    return result
```

Joonis 3.3 get_faces meetod näode leidmiseks

- Guess_emotion meetod (Joonis 3.4) arvab ära mis emotsioon inimese näol on. Seda on võimalik saavutada DeepFace.analyze funktsiooniga, kuid seda täpsus ja kiirus autorile ei sobinud. Siis autor leidis näide internetist [7] ja selle baasil ta kirjutas oma koodi.

```python
def guess_emotion(self, image, backend='ssd'):
    faces = self.get_faces(image, backend)
    gray_frame = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    results = []
    for face in faces:
        area = face['facial_area']
        x = area['x']
        y = area['y']
        w = area['w']
        h = area['h']
        face_roi = gray_frame[y:y + h, x:x + w]
        resized_face = cv2.resize(face_roi, (48, 48), interpolation=cv2.INTER_AREA)
        normalized_face = resized_face / 255.0
        reshaped_face = normalized_face.reshape(1, 48, 48, 1)
        predictions = emotion_model.predict(reshaped_face, verbose=False)[0]
        emotion_idx = predictions.argmax()
        emotion = emotion_labels[emotion_idx]
        result = {
            'region': area,
            'dominant_emotion': emotion
        }
        results.append(result)
    return results
```

Joonis 3.4 guess_emotion meetod, mis määrab näode emotsioone

- Guess_age meetod (Joonis 3.5) arvab ära mis vanus inimesel on. Seda on võimalik saavutada DeepFace.analyze funktsiooniga, kuid kuna autor juba kirjutas guess_emotion funktsiooni laiemalt, siis seda ka otsustas ta laiemalt kirjutada. Analyze funktsioon töötas piisavalt hästi, kuid esimesel käivitamisel ta laadis Age mudeli, aga pärast ümberkirjutamist on see rakenduse käivitamisel laetud. Kuidas funktsiooni teha autor leids DeepFace lähtekoodist.

```
def guess_age(self, image):
    faces = self.get_faces(image)
    results = []
    for face in faces:
        predictions = age_model.predict(face['face'].reshape(1,224,224,3), verbose=0)
        age = int(Age.findApparentAge(predictions[0, :]))
        result = {
            'region': face['facial_area'],
            'age': age
        }
        results.append(result)
    return results
```

Joonis 3.5 guess_age meetod, mis määrab näode vanuseid

- Guess_gender meetodil (Joonis 3.6) on sama lugu nagu eelmistega.

```
def guess_gender(self, image):
    faces = self.get_faces(image)
    results = []
    for face in faces:
        predictions = gender_model.predict(face['face'].reshape(1,224,224,3), verbose=0)
        gender = gender_labels[predictions.argmax()]
        result = {
            'region': face['facial_area'],
            'dominant_gender': gender
        }
        results.append(result)
    return results
```

Joonis 3.6 guess_gender meetod, mis määrab näode sugusid

- Get_face_image meetod (Joonis 3.7) on selleks, et välja lõigata natuke suurema pildi, kui get_face leidis. Mõned meetodit kasutavad seda, näiteks kui on vaja näo pildi salvestada, siis on parem pilt. Kuid on võimalik, et see ei suurenda näotuvastamise täpsust.

```
def get_face_image(self, image, region):
    x = region['x'] - 15
    y = region['y'] - 25
    w = region['w'] + 30
    h = region['h'] + 50
    if x < 0:
        x = 0
    if y < 0:
        y = 0
    return image[y:y + h, x:x + w]
```

Joonis 3.7 get_face_image meetod, mis lõigab välja näo pildist

- Verify meetod (Joonis 3.8) kontrollid, et kahel pildil oleks üks ja sama nägu, see meetod on kasutusel siis, kui toimub inimese lisamise sündmus.

```
def verify(self, frame, face):
    return DeepFace.verify(frame, face, enforce_detection=False, detector_backend='mediapipe', model_name='Facenet512')
```

Joonis 3.8 verify meetod, mis vaatab et kahel piltidel oleks sama nägu

- Recognize_person meetod (Joonis 3.9) tuvastab kellele näo kuulub, kui kaamera ees on üks inimene, või annab teada kui palju inimesi on. Siin autor kasutab mudeli Facenet512, sest ta on kiirem ja kergem, aga samal ajal täpsuse erinevust autor ei märkanud. Samuti detector on mediapipe, sest Ssd andis vea, aga standartne OpenCV andis halva tulemuse. Võib olla põhjus on selles, mis näo piirkonna OpenCV leiab. Find funktsioon suudab tuvastada mitu nägusi, kuid rakenduses seda pole vaja.

```
def recognize_person(self, image, backend='ssd') -> (str, str):
    faces = self.get_faces(image, backend)
    face = None
    if len(faces) == 1:
        region = faces[0]['facial_area']
        face = self.get_face_image(image, region)
    if face is not None:
        try:
            res = DeepFace.find(face, self.path, enforce_detection=False, detector_backend='mediapipe',
                                model_name='Facenet512', silent=True)
            if len(res) > 0 and len(res[0]) > 0:
                img_path = res[0]['identity'][0]
                img_name = re.search('\\d+.jpg', img_path).group()
                id = img_name[:img_name.rfind('.jpg')]
                if id not in self.names_db:
                    print('ERROR: no name in db')
                else:
                    return self.names_db[id], id
        except ValueError:
            pass
    return '', str(len(faces))
```

Joonis 3.9 recognize_person meetod, mis tegeleb näotuvastusega

- Save_names_db meetod (Joonis 3.10) salvestab andmed (id ja nimi) csv faili.

```
def save_names_db(self):
    with open(self.csv_path, 'w', newline='', encoding='utf-8') as csv_file:
        csv_writer = csv.writer(csv_file)
        for id in self.names_db:
            csv_writer.writerow([id, self.names_db[id]])
```

Joonis 3.10 save_names_db meetod, mis salvestab piltide ja nimide info csv faili

- Add_person meetod (Joonis 3.11) on uue näo lisamiseks. Lisab uue rea names_db struktuuri, siis salvestab seda ja siis salvestab näo pildi õiges kohas. Pärast seda eemaldab pkl faili, sest, nagu oli varem mainitud, seda on vaja et näotuvastuse funktsioon saaks uue pildi kasutada.

```
def add_person(self, name, image):
    self.last_id = self.last_id + 1
    self.names_db[str(self.last_id)] = name
    self.save_names_db()
    cv2.imwrite(f'{self.path}{self.last_id}.jpg', image)
    if os.path.isfile(f'{self.path}representations_facenet512.pkl'):
        os.remove(f'{self.path}representations_facenet512.pkl')
```

Joonis 3.11 add_person meetod inimese lisamiseks

- Delete_person meetod (Joonis 3.12) eemaldab inimese näo andmebaasist. Siis ta eemaldab pildi ja pkl faili, sest selles failis on andmed vanast pildist. Ja lõpus ta salvestab nimide andmebaasi csv faili.

