Programsko rješenje za kontrolu računala gestama ruku

Jakuš-Mejarec, Josip

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Josip Jakuš-Mejarec

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Josip Jakuš-Mejarec

Zagreb, 2024.

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

Velike zahvale mentoru dr. sc. Tomislav Stipančić, dipl. ing. Tomislavu Stipančiću na pristupačnosti i pruženoj pomoći tijekom izrade rada.

Josip Jakuš-Mejarec

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Josip Jakuš Mejarec** JMBAG: **0035239645**

Naslov rada na hrvatskom jeziku: **Programsko rješenje za kontrolu računala gestama ruku**

Naslov rada na engleskom jeziku: **Software solution for controlling a computer with hand gestures**

Opis zadatka:

U rada je potrebno razviti programsko rješenje koje omogućuje upravljanje funkcijama računala putem gesti ruku (na primjer, promjena glasnoće ili kretanje kroz web stranicu). Komponente sustava trebaju uključivati kameru za akviziciju slike, računalo te razvijenu programsku podršku. Rad treba temeljiti na programskom jeziku Python te biblioteci MediaPipe za praćenje ruku i identifikaciju gesti.

Rješenje treba uključivati:

- računalni modul temeljen na MediaPipe programskoj biblioteci za analizu podataka o položaju ruke za identifikaciju gesti u realnom vremenu
- skup od barem dvije funkcije na računalu kojima se želi postići upravljanje koristeći geste ruku
- računalni modul za identifikaciju gesti i upravljanje definiranim funkcijama na temelju posebnih obilježja za ruke čije su definicije dostupne u sklopu MediaPipe biblioteke (engl. hand landmarks)
- testiranje sustava za provjeru uspješnosti izvedbi uključujući prepoznavanje gesti te izvršavanje ciljanih funkcija na računalu.

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

Zadatak zadan:
24. 4. 2024.

Zadatak zadao:
izv. prof. dr. sc. Tomislav Stipančić

Datum predaje rada:
2. rok (izvanredni): 11. 7. 2024.
3. rok: 19. i 20. 9. 2024.

Predviđeni datumi obrane:
2. rok (izvanredni): 15. 7. 2024.
3. rok: 23. 9. – 27. 9. 2024.

Predsjednik Povjerenstva:
izv. prof. dr. sc. Petar Ćurković

SADRŽAJ

POPIS SLIKA

POPIS KRATICA

Kratica	Puni naziv
CNN	konvolucijska neuronska mreža (convolutional neural network))
RGB	Crvena – Zelena – Plava (Red – Green - Blue)
BRG	Plava – Zelena – Crvena (Blue – Green - Red)
Dpi	točke po inču (dots per inch)
RNNs	rekurentne neuronske mreže (Recurrent Neural Networks)

SAŽETAK

Razvoj umjetne inteligencije i računalnog vida omogućava inovativne načine upravljanja računalom. U ovom radu koristi se programski jezik Python u kombinaciji s bibliotekama MediaPipe i TensorFlow kako bi se implementirala funkcionalnost miša i tipkovnice. Umjesto tradicionalnih uređaja, funkcije se ostvaruju pokretima ruku koji se prepoznaju pomoću kamere. Također izrađuje se model za klasifikaciju poza ruke, gdje svaka jedinstvena poza predstavlja određeno slovo. Cilj ovog rada je zamijeniti konvencionalne metode upravljanja računalom inovativnim pristupom temeljenim na gestama.

Ključne riječi: Python, MediaPipe, TensorFlow, umjetna inteligencija

SUMMARY

The development of artificial intelligence and computer vision enables innovative ways of controlling a computer. In this paper, the Python programming language is used in combination with the MediaPipe and TensorFlow libraries to implement mouse and keyboard functionalities. Instead of traditional devices, these functions are achieved through hand movements recognized by a camera. A model for hand pose classification is also created, where each unique pose represents a specific letter. The goal of this paper is to replace conventional computer control methods with an innovative gesture-based approach.

Key words: Python, MediaPipe, TensorFlow, artificial intelligence

1. UVOD

Tradicionalne metode upravljanja računalom, poput tipkovnice i miša, još uvijek su standard kod interakcije s računalom. Međutim, razvojem tehnologije pojavljuju se novi načini upravljanja računalom. U sklopu ovog rada, razvija se programsko rješenje koje omogućuje zamjenu funkcija miša jednostavnim pokretima prstiju. Neće samo imitirati pokrete miša, nego će omogućiti izvođenje bilo koje funkcije koju računalo prepoznaje.

Geste ruka služit će kao signali u kodu preko kojih će se aktivirati funkcije miša i tipkovnice. Za detekciju pokreta prstiju koristit će se biblioteka MediaPipe, koja će biti opisana u narednim poglavljima. Sljedeća funkcionalnost koja će se implementirati je podešavanje glasnoće računala koristeći biblioteku PyCaw. Osim toga, planira se mogućnost prepoznavanja znakova američkog znakovnog jezika (ASL). ASL koristi različite geste ruku kako bi predstavio slova abecede, a svaka gesta ima svoje specifično značenje.

Za postizanje ove funkcionalnosti koriste se metode nadziranog učenja. To uključuje prikupljanje velikog broja slika različitih gesta ruku, označavanje tih slika prema odgovarajućim slovima te korištenje Teachable Machine za dobivanje modela koji može detektirati te položaje ruku. Ovaj sustav omogućuje prepoznavanje znakova ASL-a i njihov unos u računalo kada se prepoznaje odgovarajuća gesta ruke.

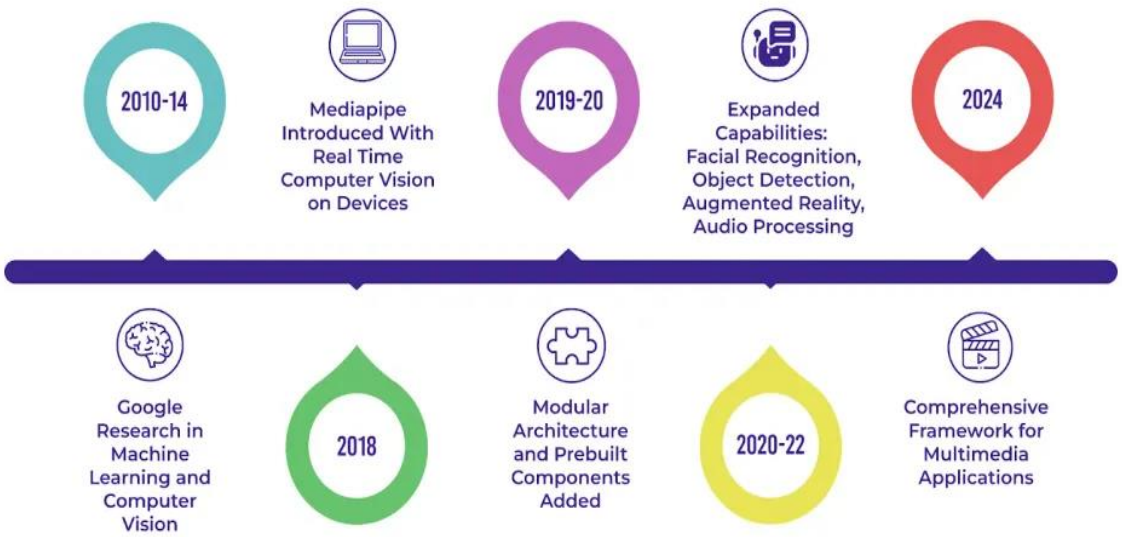
Ovaj rad doprinosi razumijevanju kako moderne tehnologije poput MediaPipea mogu transformirati način na koji ljudi komuniciraju s računalima te otvara nove mogućnosti za daljnje istraživanje i razvoj u području intuitivnih korisničkih sučelja.

2. KORIŠTENE BIBLIOTEKE

2.1 MediaPipe

2.1.1 Povijest

MediaPipe svoje korijene vuče iz ranih 2010-ih godina kada je Google započeo rad na napretku područja strojnog učenja i računalnog vida. Inicijalni razvoj i primjena ove tehnologije počeli su oko 2012. godine, kada je korištena za analizu videosadržaj na YouTubeu, što je pripremilo teren za kasniji razvoj i usavršavanje MediaPipea koji je rješenje za obradu multimedijskih podataka u stvarnom vremenu. Od 2018. godine, MediaPipe počeo je rješavati izazove povezane s primjenom složenih modela računalnog vida na uređajima poput pametnih telefona i malih računara. Porastom potreba za brzim i učinkovitim obradama multimedijskog sadržaja do 2020. godine, MediaPipe je prilagođen i unaprijeđen kako bi zadovoljio te zahtjeve. Danas se MediaPipe etablirao kao vodeći okvir za razvoj inovativnih aplikacija koje omogućuju naprednu obradu slike i videa u stvarnom vremenu na različitim uređajima.



Slika 1. Vremenska linija MediaPipea [1]

2.1.2 Mediapipe Solutions

MediaPipe Solutions pruža skup biblioteka za primjenu tehnike umjetne inteligencije i strojnog učenja u mnogim aplikacijama. Biblioteke se mogu se odmah integrirati u aplikacije, prilagoditi potrebama i koristiti na različitim platformama [2].

Glavne biblioteke za funkcionalnost MediaPipea su [2]:

- MediaPipe Tasks - API-jevi i biblioteke za implementaciju rješenja
- MediaPipe Models - unaprijed trenirani modeli za korištenje pojedinog rješenja

Alati koji omogućuju prilagodbu i evaluaciju rješenja su [2]:

- MediaPipe Model Maker – prilagodi modele za rješenja nad podacima
- MediaPipe Studio – vizualizacija, procjena i testiranje rješenja u pregledniku

MediaPipe Solutions dostupan je na više platforma, ali ne pruža sva rješenja na svakoj platformi. Svako rješenje obuhvaća nekoliko modela i može se prilagoditi specifičnim potrebama. Kako bi se koristilo određeno rješenje, važno je provjeriti je li podržan na platformi na kojoj se radi [2].

Solution	Android	Web	Python	iOS	Customize model
LLM Inference API	●	●		●	●
Object detection	●	●	●	●	●
Image classification	●	●	●	●	●
Image segmentation	●	●	●		
Interactive segmentation	●	●	●		
Hand landmark detection	●	●	●	●	
Gesture recognition	●	●	●	●	●
Image embedding	●	●	●		
Face detection	●	●	●	●	
Face landmark detection	●	●	●		
Face stylization	●	●	●		●
Pose landmark detection	●	●	●		
Image generation	●				●
Text classification	●	●	●	●	●
Text embedding	●	●	●		
Language detector	●	●	●		
Audio classification	●	●	●		

Slika 2. Kompatibilnost s jezicima [2]

2.1.3 MediPipe Hand Landmark

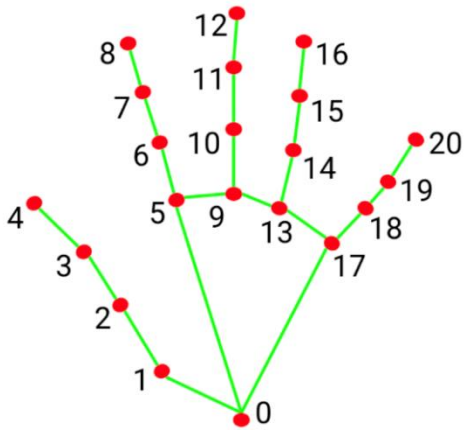
MediaPipe Hand Landmarker napredan je alat koji omogućuje prepoznavanje i praćenje zglobova ruku na danoj slici. Ova tehnologija sadrži velik broj mogućih aplikacija u budućoj komunikaciji s računalom poput proširene stvarnosti, virtualne stvarnosti te kao u našem radu sa kontrolom računala preko gesta ruku.[3]

MediaPipe Hand Landmarker je alat koji prima kao ulaz sliku ili video te obrađuje sadržaj okvir po okvir (engl. *frame by frame*). Kao izlaz, alat može prepoznati i razlikovati lijevu i desnu ruku, pri čemu daje postotak sigurnosti u pogledu toga je li detektirana lijeva ili desna ruka. Također, alat vraća indeks koji odgovara lijevoj ili desnoj ruci [3].

Dodatno, MediaPipe Hand Landmarker omogućuje dobivanje koordinata zglobova ruke unutar slike ili videa. Obično se koriste samo x i y koordinate, ali z koordinata može biti korisna za specifične aplikacije, poput analize relativnog pomaka iz čega bi se mogla izračunati brzina pokreta ruke .

Hand landmarker model detektira dvadeset jednu ključnu točku na ruci. Model je treniran na 30.000 stvarnih slika i nekoliko sintetiziranih modela na različitim pozadinama [3]

Hand landmarker model sastoji se od modela za detekciju dlana i modela za detekciju ključnih točaka na ruci. Model za detekciju dlana locira ruku na slici, dok model za detekciju ključnih točaka prepoznaje zglobove i glavne točke na ruci. S obzirom na to da je detekcija dlana vremenski zahtjevna, model koristi područje gdje se ruka nalazila u prethodnoj slici kako bi smanjio područje pretraživanja u narednim slikama [3].



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Slika 3. HandLandmarkers [3]

2.2 OpenCv

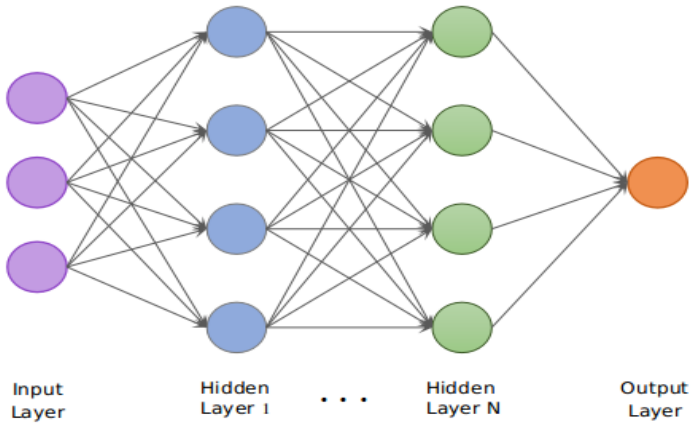
OpenCV (engl. *Open Source Computer Vision Library*) je open-source biblioteka za računalni vid i strojno učenje. Sadrži više od 2500 optimiziranih algoritama za zadatke računalnog vida, uključujući prepoznavanje lica, prepoznavanje objekata, praćenja pokreta i mnoge druge. Podržava više programskih jezika: C++, Python, Java i MATLAB te je kompatibilan s operativnim sustavima: Windows, Linux, Android i MacOS [4].