```
def delete_person(self, id):
    del self.names_db[id]
    os.remove(f'{self.path}{id}.jpg')
    if os.path.isfile(f'{self.path}representations_facenet512.pkl'):
        os.remove(f'{self.path}representations_facenet512.pkl')
    self.save_names_db()
```

Joonis 3.12 delete_person meetod inimese eemaldamiseks

4. RAKENDUSE GRAAFILINE OSA

4.1 Kasutajaliides

4.1.1 Initsialiseerimine

Initsialiseerimine toimub App klassi konstruktoris. Initsialiseeritakse kõik rakenduse komponendit, nii kasutajaliidese, kui ka muud.

Algusel ta loob FaceRecognition objekti, siis kaamera ja Tkinter objekti. Current_event on sündmus, mis hakatakse töötlema, kuid algusel pole mingid sündmus ja ta on None. Siis ta seadistab esialgse keele vene keeleks ja loob AppText objekti. (Joonis 4.1)

```
self.face_recognition = FaceRecognition('E:/face_recognition')

self.cam = cv2.VideoCapture(0)
self.app = ctk.CTk()
self.app.attributes('-fullscreen', True)
self.app.bind('<Escape>', lambda e: self.app.quit())
ctk.set_appearance_mode('Dark')
self.current_event = None

self.language = 'ru'
self.text = AppText(self.language)
```

Joonis 4.1 App konstruktor algus

Siis ta määrab suurused, mis sõltuvad kaamera ja monitori resolutsioonidest. Hiljem nad tulevad kasuks selleks, et joonistada pilte õge suurusega ja õige koha peal. (Joonis 4.2)

```
self.cam_width = self.cam.get(3)
self.cam_height = self.cam.get(4)
self.screen_width = self.app.winfo_screenwidth()
self.screen_height = self.app.winfo_screenheight()
self.cam_image_height = self.screen_height - FOOTER_HEIGHT
self.cam_image_width = int(self.cam_width * (self.cam_image_height / self.cam_height))
self.cam_image_w_ratio = self.cam_image_width / self.cam_width
self.cam_image_h_ratio = self.cam_image_height / self.cam_height
```

Joonis 4.2 kaamera pildi suuruste arvutamine

Edasi ta loob label_widget, mis hakkab sisaldama lõpliku veebikaamera pildi. (Joonis 4.3)

```
self.label_widget = tk.Label(self.app)
self.label_widget.pack()
self.label_widget.configure(background='black')
```

Joonis 4.3 label_widget loomine, mis on veebikaamera pildi konteineriks

Siis ta loob images objekti ja illustrations massiivi. Kutsub tekstide funktsioonid, kuna

nemad initsialiseerivad teksti muutujad. Images objekt laeb rakenduse pilte ja laseb ka mõned genereerida. Edasi konstruktor genereerib ja salvestab näo ovaali pildi, mis tuleb kasuks inimese lisamise ajal. (Joonis 4.4)

```
self.images = Images(self.cam_image_width - 50)
self.illustrations = []
self.show_top_text("", 0)
self.show_center_text("", 0)
self.face_oval_image = self.images.create_face_oval_image(self.cam_image_width, self.cam_image_height)
self.face_oval_image_visible = False
```

Joonis 4.4 Piltide muutujate initsialiseerimine App konstruktoris

Siis ta loob footer_containeri, mis hakkab sisaldama erinevad footerid, mis sõltub sellest mis nuppud on konkreetses ollukorras vajalikud. On olemas kaks footerit main_footer ja ok_cancel_footer. Siin ta loob main_footeri ja siis lisab ka nuppe, nendest autor kirjutab järgmises peatükis. (Joonis 4.5).

```
self.footer_container = tk.Frame(self.app, background=FOOTER_BG)
self.footer_container.pack(fill='both', expand=True)

self.main_footer = tk.Frame(self.footer_container, background=FOOTER_BG)
self.main_footer.pack(pady=(10, 10))
```

Joonis 4.5 footer_container ja main_footeri loomine

```
self.ok_cancel_footer = tk.Frame(self.footer_container, background=FOOTER_BG)
```

Joonis 4.6 ok_cancel_footeri loomine

Siis ta loob keyboard objekti. Võtab pildi kaamerast ja kutsub korra verify meetodi kaamera pildi kasutates, et ta laeks vajalikud asjad, sest esimene kutsumine võtab paar sekundit aega. Edasi recognition_event on loodud, mis vastutab näotuvastuse eest ja lõpus ta käivitab rakenduse ja Tkinteri peamised funktsioonid. (Joonis 4.7)

```
self.keyboard = Keyboard(self.label_widget, self.language)

ret, frame = self.cam.read()
self.last_frame = frame
self.face_recognition.verify(self.last_frame, self.last_frame)
from events.RecognitionEvent import RecognitionEvent
self.recognition_event = RecognitionEvent(self)
self.frame_time = get_time_ms()
self.last_event_time = self.frame_time
self.last_random_event = ''
self.app_loop()

self.app.mainloop()
```

Joonis 4.7 App konstruktori lõpp

4.1.2 Nupud

Nupud (Tabel 4.1) on CustomTkinteri poolt, sest nad on parema kujuga. Nad on loodud konstruktori abil (Joonis 4.8) ja asetatud grid meetodi abil (Joonis 4.9).

```
self.btn_add_person = ctk.CTkButton(self.main_footer, text="", command=self.add_person,
                                    image=self.images.add_person, width=BTN_WIDTH)
```

Joonis 4.8 Nupu loomine

```
self.btn_add_person.grid(row=0, column=0, padx=BTN_PAD_X, pady=BTN_PAD_Y)
```

Joonis 4.9 Nupu asetamine

```
BTN_WIDTH = 80
BTN_PAD_X = 70
BTN_PAD_Y = 10
```

Joonis 4.10 Nupu konstandid

Tabel 4.1 Nupude loetelu

Nimi	Funktsioon	Footer	Pilt
btn_add_person	add_person	main	 Joonis 4.11 [13]
btn_delete_person	delete_person	main	 Joonis 4.12 [13]
btn_guess_age	guess_age	main	 Joonis 4.13 [14]
btn_guess_gender	guess_gender	main	 Joonis 4.14 [15]

btn_guess_emotion	guess_emotion	main	 Joonis 4.15 [16]
btn_change_language	change_language	main	 Joonis 4.16 [17]
btn_cancel	cancel_event	ok_cancel	 Joonis 4.17 [18]
btn_ok	confirm_event	ok_cancel	 Joonis 4.18 [19]

Nuppude pildid on leitud googleist ja värv on muudetud valgeks GIMP rakenduse abil.

4.1.3 Peamine funktsioon

Algusel ta võtab pildi kaamerast ja salvestab seda, siis ta määrab pildistamise aega ja suurendatud pildi salvestab captured_image muutujasse. Selle suurendatud pildi peal hakkavadki toimuma kõik joonistused. (Joonis 4.19)

```
def app_loop(self):
    ret, frame = self.cam.read()
    self.last_frame = frame

    time_delta = get_time_ms() - self.frame_time
    self.frame_time = get_time_ms()

    opencv_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
    captured_image = Image.fromarray(opencv_image).resize(size=(self.cam_image_width, self.cam_image_height))
```

Joonis 4.19 Peamise funktsiooni algus

Siis ta töötleb current_event sündmuse, kui seda on, ja töötleb recognition_event, mis vastutab selle eest, et jälgib kes on kaamera ees, kui palju inimesi on ja kas on vaja tervitada. Samuti, kui kaamera ees pole kedagi, ta uuendab last_event_time, sest

sellest sõltub juhusliku sündmuse toimumine. Kui ta seda ei tee, siis see hakkab kohe käima. (Joonis 4.20)

```
if self.current_event is not None:
    self.current_event.process(self.frame_time)
self.recognition_event.process(self.frame_time)

if self.recognition_event.people_n == 0:
    self.last_event_time = self.frame_time
```

Joonis 4.20 Sündmuste töötlemine

Siis ta vajadusel tervitab kasutaja ja proovib juhusliku sündmuse alustada. (Joonis 4.21)

```
if self.current_event is None and not self.recognition_event.greeted:
    self.last_event_time = self.frame_time
    if self.recognition_event.name != '':
        self.show_top_text(f'{self.text.get_hello().capitalize()}, {self.recognition_event.name}.')
    elif self.recognition_event.people_n > 0:
        self.show_top_text(f'{self.text.get_hello().capitalize()}.')
    self.recognition_event.greeted = True

self.try_random_event()
```

Joonis 4.21 Vajadusel tervitamine

Esimene nupp main footeris on kas inimese lisamiseks või eemaldamiseks. Kui inimene on andmebaasis, siis nupp on eemaldamiseks, vastasel juhul lisamiseks. (Joonis 4.22)

```
if self.is_add_person_btn and self.recognition_event.name != '':
    self.is_add_person_btn = False
    self.btn_add_person.grid_forget()
    self.btn_delete_person.grid(row=0, column=0, padx=BTN_PAD_X, pady=BTN_PAD_Y)
elif not self.is_add_person_btn and self.recognition_event.name == '':
    self.is_add_person_btn = True
    self.btn_delete_person.grid_forget()
    self.btn_add_person.grid(row=0, column=0, padx=BTN_PAD_X, pady=BTN_PAD_Y)
```

Joonis 4.22 Määramine kas vasakpoolne nupp on inimese lisamiseks või eemaldamiseks

Siis ta joonistab lisatud illustratsioonid. Need on suurendava läbipaistavusega, ja kui pilt on liiga läbipaistev, siis ta eemaldab neid. Ja siis ta joonistab klaviatuuri. Klaviatuuril on visible muutuja, mis otsustab, kas joonistada või mitte. (Joonis 4.23)

```
visible_illustrations = []
for illustration in self.illustrations:
    illustration.alpha = illustration.alpha - ILLUSTRATION_ALPHA_DECREASE_RATE * time_delta
    if illustration.alpha > 20.0:
        captured_image = illustration.draw(captured_image)
        visible_illustrations.append(illustration)
self.illustrations = visible_illustrations
captured_image = self.keyboard.draw_keyboard(captured_image)
```

Joonis 4.23 Piltide ja klaviatuuri joonistamine

Lõpus ta paigutab finaalse pildi label_widgetisse ja käivitab selle sama funktsiooni 10ms hiljem. (Joonis 4.24)

```
photo_image = ImageTk.PhotoImage(image=captured_image)
self.label_widget.photo_image = photo_image
self.label_widget.configure(image=photo_image)
self.label_widget.after(10, self.app_loop)
```

Joonis 4.24 Veebikaamera pildi uuendamine ja uuesti peamise funktsiooni käivitamine

4.2 Illustratsioonid ja tekst

4.2.1 Pildid

Selles peatükis on räägitud Images klassist. See on klass, mis vastutab piltide laadimise ja genereerimise eest. Konstruktor (Joonis 4.25) laeb kõik pildid open_image funktsiooniga (Joonis 4.26), mis laeb pilte Pillow(PIL) teegi abil. Nuppude piltide suurused ta muudab, et nad oleksid sobiva suurusega, mis hiljem ei muutu (Joonis 4.27). Max_w argument määrab maksimaalse laiuse, mis on vaja vaid selleks, et tekst poleks piiridest välja astunud.

```
ICON_SIZE = 64

class Images:
    def __init__(self, max_w):
        self.max_w = max_w

        self.male = self.open_image('img/gender/male.png')
```

Joonis 4.25 Images klassi konstruktori osa

```
self.ok = ImageTk.PhotoImage(self.open_image('img/icons/ok.png').resize(size=(ICON_SIZE, ICON_SIZE)))
```

Joonis 4.26 Nupu pildi laadimine

```
def open_image(self, path: str) -> Image:
    return Image.open(path, 'r').convert('RGBA')
```

Joonis 4.27 Meetod pildi laadimiseks

Tekst rakenduses on ka pildina, kuna ta on joonistatud kaamera pildile. (Joonis 4.28)

```
def create_text_image(self, text, size, color='black', bold=False) -> Image.Image:
    if bold:
        font = ImageFont.truetype('fonts/Roboto-Bold.ttf', size, encoding='unic')
    else:
        font = ImageFont.truetype('fonts/Roboto-Regular.ttf', size, encoding='unic')
    width, height = Images.get_text_wh(text, font)
    img = Image.new('RGBA', (width, height), (0, 0, 0, 0))
    draw = ImageDraw.Draw(img)
    draw.text((0, 0), text, color, font)
    if width >= self.max_w:
        ratio = width / self.max_w
        height = int(height * ratio)
        width = self.max_w
        return img.resize(size=(width, height))
    return img
```

Joonis 4.28 Meetod teksti genereerimiseks

Näo ovaali genereerimine. Meetod loob osaliselt läbipaistva musta tausta ja selle peal joonistab näo ovaali täielikult läbipaistva värviga. (Joonis 4.29)

```
@staticmethod
def create_face_oval_image(width, height):
    image = Image.new('RGBA', (width, height), (0, 0, 0, 100))
    draw = ImageDraw.Draw(image)
    top = round(height * 0.1)
    bottom = round(height * 0.9)
    h = bottom - top
    w05 = round(0.5 * h / 1.25)
    center = round(width / 2)
    draw.ellipse((center - w05, top, center + w05, bottom), fill=(0, 0, 0, 0))
    return image
```

Joonis 4.29 Meetod näo ovali pildi genereerimiseks

Progressiriba kujutab ennast rohelised rüstkülikud musta tausta peal. (Joonis 4.30)

```
@staticmethod
def create_progress_bar(width, height, needed, completed):
    bar_width = int(width / needed)
    image = Image.new('RGBA', (bar_width * needed, height), (220, 220, 220, 255))
    draw = ImageDraw.Draw(image)
    for i in range(0, completed):
        draw.rectangle((i * bar_width, 0, i * bar_width + bar_width, height), fill=(0, 200, 50),
                       outline=(220, 220, 220), width=2)
    return image
```

Joonis 4.30 Meetod progressriba genereerimiseks

4.2.2 Illustratsioonid

Illustration klass kujutab ennas pildi kordinaatidega ja läbipaistavusega. Ta omab meetodi draw, mis joonistab pildi õigel kohal ja õige läbipaistavusega. Läbipaistavusega

joonistamise algoritmi oli leitud internetist [8]. Olen proovinud mitu variandi ja see oli ainuke mis töötas. Loome uue pildi läbipaistva tagataustaga, et suurus oleks sama, mis on pildil kuhu joonistame. Siis erinevate manipulatsioonidega lisame sellele uuele pildile osaliselt läbipaistva illustratsiooni. Lõpus, alpha_composite meetodi abil, saame neid ühendada. (Joonis 4.31)

```
class Illustration:
    def __init__(self, x, y, image: Image, alpha=255):
        self.x = int(x)
        self.y = int(y)
        self.image = image
        self.alpha = alpha

    def draw(self, canvas: Image):
        if self.alpha >= 250.0:
            canvas.paste(self.image, (self.x, self.y), self.image)
            return canvas
        else:
            layer = Image.new('RGBA', canvas.size, (0, 0, 0, 0))
            layer.paste(self.image, (self.x, self.y))
            layer2 = layer.copy()
            layer2.putalpha(int(self.alpha))
            layer.paste(layer2, layer)
            return Image.alpha_composite(canvas, layer)
```

Joonis 4.31 Illustration klass

Illustratsioonid on kasutusel selleks, et näidata kasutajale, mis arvuti määras nende näo sooks, vanuseks, emotsiooniks. Guess_age ja guess_emotion (Joonis 4.32) meetodit on suhteliselt sarnased. Määrata näod ja emotsioonid ja siis lisada illustratsioonid add_facial_illustration meetodiga. Näitab tulemuse teksti ka.

```
def guess_emotion(self):
    self.cancel_event()
    result = self.face_recognition.guess_emotion(self.last_frame)
    self.illustrations = []
    for r in result:
        emotion = r['dominant_emotion']
        region = r['region']
        self.add_facial_illustration(region, getattr(self.images, emotion))
        self.show_facial_traits_text(emotion)
```

Joonis 4.32 Meetod, mis joonistab määratud inimeste emotsioone

Guess_age meetod (Joonis 4.33) esialgul loob suure pildi vanuse tekstiga, mille suurus on muudetud õigeks add_facial_illustration meetodis. Siis näitab teksti. Kasutusel on kaks teksti varianti, sest vene keeles on kasutusel erinevad sõnad, sõltuvalt vanusest. Samuti nendes meetodites enne illustratsioonide joonistamist, vanad illustratsioonid on eemaldatud ja praegune sündmus on lõpetatud.