OpenCV ima široku primjenu u industriji i istraživanju te ga koriste velike tvrtke poput Googlea, Microsofta i IBM-a, kao i brojni startupovi i razne institucije. Primjeri njegove primjene uključuju spajanje slika prikaza ulica, detekciju osoba kod nadzornih sustava, pomoć kod kretanja u robotici, pregledavanje naljepnica na proizvodima u tvornicama [4].

OpenCV, sudjelovanjem zajednice potiče razvoj open-source softvera. Programeri mogu doprinijeti pregledom koda, ispravljanjem bugova ili optimizacijom biblioteke za različite platforme. Tvrtke mogu sponzorirati projekt ili pružiti podršku za infrastrukturu [4].

2.3 TensorFlow

TensorFlow vodeća je open-source biblioteka za strojno učenje, koju često koriste programeri, edukatori i znanstvenici za podatke. Razvio ju je Google Brain tim i omogućuje kreiranje i izvođenje modela strojnog učenja koristeći grafove toka podataka. Čvorovi predstavljaju matematičke operacije, dok rubovi grafova predstavljaju višedimenzionalne podatkovne nizove (engl. *tensors*) koji teku čvorovima [5].



Slika 4. Primjer neuronske mreže [6]

TensorFlow uključuje mnoge korisne značajke. Na primjer, TensorBoard omogućuje vizualno praćenje procesa obuke, temeljnog računalnog grafa i matrice za svrhe ispravljanja grešaka i procjene performansi modela. Keras je visoko apstrahirani API koji se koristi s TensorFlowom

i pruža pojednostavljeni API za izgradnju modela za uobičajene slučajeve upotrebe, omogućujući bržu pretvorbu ideja u rezultate [5].

TensorFlow pruža brojne mogućnosti za razvoj modela, omogućujući brzo izrađivanje inovativnih ideja i algoritama. Programeri mogu koristiti TensorFlow na širokom rasponu hardverskih platformi i operativnih okruženja, s lakoćom u implementaciji modela zahvaljujući interoperabilnosti. Google je snažno podržao TensorFlow, što doprinosi njegovom brzom razvoju i stalnom poboljšanju [5].

2.4 NumPy

NumPy je open-source Python biblioteka široko korištena u znanstvenim i inženjerskim projektima. Nudi strukturu višedimenzionalnog niza (engl. *array*) podataka iste vrste, tzv. N-dimenzionalni niz te sadrži veliki broj funkcija za obradu nizova podataka [7].

Iako Python ima ugrađene liste koje mogu sadržavati bilo koji tip podataka i koje su vrlo brze za operacije manjeg broja podataka, korištenje NumPy niza omogućuje smanjenje potrošnje memorije i pruža visoku razinu sintakse za izvršavanje raznih zadataka obrade podataka. NumPy se posebno ističe kada je potrebno obraditi velike količine homogenih podataka na CPU-u [7].

Zbog svoje učinkovitosti i svestranosti, NumPy je neizostavan alat za sve koji se bave znanstvenim računalnim radom, analizom podataka ili bilo kojom vrstom numeričkih izračuna u Pythonu [7].

2.5 PyAutoGUI

PyAutoGUI omogućuje Python-skriptama kontrolu nad mišem i tipkovnicom. Dizajniran je s jednostavnim API-jem i podržava rad na različitim operativnim sustavima poput Windowsa, macOS-a i Linuxa. Prikladan je za automatizaciju repetitivnih zadataka, čime se značajno povećava produktivnost [8].

PyAutoGUI nudi jednostavan, ali moćan način za automatizaciju svakodnevnih zadataka, pružajući korisnicima pristup funkcijama računala bez potrebe za ručnim korištenjem tipkovnice ili miša [8].

3. OBRADA SLIKA I VIDEOZAPISA

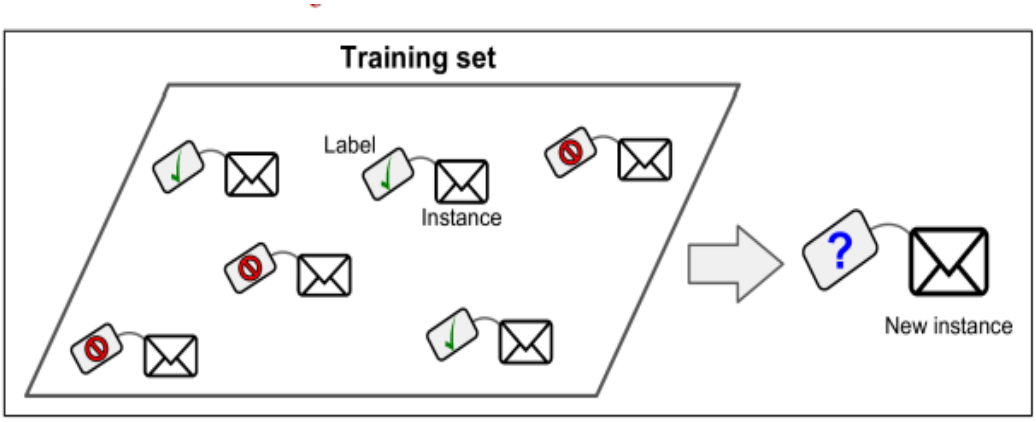
3.1 Strojno učenje

Strojno učenje je znanost o programiranju računala usredotočeno na imitiranje ljudskog učenja. Postoji nekoliko različitih definicija, a Arthur Samuel 1959. strojno učenje definira: "[Strojno učenje je] područje proučavanja koje omogućuje računalima da uče bez eksplicitnog programiranja." [9].

Najjednostavniji primjer strojnog učenja je filtar neželjene elektronične pošte (engl. *spam filter*), tj. program koji nauči kako sortirati neželjenu elektroničku poštu. Elektronička pošta koja se koristi za učenje naziva se skupom za treniranje [9].

Strojno učenje se obično dijeli na nekoliko glavnih vrsta, uključujući:

1. Nadzirano učenje (engl. *supervised learning*): Ova metoda koristi označene podatke za treniranje modela. Model se "uči" na temelju ulazno-izlaznih parova podataka kako bi mogao predviđati ili klasificirati nove, neoznačene podatke. Primjeri nadziranog učenja uključuju klasifikaciju elektroničke pošte kao spam ili ne-spam te regresiju, gdje se predviđa kontinuirana vrijednost, poput cijene nekretnine [10].



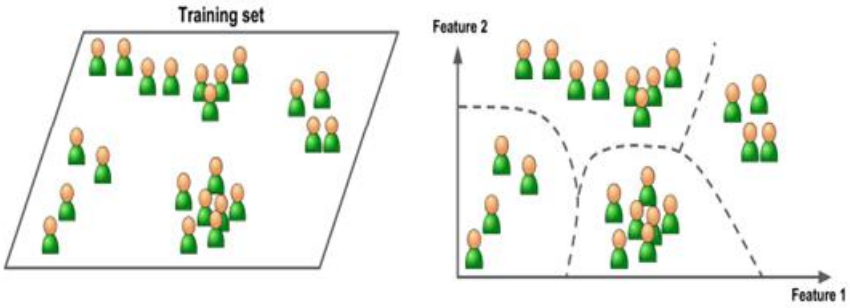
Slika 5. Primjer nadziranog učenja [10]

Evo nekoliko najvažnijih algoritama nadziranog učenja [10]:

- k-Nearest Neighbors (k-NN): Algoritam koji klasificira podatke na temelju blizine njihovih najbližih susjeda u prostoru značajki.
- Linear Regression: Metoda koja modelira odnos između ovisnih i neovisnih varijabli koristeći linearnu funkciju.
- Logistic Regression: Algoritam za binarnu klasifikaciju koji koristi logističku funkciju za predviđanje vjerojatnosti pripadnosti klasi.

- Support Vector Machines (SVMs): Algoritam koji traži optimalnu granicu između klasa u prostorima visoke dimenzionalnosti.
- Decision Trees and Random Forests: Algoritmi koji koriste stablo odluka za donošenje predikcija ili klasifikacija, gdje Random Forests koristi ensemble metodu za poboljšanje točnosti.
- Neural Networks: Modeli koji oponašaju način rada ljudskog mozga i koriste višeslojne neuronske mreže za prepoznavanje složenih obrazaca.

2. Nenadzirano učenje (engl. *unsupervised learning*): Ova metoda koristi neoznačene podatke i traži obrasce ili strukture u podacima. Klasteriranje je jedan od najčešćih metoda u nenadziranom učenju, gdje se podaci grupiraju u skupine prema sličnostima, poput grupiranja korisnika u marketinške segmente [10].

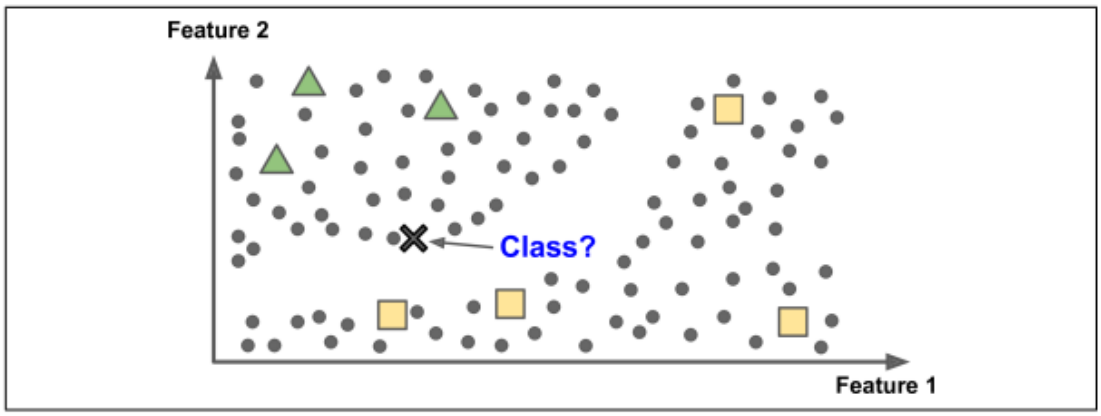


Slika 6. Primjer nenadziranog učenja [10]

Nekoliko ključnih algoritama u nenadziranom učenju uključuju:

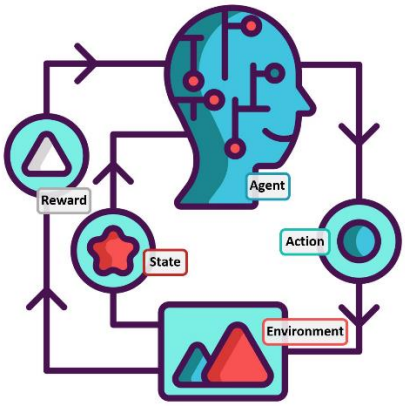
- Klasteriranje (engl. *Clustering*): Grupira podatke u klastere prema sličnostima.
- Učenje pravila asocijacije (engl. *Association Rule Learning*): Otkriva veze između varijabli u podacima.
- Vizualizacija i smanjenje dimenzionalnosti (engl. *Visualization and Dimensionality Reduction*): Prikazuje podatke u smanjenom prostoru za lakše razumijevanje.
- Detekcija anomalija i noviteta (engl. *Anomaly Detection and Novelty Detection*): Prepoznaje neuobičajene obrasce u podacima.

3. Polu-nadzirano učenje (engl. *semi-supervised learning*): Ova metoda koristi kombinaciju označenih i neoznačenih podataka. Ovo je korisno kada je označavanje podataka skupo ili zahtjevno. Polu-nadzirano učenje može poboljšati performanse modela u usporedbi s potpuno nadziranim učenjem kada je količina označenih podataka ograničena [10].



Slika 7. Primjer polu-nadziranog učenja [10]

4. Učenje putem pojačanja (engl. *reinforcement learning*): U ovoj metodi, agent uči kako donositi odluke kroz interakciju s okolinom. Agent prima povratne informacije u obliku nagrada ili kazni i prilagođava svoje akcije kako bi maksimizirao ukupnu nagradu. Ova metoda često se koristi u igrama, robotici i optimizaciji [10].



Slika 8. Primjer učenja putem pojačanja [10]

Različite vrste strojnog učenja omogućuju računalnim sustavima da se prilagode i mijenjaju u virtualnim okolinama. Iako ove metode nisu jedine dostupne, one su među najčešće korištenima i najvažnijima u primjeni strojnog učenja.

3.2 Računalni vid

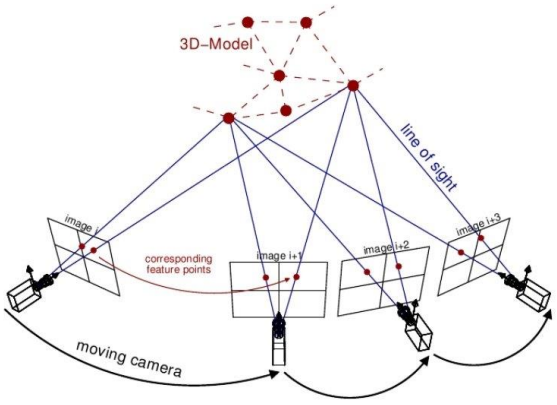
Računalni vid omogućuje računalima da razumiju i interpretiraju vizualne informacije iz svijeta na način usporediv s ljudskom percepcijom. Računalni vid koristi raznovrsne algoritme, matematičke modele i tehnike strojnog učenja kako bi omogućio računalima analizu i razumijevanje vizualnih podataka. Obradom tih podataka računala mogu identificirati objekte, prepoznati uzorke, pa čak i donositi odluke temeljene na vizualnim informacijama [11].