```
def guess_age(self):
    self.cancel_event()
    result = self.face_recognition.guess_age(self.last_frame)
    self.illustrations = []
    for r in result:
        age = r['age']
        region = r['region']
        self.add_facial_illustration(region, self.images.create_text_image(str(age), 250, FONT_COLOR, True),
                                     0.35)
        if (10 <= age <= 20) or age % 10 == 0 or age % 10 >= 5:
            self.show_facial_traits_text('age', age)
        else:
            self.show_facial_traits_text('age2', age)
```

Joonis 4.33 Meetod, mis joonistab määratud inimeste vanuseid

Add_facial_illustration meetod (Joonis 4.34) lisab illustratsiooni õige kohta. Selleks ta algusel muudab näo piirkonna, et ta vastaks suurendatud pildile translate_region meetodi abil (Joonis 4.35). Siis ta muudab pildi suurust, et see oleks mingi protsent näo piirkonna laiusest. Lõpus pilt on lisatud illustrations massiivi, kust teda joonistatakse peamises funktsioonis.

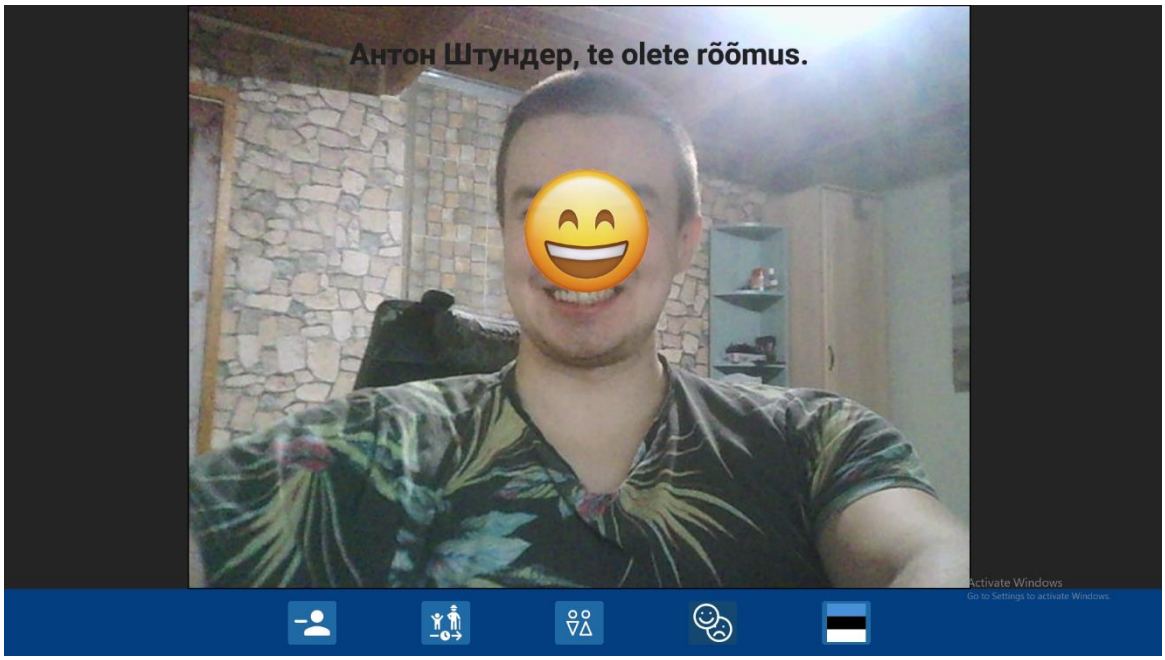
```
def add_facial_illustration(self, region, image, width_ratio=0.5):
    region = self.translate_region(region)
    w = region['w'] * width_ratio
    ratio = w / image.width
    h = image.height * ratio
    image = image.resize((int(w), int(h)))

    self.illustrations.append(Illustration(region['x'] + region['w'] / 2 - image.width / 2,
                                           region['y'] + region['h'] / 2 - image.height / 2,
                                           image))
```

Joonis 4.34 Meetod näole illustratsiooni lisamiseks, näiteks emotsiooni emoji

```
def translate_region(self, region):
    return {
        'x': region['x'] * self.cam_image_w_ratio,
        'w': region['w'] * self.cam_image_w_ratio,
        'y': region['y'] * self.cam_image_h_ratio,
        'h': region['h'] * self.cam_image_h_ratio
    }
```

Joonis 4.35 Meetod, mis arvutab kordinaate vastavalt muudetud veebikamera pildi suurusest



Joonis 4.36 guess_emotion meetodi tulemus

Tabel 4.2 illustratsioonid

Nimi	**Pilt**
angry.png	 Joonis 4.37 [20]
disgust.png	 Joonis 4.38 [21]

fear.png	 Joonis 4.39 [22]
happy.png	 Joonis 4.40 [23]
neutral.png	 Joonis 4.41 [24]
sad.png	 Joonis 4.42 [25]
surprise.png	 Joonis 4.43 [26]

female.png	 Joonis 4.44 [27]
male.png	 Joonis 4.45 [28]

4.2.3 Teksti joonistamine

On loodud kaks meetodi teksti joonistamiseks (Joonis 4.46). Nad on sarnased ja üks joonistab teksti üleval ja teine keskel. Esimese rakendus kasutab info edastamiseks ja teise kasutab AddPersonEvent, et näidata mis on kasutaja trükkinud.

```
def show_top_text(self, text: str, show_time=3000, size=50, color=FONT_COLOR):
    self.top_text = self.images.create_text_image(text, size, color, True)
    self.top_text_time = show_time
    self.top_text_show_start = get_time_ms()

def show_center_text(self, text: str, show_time=3000, size=50, color=FONT_COLOR):
    self.center_text = self.images.create_text_image(text, size, color)
    self.center_text_time = show_time
    self.center_text_show_start = get_time_ms()
```

Joonis 4.46 Meetodid teksti joonistamiseks

Show_facial_traits_text funktsioon (Joonis 4.47) on selleks et näidata näo omaduse info. Tema joonistab üleval teksti ainult juhul, kui rakenduses on hetkel üks kasutaja. Põhjus on näiteks see, et inimestel on erinevad vanused. Juhul kui kasutaja on andmebaasil, siis ta pöördub tema poole tema nimi kasutades.

```
def show_facial_traits_text(self, trait, *args):
    if self.recognition_event.people_n == 1:
        if self.recognition_event.name != '':
            self.show_top_text(
                f'{self.recognition_event.name}, {self.text.get_text_from_db(trait).format(*args)}.')
        else:
            self.show_top_text(f'{self.text.get_text_from_db(trait).capitalize().format(*args)}.')
```

Joonis 4.47 Meetod, mis joonistab teksti vastavalt näo omaduste määramise meetodite tulemustest

4.3 Klaviatuur

Klaviatuur on realiseeritud Keyboard klassis.

Joonis 4.48 Klaviatuuri kuju

Erinevate keelte tähed on kolmerealises matriksis. (Joonis 4.49)

```
EE_KEYS = [
    ['Ö', 'Ä', 'E', 'R', 'T', 'Ü', 'U', 'I', 'O', 'P'],
    ['A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L'],
    ['Z', 'Ž', 'Õ', 'Š', 'C', 'V', 'B', 'N', 'M']
]

EN_KEYS = [
    ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P'],
    ['A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L'],
    ['Z', 'X', 'C', 'V', 'B', 'N', 'M']
]

RU_KEYS = [
    ['Й', 'Ц', 'У', 'К', 'Е', 'Н', 'Г', 'Ш', 'Щ', 'З', 'Х'],
    ['Ф', 'Ы', 'В', 'А', 'П', 'Р', 'О', 'Л', 'Д', 'Ж', 'Э'],
    ['Я', 'Ч', 'С', 'М', 'И', 'Т', 'Ь', 'Б', 'Ю', 'Ъ', 'Ё']
]
```

Joonis 4.49 Klaviatuuri tähtede matriksid

Konstruktor lisab hiire vajutuse sündmusele funktsiooni click. (Joonis 4.50)

```
class Keyboard:
    def __init__(self, parent, language, image_w, image_h):
        self.parent = parent
        self.parent.bind("<Button-1>", self.click)
        self.visible = False
        self.image_w = image_w
        self.image_h = image_h
        self.language = language
        self.callback = None
        self.font = ImageFont.truetype('fonts/Roboto-Bold.ttf', FONT_SIZE, encoding='unic')
```

Joonis 4.50 Keyboard konstruktor

Klahvide kuju on määratud konstantidega. (Joonis 4.51)

```
KEY_SIZE = 60
FONT_SIZE = 30
FILL = (31, 106, 165, 180)
PADDING = 10
BOTTOM_CANVAS_PADDING = 100
```

Joonis 4.51 Klaviatuuri konstandid

Klaviatuuri joonistamine toimub draw_keyboard meetodiga (Joonis 4.52). Esiteks see loob läbipaistva pildi veebikaamera pildi suurusega. Siis ta kutsub calculate_xy meetodi, mis määrab klaviatuuri kordinaate nii, et see oleks horisontaalselt keskel ja et tema põhi oleks BOTTOM_CANVAS_PADDING (100) pikslit pildi alusest. Samuti get_keys meetodiga ta saab praeguse keele tähti. Siis ta joonistab neid draw_key meetodiga. Järgmisena ta eraldi joonistab tühiku ja backspace klahvid. Lõpus ta ühendab veebikaamera ja klaviatuuri pildid funktsiooniga alpha_composite, sest see funktsioon arvestab klahvide läbipaistavusega.