Računalni vid temelji se na sposobnosti računala da obrađuje slike i videozapise te da izvuče korisne informacije [11]. Primjer toga mogu biti:

- Optičko prepoznavanje znakova (engl. *Optical Character Recognition*)

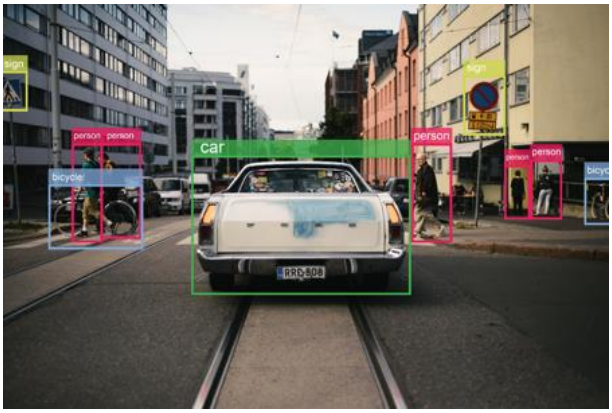
Slika 9. Primjer prepoznavanja znakova [12]

- Izrada 3D modela (engl. *3D model building*)



Slika 10. Primjer izrade 3D modela [13]

- Automobilska sigurnost (engl. *Automotive safety*)



Slika 11. Primjer prepoznavanja automobila [14]

3.3 Teachable Machine

Teachable Machine je mrežni alat s grafičkim korisničkim sučeljem za stvaranje vlastitog modela, bez potrebe za dubokim znanjem strojnog učenja. Razvio ga je Google, a njegova primarna namjena je obrazovanje i uporaba u kreativnim projektima [15].

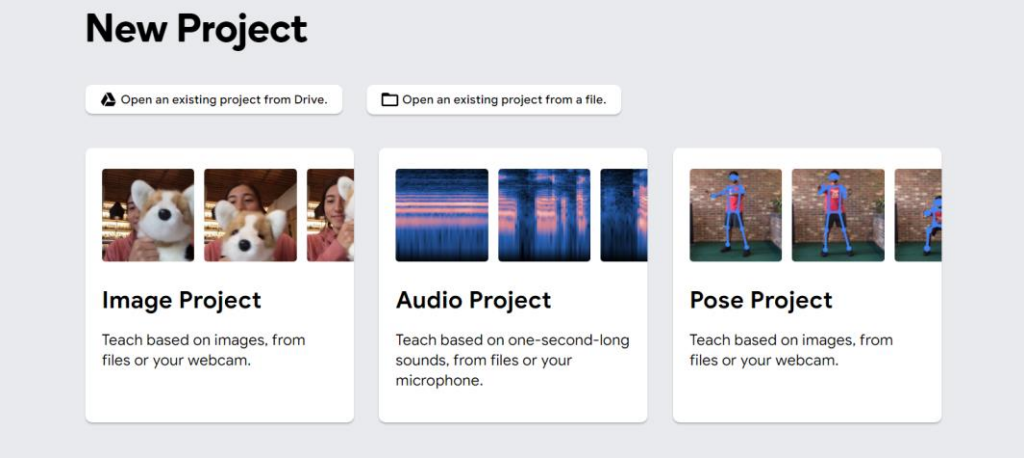
Teachable Machine koristi konvolucijske neuronske mreže (CNNs) za obradu slika, kao i rekurentne neuronske mreže (RNNs) za obradu zvuka i teksta. Takve mreže pokazale su visoku efikasnost u zadacima kao što su prepoznavanje objekata, klasifikaciji zvukova i analizi teksta. Ovaj alat također omogućava korisnicima izvoz (engl. *export*) svojeg modela u različitim formatima, uključujući TensorFlow.js, koji omogućava pokretanje modela direktno u pregledniku ili TensorFlow Lite, koji omogućava izvođenje modela na mobilnim uređajima [15].

3.3.1 Vlastiti model

Ovdje će se govoriti o načinu korištenja alata Teachable Machine za kreiranje vlastitog modela strojnog učenja. Opisat će se koraci koje korisnik treba pratiti kako bi samostalno izradio, obučio i testirao model koristeći ovaj alat. Proces je jednostavan i ne zahtijeva prethodno tehničko znanje, što ga čini pristupačnim širokom spektru korisnika.

Koraci su sljedeći:

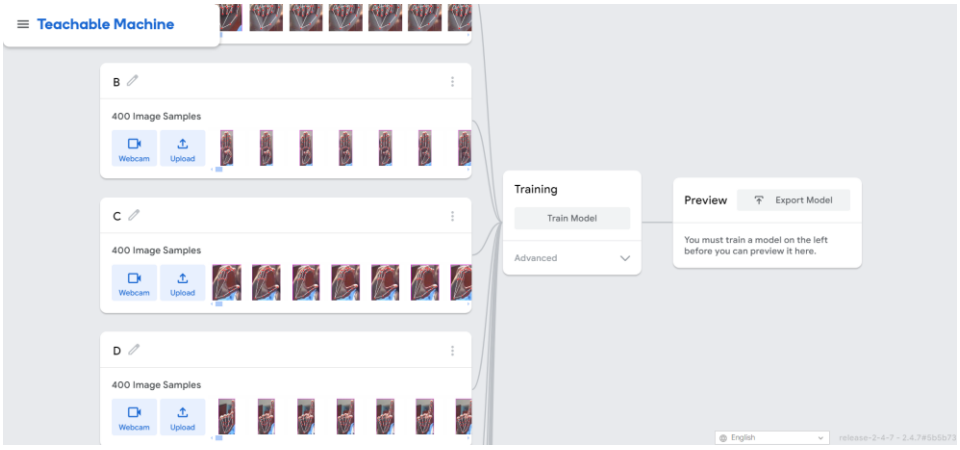
1. Odabir tipa projekta – Na početku potrebno je odabrati vrstu podataka koje će model klasificirati. Teachable Machine može kao ulazne podatke uzimati zvukove, slike ili poze. Za ovaj rad potrebna je klasifikacija slike.



Slika 12. Odabir tipa projekta

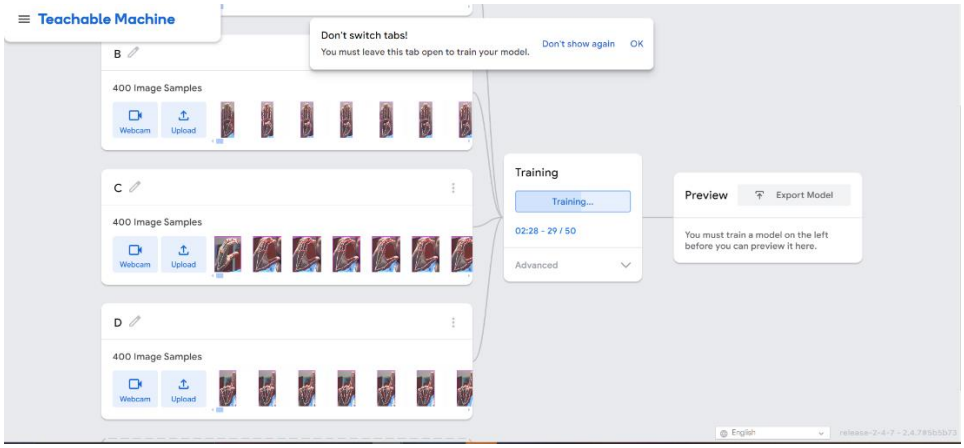
2. Obuka modela – Teachable Machine koristi oblik nadziranog učenja kod kojeg se slika, koja se prenese, klasificira. Kod ovog rada pohranjeno je 400 slika za svaku gestu ruke

koja odgovara određenom slovu. Slike koje se klasificiraju dobe se iz zasebnog koda koji će biti objašnjen u sljedećem poglavlju.

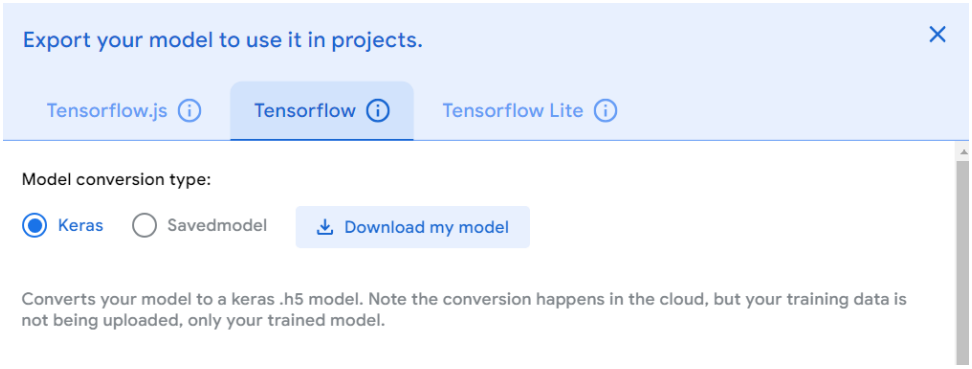


Slika 13. Obuka modela

3. Treniranje i izvoz modela – Kada su sve slike klasificirane, može se započeti trening pritiskom na tipku „Training“. Nakon treniranja moguće je izvesti željeni model. Za ovaj rad potreban je model TensorFlow tipa Keras zbog odabrane biblioteke.



Slika 14. Treniranje modela



Slika 15. Izvod modela

4. IZVEDBA RADA

4.1 Prikupljanje podataka

Ovdje će se detaljno objasniti rad koda koji služi za prikupljanje slika potrebnih za treniranje modela koji prepoznaje geste ruku. Cilj modela je da u glavnom kodu prepoznaje kojem slovu pripada određena gesta ruke.

Prvi korak u kodu je uvođenje korištenih biblioteka. Za ovaj kod potrebne su biblioteke OpenCV kojim se pristupa kameri i pomoću koje se dobiva vrijednosti okvira (engl. *frame*) nakon će ga se obrađuje, prikazuje i sprema slika. Za ovaj rad potrebno je poznavati koordinate ruke i zglobova pa za to se koristi biblioteka MediaPipe. Također koristi se i biblioteka Time koja će služiti samo za opredjeljivanje jedinstvene vrijednosti imena slikama koje se spremaju. Za spremanje slika također je potrebna biblioteka PathLib. Posljednja korištena biblioteka je NumPy.

```
import time
import mediapipe as mp
import cv2
from pathlib import Path

import numpy as np
```

Slika 16. Uvođenje biblioteka

Sljedeći glavni korak je inicijalizacija varijabli s kojima će se raditi. Definiraju se svi brojači, koji su označeni prefiksom „*c_*“. Također, definira se folder u koji će se spremiti snimljene slike. Zatim, stvori se lista slova od A do Z pomoću funkcije *map*, koja svaki broj od 65 do 90 pretvara u znak (engl. *character*) koji se određuje prema ASCII kodu (brojevi od 65 do 90 odgovaraju velikim slovima u ASCII tabeli).

Potrebno je još inicijalizirati varijable koje instanciraju module iz Mediapipe biblioteke, uključujući *drawing_utils* za vizualizaciju rezultata i *Hands* za detekciju i praćenje ruku.

```
folder = "Data/A"
c_alphabet = 0
c_frame = 0
alphabet = list(map(chr, range(65, 91)))

mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=2, min_detection_confidence=0.5, min_tracking_confidence=0.5)
```

Slika 17. Inicijalizacija varijabli

Još je potrebno inicijalizirati objekt *cap*, čijim će se metodama pristupiti sadržaju videokamere. Ostatak koda izvodi će se unutar petlje koja će trajati sve dok se ne aktivira funkciju *break*. Svaki put kada se petlja ponovo pokrene, uzima se slika s kamere i sprema se u varijablu *frame*, dok varijabla *ret* označava je li okvir (engl. *frame*) uspješno očitan. MediaPipe kao argument prihvaća samo RGB slike pa se slika mora promijeniti jer je u BGR formatu. Iako u ovom kodu to nije od velike važnosti, slika će se u ovom slučaju preokrenuti da se ponaša kao ogledalo. Na kraju, nakon što je slika prebačena u RGB format i nakon što je obavljeno preokretanje, poziva se funkcija *hands.process* kako bi se slika obradila pomoću Mediapipea da se dobiju željene koordinate.

```
cap = cv2.VideoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, flipCode: 1)
    RGB_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    result = hands.process(RGB_frame)
```

Slika 18. Učitavanje slika

Prvo se provjerava postoje li koordinate koje omogućavaju da se slika analizira. Ako su koordinate dostupne, mogu se zbog vizualnih razloga nacrtati zglobovi ruke na videoprikazu. U ovom kodu, za razliku od glavnog programa, nije potrebno iterirati kroz svaku ruku jer će druga ruka biti potrebna za pritiskanje odgovarajuće tipke na tipkovnici.

Za slike koje se prikupljaju, potrebno je prvo odrediti maksimalne i minimalne x i y koordinate ruke. Nakon toga, treba proširiti područje koje pokriva ruka kako bi se osiguralo da rubovi područja ruke ne budu preblizu prstima. Ovo proširenje omogućava da se ne izgube važni dijelovi slike zbog blizine ruba područja detekcije.

```
if result.multi_hand_landmarks:
        mp_drawing.draw_landmarks(frame, result.multi_hand_landmarks[0], mp_hands.HAND_CONNECTIONS)

        # Extract landmark positions
        lm_list = [(int(lm.x * 640), int(lm.y * 480)) for lm in result.multi_hand_landmarks[0].landmark]
        x_list, y_list = zip(*lm_list)

        xmin, xmax = min(x_list), max(x_list)
        ymin, ymax = min(y_list), max(y_list)
        hand_region = frame[max(0, ymin - 20):min(480, ymax + 20), max(0, xmin - 20):min(640, xmax + 20)]
```