```
def draw_keyboard(self, image: Image):
    if not self.visible:
        return image
    keyboard_image = Image.new('RGBA', image.size, (0, 0, 0, 0))
    keys = self.get_keys()
    x, y = self.calculate_xy()
    draw = ImageDraw.Draw(keyboard_image)
    for i in range(len(keys)):
        row_keys = keys[i]
        for j in range(len(row_keys)):
            self.draw_key(draw, row_keys[j], x + (KEY_SIZE + PADDING) * j, y + (KEY_SIZE + PADDING) * i)

    n_row_keys = len(keys[0])
    space_y = y + 3 * KEY_SIZE + 3 * PADDING
    space_w = (n_row_keys - 1) * KEY_SIZE + (n_row_keys - 2) * PADDING
    draw.rectangle((x, space_y, x + space_w, space_y + KEY_SIZE), fill=FILL)

    self.draw_key(draw, '<', x + space_w + PADDING, space_y)

    return Image.alpha_composite(image, keyboard_image)
```

Joonis 4.52 Klaviatuuri joonistamise meetod

Click meetod (Joonis 4.53) määrab vajutatud klahvi ja kutsub callback meetodi saadates klahvi tähe argumendina.

```
def click(self, event):
    if self.visible and self.callback is not None:
        keys = self.get_keys()
        x, y = self.calculate_xy()
        c_x = event.x
        c_y = event.y
        rel_y = c_y - y
        row = int(rel_y / (KEY_SIZE + PADDING))
        if row < 0 or row > 3 or rel_y > row * (KEY_SIZE + PADDING) + KEY_SIZE:
            return

        rel_x = c_x - x
        col = int(rel_x / (KEY_SIZE + PADDING))
        if row != 3:
            row_keys = keys[row]
            if 0 <= col < len(row_keys) and rel_x <= col * (KEY_SIZE + PADDING) + KEY_SIZE:
                self.callback(keys[row][col])
        else:
            last_col = len(keys[0]) - 1
            if col < last_col:
                if rel_x <= (last_col - 1) * (KEY_SIZE + PADDING) + KEY_SIZE:
                    self.callback(' ')
            elif col == last_col:
                if rel_x <= col * (KEY_SIZE + PADDING) + KEY_SIZE:
                    self.callback('<')
```

Joonis 4.53 Klaviatuuri vajutamise meetod

5. MUUD KOMPONENDID

5.1 AppText

AppText klass vastutab, nagu tema nimest tuleneb, rakenduse teksti eest. Konstruktori argumendiks on esialgne keel ja tema loeb teksti angmed text.csv failist. (Joonis 5.1)

```
class AppText:
    def __init__(self, language):
        self.language = language
        self.text_db = dict()
        with open('./text.csv', 'r', newline='', encoding='utf-8') as csv_file:
            csv_reader = csv.reader(csv_file, delimiter=';')
            for row in csv_reader:
                self.text_db[row[0]] = [row[1], row[2], row[3]]
```

Joonis 5.1 AppText konstruktor

Text.csv failis (Joonis 5.2) esimene veerg on teksti id, siis tulevad inglise, eesti ja vene keelsed tekstid.

```
hello;hello;tere;привет
good_morning;good morning;tere hommikust;доброе утро
good_afternoon;good afternoon;tere päevast;добрый день
good_evening;good evening;tere õhtust;добрый вечер
angry;you are angry;te olete kuri;вы злы
disgust;you are disgusted;te tunnete vastikust;вы чувствуете отвращение
fear;you are scared;te kardate;вы напуганы
happy;you are happy;te olete rõõmus;вы рады
sad;you are sad;te olete kurb;вам грустно
surprise;you are surprised;te olete üllatuses;вы удивлены
neutral;you are calm;te olete rahulik;вы спокойны
```

Joonis 5.2 text.csv osa

Siis teksti saab küsida get_text_from_db meetodiga (Joonis 5.3) ja tema tagastab teksti õiges keeles.

```
def get_text_from_db(self, id):
    if id not in self.text_db:
        return id
    lang_i = 0
    if self.language == 'ee':
        lang_i = 1
    elif self.language == 'ru':
        lang_i = 2
    return self.text_db[id][lang_i]
```

Joonis 5.3 Teksti hankimise meetod

Samuti on olemas meetod get_hello, mis tagastab tervitamist teksti. Tema tagastab juhuslikult kas lihtsalt “tere” või tervitust, mis sobiks kellaajale nagu “tere õhtust”. (Joonis 5.4)

```
def get_hello(self):
    rand = random.randint(0, 1)
    print(rand)
    if rand == 0:
        hour = datetime.now().hour
        if 6 <= hour < 12:
            return self.get_text_from_db('good_morning')
        elif 12 <= hour < 17:
            return self.get_text_from_db('good_afternoon')
        elif 17 <= hour < 22:
            return self.get_text_from_db('good_evening')
    return self.get_text_from_db('hello')
```

Joonis 5.4 Tervitamise teksti hankimise meetod

5.2 Event klassid

5.2.1 AppEvent

Events klassid on rakenduses selleks, et lahendada ülesanned, mis peavad käima põhi funktsiooniga paralleelselt. Reaalsuses nad pole päriselt paralleelsed, kuid nad on töödeldud iga peamise programmi iteratsiooniga kuni enda elu lõpuni. AppEvent klass (Joonis 5.5) on super klass nendele. Kontruktoris ta saab peamise rakenduse objekti, et ta saaks teda meetodeid kasutada. Process meetod ongi see mis on peamises funktsioonis pidevalt kutsetud ja ta võtab kaadri aega argumendina. Siis on cancel meetod, mis tagastab kõik asjad esialgsesse seisundisse, näiteks kui kasutaja vajutas cancel, ja confirm funktsioon, mis on kutsutud kui inimene vajutab ok nuppu.

```
class AppEvent:
    def __init__(self, app: App):
        self.app = app

    def process(self, timestamp: int):
        pass

    def cancel(self):
        pass

    def confirm(self):
        pass
```

Joonis 5.5 AppEvent konstruktor

5.2.2 RecognitionEvent

RecognitionEvent on teistest erinev, sest ta on loodud rakenduse käivitamisel ja ta elab lõpuni. Tema vastutab selle eest, et jälgib kes on kaamera eest, või kui palju inimest on kaamera ees. Siis tal on ka selline omadus nagu lives ehk elud. Seda on vaja selleks, et ta ei teeks liiga kiired otsuseid, kuna inimene võib korraks näo katta, või keegi tagataustal vaatab korra kaamerasse jne. (Joonis 5.6)

```
LIVES = 4

class RecognitionEvent(AppEvent):
    def __init__(self, app: App):
        super().__init__(app)
        self.timestamp = 0
        self.recognition_lives = 0
        self.name = ''
        self.id = '0'
        self.people_n = 0
        self.people_n_lives = 0
        self.last_person_timestamp = 0
        self.greeted = False
```

Joonis 5.6 RecognitionEvent konstruktor

Tema process meetod (Joonis 5.7) käivitub iga 750 ms. Algusel ta küsib face_recognition objektist, et ta tuvastaks näod. Recognize_person meetod tagastab inimese nimi ja id, kui inimene on tuvastatud ja ta on üksi, vastasel juhul nimi on tühi ja id näitab inimeste arvu. Kui inimene pole selles iteratsioonis tuvastatud, aga kui ta oli varem tuvastatud ja tal on rohkem nullist elu, siis ta kaotab ühe elu.