Slika 19. Izdvajanje položaja ključnih točaka

S obzirom na to da se želi da naše klasificirane poze budu čim lakše prepoznatljive, napravi se *white_image* koja će biti samo prazna slika, matematički je predstavljena sa matricom 300x300, sa svim brojevima 255, na koju se stavlja naša slika ruke. S obzirom na to da naša slika ruke ovisi o udaljenosti od kamere i omjeru visine i širine, mora se promijeniti veličina slike ruke. Za promjenu veličine slike potrebno je usporediti širinu i visinu. Ako je visina veća onda se smanjuje širina i obratno. Nakon što se odredi koja se veličina mijenja, veća veličina postaje 300 piksela, dok manja promjeni veličinu pomoću varijable *scale*. Također, potrebno je i osigurati da manja veličina ne prelazi 300 piksela. Na kraju se varijabla *resized* stavlja u varijablu *white_image* pomoću presijecanja (engl. *slicing*). Na kraju se mogu prikazati rezultati.

```
# Prepare the white image
white_image = np.ones( shape: (300, 300, 3), dtype=np.uint8) * 255

h, w = ymax - ymin + 40, xmax - xmin + 40
if h > w:
    scale = 300 / h
    new_w = min(int(scale * w), 300)
    resized = cv2.resize(hand_region,  dsize: (new_w, 300))
    x_offset = (300 - new_w) // 2
    white_image[:, x_offset:x_offset + new_w] = resized
else:
    scale = 300 / w
    new_h = min(int(scale * h), 300)
    resized = cv2.resize(hand_region,  dsize: (300, new_h))
    y_offset = (300 - new_h) // 2
    white_image[y_offset:y_offset + new_h, :] = resized

# Display the results
cv2.imshow( winname: 'White image', white_image)
cv2.imshow( winname: "Camera", frame)
```

Slika 20. Prilagođavanje veličine slike

Ostatak koda fokusira se na spremanje slika i stvaranje potrebnih direktorija za brži način prikupljanja podataka. Prvo se ispituju brojači koji su definirani na početku. Ako su oba uvjeta točna, za spremanje slika potrebno je pritisnuti tipku velikog slova čiju sliku sprema te sve dok brojač ne izbroji 400 slika, proces će se ponavljati. Nakon čega se brojač slika prebacuje na 0, a brojač abecede povećava se za 1. Još je potrebno stvoriti novi direktorij u kojem će se slike spremati te se proces ponavlja za svako slovo abecede. Kad se završi s prikupljanjem slika može se pritiskom na slovo q zaustaviti program te se svi otvoreni prozori (engl. *windows*) zatvore.

```
        if c_alphabet < 26:
            if c_frame < 400:
                if cv2.waitKey(1) == ord(alphabet[c_alphabet]):
                    print(alphabet[c_alphabet])
                    cv2.imwrite( filename: f'{folder}/Image_{time.time()}.jpg', white_image)
                    c_frame += 1
            else:
                c_alphabet += 1

                c_frame = 0
                segments = folder.split('/')
                last_segment = segments.pop()
                remaining_folder = '/'.join(segments)
                folder = remaining_folder + "/" + alphabet[c_alphabet]
                folder_path = Path(folder)
                folder_path.mkdir(parents=True, exist_ok=True)

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

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

Slika 21. Spremanje slika

4.2 Razrada glavnog koda upravljanja računalom

Prvi korak je uvođenje potrebnih biblioteka. Ponovno se koriste biblioteke: MediaPipe, OpenCV i NumPy. Za ovaj kod također potrebne su još biblioteke TensorFlow zbog detekcije slova, PyAutoGUI za funkcije tipkovnice, AutoPy za upravljanje mišem te PyCaw za upravljanje zvukom računala.

```
import mediapipe as mp
import tensorflow as tf
import cv2

import math
import numpy as np

import pyautogui
import autopy

from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
```

Slika 22. Uvođenje biblioteka

Nakon uvođenja biblioteka, kreće se sa inicijalizacijom unaprijed treniranog modela dubokog učenja koji omogućuje detekciju slova. To se postiže definiranjem varijabli koje će sadržavati putanje željenih datoteka. Model se dobiva korištenjem funkcije koja kao ulazni argument uzima putanju datoteke u „.h5“ formatu, što je uobičajeno za Keras modele.

```
# Load the pre-trained model
modelPath = "Model/keras_model.h5"
labelFile = "Model/labels.txt"
Model = tf.keras.models.load_model(modelPath)
Data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
```

Slika 23. Uvođenje modela

Posljednja faza inicijalizacije odnosi se na pristup funkcijama promjene jačine zvuka

```
# Initialize Audio
devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = interface.QueryInterface(IAudioEndpointVolume)
```

Slika 24. Inicijalizacija upravljanja zvuka

Nadalje potrebne su varijable koje će pratiti stanje, pamtiti položaj koordinata ruke i neke konstantne vrijednosti koje će se koristiti tijekom glavnog programa. Također, ako postoji, pokušava se otvoriti datoteka „label.txt“ koja se koristi za detekciju slova.

```
# Constants and Globals
pyautogui.FAILSAFE = False
tracker = np.zeros(20)
x_cord = np.zeros(21)
y_cord = np.zeros(21)
prev_right_x = [0, 0]
prev_right_y = [0, 0]
off_set = 20
mode = 0
prev_left_x, prev_left_y = 240, 240

# Load labels
if labelFile:
    with open(labelFile, "r") as f:
        list_labels = [line.strip() for line in f]
else:
    print("No Labels Found")
```

Slika 25. Inicijalizacija varijabli

Ulazne informacije o upravljanju dolaze iz kamere. Stoga se ponovo, kao u prethodnom kodu, inicijalizira objekt *cap* kojim će se dobivati ulazne slike videozapisa. U ovom slučaju, mijenja se veličina okvira (engl. *frame*) radi bržeg procesiranja. Također, definiraju se varijable *mp_drawing*, *mp_hands* i *hands* kojima će se detektirati ruke, dobivati koordinate te ubacivati ključne točke ruke na slici.

```python
# Video Capture Setup
cap = cv2.VideoCapture(0)

# Resize frame for faster processing
cap.set(cv2.CAP_PROP_FRAME_WIDTH, value: 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, value: 480)

# Initialize MediaPipe components
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=2, min_detection_confidence=0.5,
                       min_tracking_confidence=0.5)
```

Slika 26. Učitavanje i prilagodba veličine slike

Prije objašnjenja koda glavne funkcije, uvest će se pomoćne funkcije koje će učiniti glavnu funkciju čitljivijom. Struktura sljedećih funkcija bit će relativno ista. Uvijek se ispituje visina dvije točke na ruci i ako uvjet ispadne točnim, tada se aktivira funkcija tipkovnice ili miša. Također, stavlja se uvjet za varijablu praćenja, zbog toga što bi bez nje cijelo vrijeme, dok je uvjet ispunjen, funkcija bila aktivna. Neće se staviti sve slike, ali komande koje se mogu pozivati su: lijevi klik miša, dupli lijevi klik miša, desni klik miša, držanje lijevog klika miša, prijelaz na prošle stranice u pretražitelju te skrolanje. Prijelaz između stranica postiže se pomoću kombinacija tipke CTRL i lijeva odnosno desna strelica.