```
def process(self, timestamp: int):
    if timestamp - self.timestamp >= 750:
        self.timestamp = timestamp
        name, id = self.app.face_recognition.recognize_person(self.app.last_frame)
        if name == '':
            if self.recognition_lives > 0:
                self.recognition_lives = self.recognition_lives - 1
            else:
```

Joonis 5.7 RecognitionEvent process meetodi algus

Kui tal pole elu, siis id muutujas on inimeste arv. Aga inimeste arvu muutujas on ka mitu elu ja kui uus inimeste arv pole võrdne praegusega ja praegune nullist suurem, siis toimub sama loogika tema kohta. (Joonis 5.8)

```
else:
    people_n = int(id)
    if people_n != self.people_n:
        if self.people_n_lives > 0 and self.people_n != 0:
            self.people_n_lives = self.people_n_lives - 1
            people_n = self.people_n
        else:
            self.people_n_lives = LIVES
    else:
        self.people_n_lives = LIVES
```

Joonis 5.8 Inimeste numbri määramine

Siis meetod määrab kas on vaja inimese tervitada (Joonis 5.9). On vaja siis, kui inimeste arv on suurendanud või uus inimene on teadmata, kuid ennem oli tuttav nägu. Ja nimeks on siis tühjus ja inimeste arv on määratud.

```
        if self.people_n < people_n or self.name != '':
            self.greeted = False

        self.name = ''
        self.id = '0'
        self.people_n = people_n
else:
```

Joonis 5.9 Inimeste number on määratud

Juhul kui inimene on tuvastatud ja id ei klappi, siis id on uuendatud (Joonis 5.10). Ja kui kaamera objektiivis ennem kedagi ei olnud või oli ainult tundmatu inimene, siis on vaja tervitada. Loogika on selline, et kaamera ees võib olla kaks inimest ja võib juhtuda nii, et ühel iterratsioonil üks nägu ei leidu, siis see tingimuse kontroll kaitseb selle eest, et inimene oleks üleliigselt tervitatud.

```
else:
    if self.id != id:
        self.name = name
        self.id = id
        if self.people_n <= 1:
            self.greeted = False
    self.recognition_lives = LIVES
    self.people_n = 1
```

Joonis 5.10 Näo oli tuvastatud

Lõpus on salvestatud info viimase märgatud näo kohta. (Joonis 5.11)

```
if self.people_n > 0:
    self.last_person_timestamp = timestamp
```

Joonis 5.11 process meetodi lõpp

5.2.3 AddPersonEvent

See klass vastutab inimese lisamise eest. Konstruktoris (Joonis 5.13) ta vahetab kasutajaliidese footeri, sest vajaks lähevad ok ja cancel nupud. Kuid pildi tegemise ajal ok nupp on peidetud, sest see protsess toimub automaatselt. Samuti on loodud found_once ja face_saved muutujad, mis tulevad kasulikuks pildi tegemise ajal. Sellel sündmusel on kaks seisundi: pildi tegimine ja nimi sisestamine (Joonis 5.12). Face muutujas ta salvestab näo pildi.

```
class FaceAddState(Enum):
    TAKE_PICTURE = 1
    ADD_NAME = 2

class AddPersonEvent(AppEvent):
    face: cv2.typing.MatLike
```

Joonis 5.12 AddPersonEvent seisundid

```
def __init__(self, app: App):
    super().__init__(app)
    self.found_once = False
    self.face_saved = False
    self.name = ''
    self.state = FaceAddState.TAKE_PICTURE
    self.app.face_oval_image_visible = True
    self.app.main_footer.pack_forget()
    self.app.ok_cancel_footer.pack(pady=(10, 10))
    self.app.btn_ok.grid_forget()
    self.timestamp = self.app.frame_time
```

Joonis 5.13 AddPersonEvent konstruktor

Process meetod (Joonis 5.14) tühistab sündmuse, kui inimese nägu on juba tuvastatud või kui kellegi näo polnud juba 5 sekundit kaamera ees. Process meetod teeb tööd ainult pildi tegemise ajal iga 500 ms tagant.

```
def process(self, timestamp: int):
    if self.app.recognition_event.name != '' or timestamp - self.app.recognition_event.last_person_timestamp > 5000:
        self.app.cancel_event()
    elif self.state == FaceAddState.TAKE_PICTURE:
        if timestamp - self.timestamp >= 500:
```

Joonis 5.14 AddPersonEvent process algus

Algusel ta küsib face_recognitioni poolt näode piirkonnad nende emotsioonidega. (Joonis 5.15) Mudeliks on kasutusel OpenCV, kuna ta ei tuvasta kallutatud näod.

```
self.timestamp = timestamp
result = self.app.face_recognition.guess_emotion(self.app.last_frame, 'opencv')
found = False
```

Joonis 5.15 Näode emotsioonidega hankimine

Siis ta otsib tulemustest näo, mis oleks õiges piirkonnas ja ilma emotsioonideta. (Joonis 5.16)

```
for r in result:
    emotion = r['dominant_emotion']
    region = r['region']
    if emotion == 'neutral' and 0.26 * self.app.cam_width <= region['x'] <= 0.35 * self.app.cam_width \
            and 0.16 * self.app.cam_height <= region['y'] <= 0.35 * self.app.cam_height:
```

Joonis 5.16 Õige näo otsimine

Siis toimub kontroll, et see nägu poleks juba lisatud, sest kui kaamera ees on kaks inimest, siis juba salvestatud inimene võib siia sattuda. (Joonis 5.17)

```
face = self.app.face_recognition.get_face_image(self.app.last_frame, region)
name, id = self.app.face_recognition.recognize_person(face, 'ssd')
if name != '':
    self.app.show_top_text(self.app.text.get_text_from_db('already_in_the_memory'), 3000, 50, 'red')
    break
```

Joonis 5.17 Kontroll et see nägu poleks varem lisatud

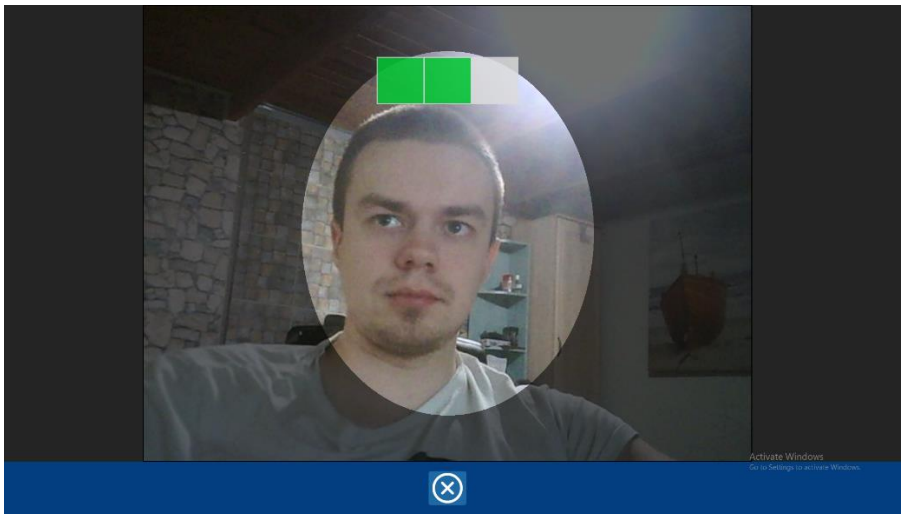
Selleks et näo oleks salvestatud, see peab olema 3 korda leitud. See on tehtud selleks, et pilt oleks tehtud hoolikalt. Siin on kood (Joonis 5.18), mis toimub kui näo on juba vähemalt üks kord leitud. Kui ta polnud salvestatud siis teda salvestatakse face muutujasse. Kui ta olnud salvestatud, siis seda sobilikkust kontrollitakse verify meetodiga, ja kui tulemus on hea, siis kasutaja saab sisestada oma nimi. Ette tuleb klaviatuur ja ilmub ok nupp. Klaviatuuri vajutamise funktsiooniks on määratud enter_key meetod. Samuti näo leidmise etappid on kaamerapildile lisatud progressiriba.

```
if self.found_once:
    if self.face_saved:
        verification_result = self.app.face_recognition.verify(self.app.last_frame,
                                                               face)
        self.timestamp = get_time_ms()
        self.face_saved = False
        self.found_once = False
        if verification_result['verified'] and verification_result['distance'] < 0.1:
            self.app.progress_bar(3, 3)
            self.state = FaceAddState.ADD_NAME
            self.app.face_oval_image_visible = False
            self.app.keyboard.visible = True
            self.app.keyboard.callback = self.enter_key
            self.app.btn_ok.grid(row=0, column=1, padx=BTN_PAD_X, pady=BTN_PAD_Y)
            self.app.show_top_text(f'{self.app.text.get_text_from_db("whats_your_name").capitalize()}?', 5000)
    else:
        self.app.progress_bar(3, 2)
        self.face = face
        self.face_saved = True
        self.timestamp = get_time_ms()
```

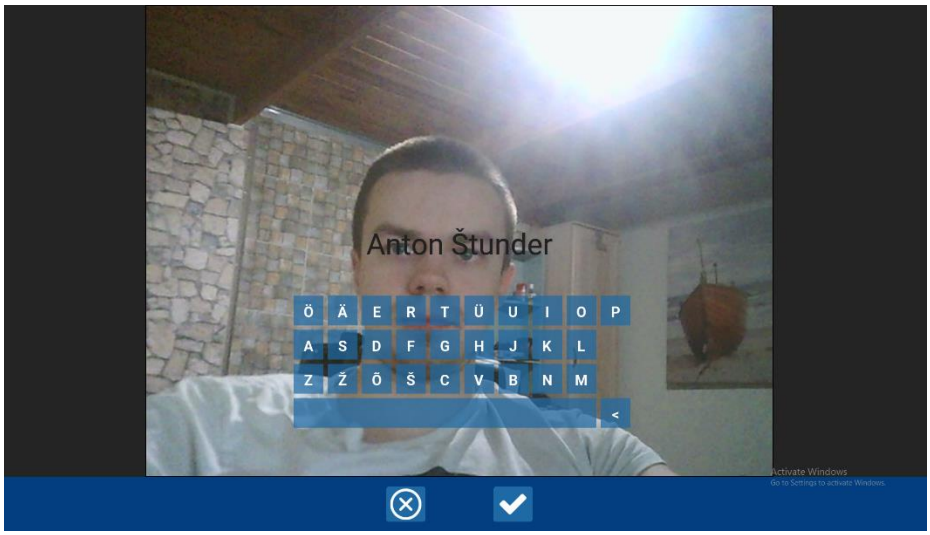
Joonis 5.18 Näo pildi tegemine

```
            else:
                self.app.progress_bar(3, 1)
            found = True
self.found_once = found
if not found:
    self.face_saved = False
```

Joonis 5.19 Näo pildi kontroll



Joonis 5.20 Pildistamine

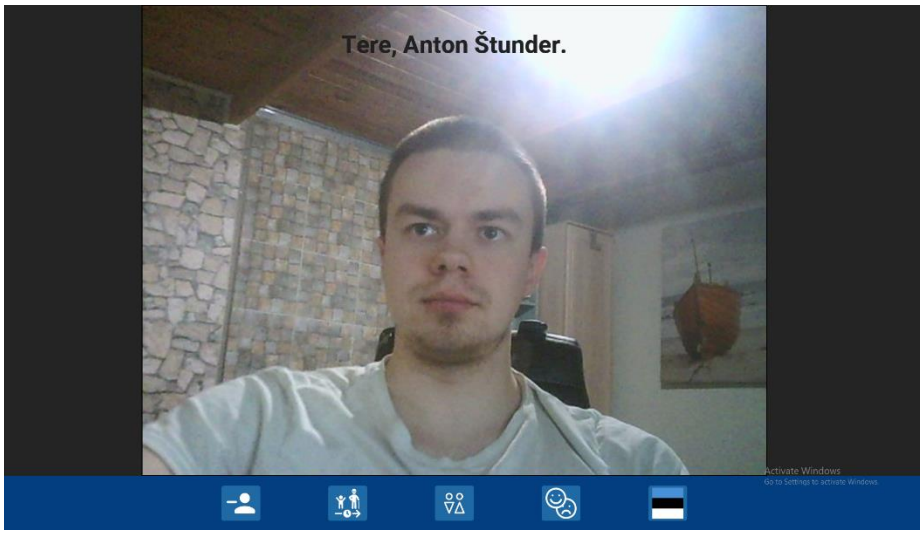


Joonis 5.21 Nime sisestamine

Confirm meetod (Joonis 5.22) lisab salvestatud näo pildiga ja nimega inimest.

```
def confirm(self):
    if self.state == FaceAddState.ADD_NAME and len(self.name) >= 3:
        self.app.face_recognition.add_person(self.name, self.face)
        self.app.cancel_event()
```

Joonis 5.22 Imimese lisamine



Joonis 5.23 Inimene on lisatud

Cancel meetod muudab kõik tagasi ja enter_key muudab name muutuja.

5.2.4 DeletePersonEvent

Konstruktor salvestab praeguse kasutaja id ja küsib tema käest kas ta on kindel, et tahab ennast eemaldada. (Joonis 5.24)

```
class DeletePersonEvent(AppEvent):
    def __init__(self, app: App):
        super().__init__(app)
        self.id = self.app.recognition_event.id
        self.app.main_footer.pack_forget()
        self.app.ok_cancel_footer.pack(pady=(10, 10))
        self.canceled = False
        self.app.show_top_text(
            f'{self.app.recognition_event.name}, {self.app.text.get_text_from_db("are_you_sure_delete")}?', 5000)
```

Joonis 5.24 DeletePersonEvent konstruktor

Process meetod kontrollib, et kamera ees oleks ikka sama inimene. (Joonis 5.25)

```
def process(self, timestamp: int):
    if self.app.recognition_event.id != self.id:
        self.app.cancel_event()
```

Joonis 5.25 DeletePersonEvent on tühistatud, kui inimene pole see, kes vajutas eemaldamise nupu

Confirm meetod kutsub delete_person meetodi, et eemaldata inimest andmebaasist. (Joonis 5.26)

```
def confirm(self):
    self.app.recognition_event.name = ''
    self.app.recognition_event.id = '0'
    self.cancel()
    self.app.face_recognition.delete_person(self.id)
    self.app.cancel_event()
```

Joonis 5.26 Inimese eemaldamine

Cancel meetodis on kontroll, et see poleks veel täidetud, kuna confirm meetodi käivitamises, cancel meetod on kutsutud kaks korda: otseselt, ja app.cancel_event kaudu. (Joonis 5.27)

```
def cancel(self):
    if not self.canceled:
        self.app.main_footer.pack(pady=(10, 10))
        self.app.ok_cancel_footer.pack_forget()
        self.canceled = True
```

Joonis 5.27 DeletePersonEvent cancel

5.2.5 SmileEvent ja AngryEvent

Need on väga sarnased sündmused. Need küsivad kasutaja käest vastava näo emotsiooni näidata ja reageerivad sellele. Sellepärast selles peatükis on vaadeldud ainult üks neist, nimelt SmileEvent.

Konstruktor salvestab sündmuse alguse ja näitab inimesele teksti, mis pakkub talle naeratada. (Joonis 5.28)

```
class SmileEvent(AppEvent):
    def __init__(self, app: App, timestamp):
        super().__init__(app)
        self.timestamp = timestamp
        self.started = timestamp
        if self.app.recognition_event.name != '':
            self.app.show_top_text(
                f'{self.app.recognition_event.name}, {self.app.text.get_text_from_db("why_so_sad_smile")}.', 5000)
        else:
            self.app.show_top_text(f'{self.app.text.get_text_from_db("why_so_sad_smile").capitalize()}.', 5000)
```

Joonis 5.28 SmileEvent konstruktor

Process event lõpetab sündmuse, kui kaamera ees pole kedagi või on möödunud 10 sekundit. Iga 500 ms tagant ta vaatab, kas keegi naeratab. Kui keegi naeratab, siis ta reageerib sellele tekstiga ja illustratsiooniga. Kui sündmus lõpeb, siis ta kutsub cancel_event ja saadab argumendina aega, mis on 10 sekundit hiljem kui toimumise aeg. Selline meetodi käivitamine mitte ainult nullindab praeguse sündmuse, vaid ka muudab kunstlikult viimase sündmuse toimumise aega. (Jonis 5.29)

```
def process(self, timestamp: int):
    if self.app.recognition_event.people_n == 0:
        self.app.cancel_event()
    if timestamp - self.started >= 10000:
        self.app.cancel_event(timestamp + 10000)
    elif timestamp - self.timestamp >= 500:
        self.timestamp = timestamp
        result = self.app.face_recognition.guess_emotion(self.app.last_frame)
        success = False
        for r in result:
            emotion = r['dominant_emotion']
            region = r['region']
            if emotion == 'happy':
                success = True
                self.app.add_facial_illustration(region, self.app.images.happy)
        if success:
            self.app.show_top_text(self.app.text.get_text_from_db("why_so_sad_smile_ok"), 5000)
            self.app.cancel_event(timestamp + 10000)
```

Joonis 5.29 SmileEvent process

Need sündmused saavad alguse App objektis siis, kui rakenduse meelest 8 sekundi jooksul midagi pole toimunud. Need sündmused on loodud try_random_event meetodi (Joonis 5.30) abil.

```
def try_random_event(self):
    if self.current_event is None and self.frame_time - self.last_event_time >= 8000:
        from events.SmileEvent import SmileEvent
        from events.AngryEvent import AngryEvent
        rand = random.randint(0, 2)
        if rand == 0:
            if self.last_random_event != 'smile':
                self.current_event = SmileEvent(self, self.frame_time)
                self.last_random_event = 'smile'
        elif rand == 1:
            if self.last_random_event != 'angry':
                self.current_event = AngryEvent(self, self.frame_time)
                self.last_random_event = 'angry'
        else:
            self.last_event_time = self.frame_time
```

Joonis 5.30 Meetod, mis alustab juhusliku sündmuse

KOKKUVÕTE

Loodud rakendus suudab tuvastada inimest tema näo järgi ja teda tervitada. Kasutaja saab end rakendusse lihtsalt ja mugavalt lisada ning samuti sealt eemaldada. Lisaks sellele saab kasutaja paluda rakendusel näo põhjal ära arvata tema vanust, sugu või emotsiooni. Rakenduse kasutamiseks pole vaja end lisada, kuid sel juhul ei pöördu rakendus kasutaja poole nimepidi, vaid suhtub temasse võõrana. Autor lõi rakenduse kasutajasõbraliku liidese ja huvitava funktsionaalsusega.

Rakenduse loomise ajal tutvus autor erinevate teekidega nii näotuvastuse kui ka graafilise kasutajaliidese loomise valdkonnas. Autor arendas oma programmeerimisoskusi ja omandas väärtusliku praktilise kogemuse.

Kirjaliku töö saab jaotada neljaks osaks. Esimene osa on teoreetiline, kus autor vaatab üle arvutinägemise ja näotuvastuse teema ning uurib erinevaid selle valdkonna teeke. Projekti elluviimiseks valib autor teegi DeepFace. Pärast valiku tegemist uurib autor erinevaid DeepFace poolt pakutavaid näotuvastusmudeleid. Uuringu tulemusena selgub, et SSD lähenemine sobib teistest paremini. Järgmises osas kirjutab autor näotuvastuse realiseerimisest rakenduses. Kolmandas osas kirjeldab autor rakenduse graafilise osa teostamist. Viimases osas käsitletakse ülejäänud komponente.

SUMMARY

The topic of the thesis is "Creating an application for age, gender and emotion detection from real-time facial images" and the author's name is Anton Štunder.

The aim of the project is to create an interactive and engaging app. This project can raise interest in the field and also entertain users to some extent. Also, the process of creating the project raises the skill level of the author and provides practical experience.

To create this app, several tasks were have to be completed. First of all, the development of the user interface. After that it was necessary to develop facial recognition and other computer vision functionality. Then it had to be connected together and application had to be made multilingual.

An app was created that can recognise a person by their face and greet them. A person can quickly and easily add and remove themselves from the app. In addition, the user can let the app guess their facial age, gender and emotion. In order to use the app, you do not have to add yourself, but then the app cannot interact with the person by their name. The author created an app that has a user-friendly interface and interesting functionality.

While creating the app, the author familiarized himself with different technologies for facial recognition as well as for creating the user interface. The author developed his programming skills and gained useful experience.

The written work can be divided into four parts. The first is theoretical, where the author gives an overview of computer vision and face recognition and then takes a look at different libraries. For the realisation of the project, he chose the DeepFace library. After making the selection, the author conducts a study of the different face detection models proposed by DeepFace and finds that ssd worked better than the others. After the theoretical part, the author gives an overview of the realization of the face detection application. Then, he writes about the graphical part of the application and in the end he gives an overview about remaining components of the application.

KASUTATUD KIRJANDUSE LOETELU

1) What Is Computer Vision: Applications, Benefits and How to Learn It [Online] https://www.simplilearn.com/computer-vision-article (10.08.2023)
2) Face detection definition [Online] https://www.techtarget.com/searchenterpriseai/definition/face-detection (01.04.2023)
3) What is facial recognition and how does it work? [Online] https://us.norton.com/blog/iot/how-facial-recognition-software-works (21.07.2023)
4) ML Kit [Online] https://developers.google.com/ml-kit (26.12.2023)
5) OpenCV [Online] https://opencv.org/ (10.12.2023)
6) DeepFace [Online] https://github.com/serengil/deepface (08.12.2023)
7) Facial-Emotion-Recognition-using-OpenCV-and-Deepface [Online] https://github.com/manish-9245/Facial-Emotion-Recognition-using-OpenCV-and-Deepface (18.06.2023)
8) How to use alpha_composite in pillow? [Online] https://stackoverflow.com/questions/64321963/how-to-use-alpha-composite-in-pillow (12.10.2020)
9) 6_8.jpg https://www.linkedin.com/company/all-about-people/
10) 6_8.jpg https://www.linkedin.com/company/all-about-people/
11) 2_7.jpg https://www.weforum.org/agenda/2021/08/heres-what-young-people-think-is-key-to-a-long-and-fulfilled-life/
12) 3_8.jpg https://www.quickanddirtytips.com/articles/people-or-persons-2/
13) add_person.png ja delete_person.png https://iconscout.com/free-icon/add-person-1780869
14) age.png https://thenounproject.com/icon/age-2941875/
15) gender.png https://iconscout.com/free-icon/gender-2188786
16) emotion.png https://www.freepik.com/icon/emotion_4814259
17) flag-ee.png https://vectorflags.com/estonia/ee-square-01?ref=home-flag
18) cancel.png https://www.freepik.com/icon/close_10289692
19) ok.png https://www.freepik.com/icon/close_10289692
20) angry.png https://1000logos.net/angry-face-emoji/
21) disgust.png https://www.vectorstock.com/royalty-free-vector/vomiting-emoji-vector-39310506
22) fear.png https://ru.pinterest.com/pin/288230444899277933/
23) happy.png https://bethfreeman.coach/where-does-happy-come-from/
24) neutral.png https://emojiisland.com/products/neutral-face-emoji-icon
25) sad.png https://www.symbols-n-emoticons.com/2018/05/worrisome.html

26) surprise.png https://www.istockphoto.com/vector/cute-surprised-emoticon-on-white-background-gm840934838-137085199
27) female.png https://www.freepik.com/icon/woman_3001801
28) male.png https://www.freepik.com/icon/man_3001764