```python
4 usages  JosipJM
def key_press(joint1, joint2, key, num_tracker):
    if tracker[num_tracker] == 0 and joint1 - joint2 < 0:
        pyautogui.press(key)
        tracker[num_tracker] = 1
    if tracker[num_tracker] == 1 and joint1 - joint2 > 0:
        tracker[num_tracker] = 0
```

Slika 27. Funkcija pritiska tipki

```python
1 usage  JosipJM
def click(joint1, joint2):
    if tracker[0] == 0 and joint1 - joint2 < 0:
        print("click")
        pyautogui.click()
        tracker[0] = 1

    elif tracker[0] == 1 and joint1 - joint2 > 0:
        tracker[0] = 0
```

Slika 28. Funkcija lijevog klika miša

```
1 usage   JosipJM
def scroll(joint1, joint2):
    if joint2 - joint1 < 0:
        print(joint1, joint2)
        if joint2 < 640:
            print("scroll down")
            pyautogui.scroll(int((640 - joint1) * 0.5))
        else:
            print("scroll up")
            pyautogui.scroll(int((640 - joint1) * 0.5))
```

Slika 29. Funkcija skrolanja

Ostale funkcije pomažu u računanju ili skraćuju kompleksnije funkcije poput *cv2.circle*. Primjer toga je funkcija *distance* koja služi izračunavanju duljine linije od jednog do drugog vrha kažiprsta za postavljanje jačine zvuka. Funkcija *get_finger_coords* koristit će se za dobivanje koordinata kojima se upravlja mišem. Te koordinate koriste se samo na dva mjesta jer ne govore zapravo gdje se prsti na slici nalaze. Koordinate su umanjene za određeni iznos da se ne bi (u ovom slučaju vrhom kažiprsta) trebalo dodirivati rub slike te da nam miš uspješno dospije do lijevog ili gornjeg ruba na računalu. A množenje broja služi da se poveća dpi (engl. *dots per inch*) i da desni odnosno donji rub zahtijevaju manju vrijednost varijable *indeks_x*. Još bi bilo dobro spomenuti funkciju *mode_change* koja mijenja funkcije programa.

```
# Helper functions
3 usages   JosipJM
def mode_change(joint1, joint2):
    global mode
    if joint1 - joint2 < 0 and tracker[3] == 0:
        tracker[3] = 1
        mode += 1
    elif joint1 - joint2 > 0 and tracker[3] == 1:
        tracker[3] = 0
```

Slika 30. Funkcija promjene načina rada

```
1 usage   JosipJM
def distance(p1, p2, p3, p4):
    return pow(pow(p1 - p2, 2) + pow(p3 - p4, 2), 0.5)
```

Slika 31. Funkcija izračunavanja duljine

```
1 usage   JosipJM
def get_finger_coords(hand, joint):
    index_tip = hand.landmark[joint]
    # *1.5 changes dpi and -400 offsetting minimum needed value to reach edge
    return int(index_tip.x * 1.5 * 1920 - 400), int(index_tip.y * 1.5 * 1080 - 400)
```

Slika 32. Funkcija izračunavanja koordinata miša

```
2 usages  JosipJM *
def get_hand_label_and_wrist_coords(index, hand, results):

    output = None
    if index == 0:
        label = results.multi_handedness[0].classification[0].label
        coords = tuple(np.multiply(
            np.array((hand.landmark[mp_hands.HandLandmark.WRIST].x,
                      hand.landmark[mp_hands.HandLandmark.WRIST].y)),
                      [640, 480]).astype(int))

        output = label, coords
        return output

    if index == 1:
        print(results.multi_handedness)
        label = results.multi_handedness[1].classification[0].label
        coords = tuple(np.multiply(
            np.array((hand.landmark[mp_hands.HandLandmark.WRIST].x,
                      hand.landmark[mp_hands.HandLandmark.WRIST].y)),
                      [640, 480]).astype(int))

        output = label, coords
        return output
    return output, None
```

Slika 33. Funkcija koordinata ruku i označavanja ruke

```
1 usage  JosipJM
def smooth_movement(prev_coords, new_coords, smoothing_factor=0.7):
    smoothed_x = int(prev_coords[0] * smoothing_factor + new_coords[0] * (1 - smoothing_factor))
    smoothed_y = int(prev_coords[1] * smoothing_factor + new_coords[1] * (1 - smoothing_factor))
    return smoothed_x, smoothed_y
```

Slika 34. Funkcija ublažavanja skoka miša

```
1 usage  JosipJM
def move_mouse_safely(x, y):
    global screen_height, screen_width
    x = max(0, min(screen_width - 1, x))
    y = max(0, min(screen_height - 1, y))
    autopy.mouse.move(x, y)
```

Slika 35. Funkcija ograničavanja miša

```
40 usages  JosipJM *
def circle(img, joint_number, color, x_list, y_list):
    return cv2.circle(img,  center: (x_list[joint_number], y_list[joint_number]),
                      radius: 5, color, -1)
```

Slika 36. Funkcija crtanja kruga

Nakon definiranja svih pomoćnih funkcija može se krenuti u funkciju *main* u kojoj se nalazi cijeli kod i funkcije koje ostvaruju ovaj program. Kod funkcije *main* započinje se s *while* petljom koja je uvijek istinita, zatim kao i u prošlom kodu uzima se slika iz kamere i sačuva se u varijablu *frame*. Varijabla *frame* okrene se da slika bude kao zrcalo. Promijeni se *frame* u

RGB i nakon toga se obrađuje *frame* da se dobe koordinate te da se odredi je li riječ o lijevoj ili desnoj ruci.

```
1 usage   JosipJM *
def main():
    while cap.isOpened():
        # frame capturing
        ret, frame = cap.read()

        frame = cv2.flip(frame, flipCode: 1)
        # Convert the frame to RGB for hand detection
        RGB_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        result = hands.process(RGB_frame)
```

Slika 37. Učitavanje i prilagodba slike

Nadalje, se kod grana. Ako nema ruku ili ako nije varijabla *frame* učitana, cijeli sljedeći opisani kod se neće izvršiti. Nadalje iterira se kroz *result.multi_hand_landmarks* i pomoću funkcije *enumerate* dobivaju se indeksi o desnoj ili lijevoj ruci.

```
if result.multi_hand_landmarks and ret:
    for num, hand_landmarks in enumerate(result.multi_hand_landmarks):
        label = ""
        wrist_coords = 0
        try:
            label, wrist_coords = get_hand_label_and_wrist_coords(num, hand_landmarks, result)
        except Exception as e:
            print(f"Error: {e}")
```

Slika 38. Iteriranje kroz pojedinu ruku

Nakon toga provjerava se postojanost varijable *label*, *wrist_coords* i koliki je ostatak varijable *mode*, koja određuje koje se funkcije koriste. Početni ostatak varijable jednak je nula pa je početni način rada kontrola miša. Za jasno korištenje može se staviti tekst koji govori u kojem se načinu rada nalazi program pomoću funkcije *cv2.putText*. Također, može se staviti pokraj svake ruke tekst za raspoznavanje lijeve i desne ruke da se ne klasificiraju pogrešno. Za lakše praćenje glavnih točaka koje će se koriste, mogu se nacrtati u varijablu *frame* sve glavne točke ruku. Za aktivaciju funkcija uspoređivat će se položaji koordinata pa su potrebne koordinate svake ključne točke. One se sačuvaju u listama *xList* i *yList*, dok koordinate za pomak miša sačuvaju se u *x_cord* i *y_cord* listama.

```
if label and wrist_coords and mode % 3 == 0:
    cv2.putText(frame, text: "Mouse control mode", org: (10, 25),
                cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (255, 255, 255), thickness: 2,
                cv2.LINE_AA)
    mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)

    cv2.putText(frame, label, wrist_coords, cv2.FONT_HERSHEY_SIMPLEX,
                fontScale: 1, color: (255, 255, 255), thickness: 2,
                cv2.LINE_AA)
    xList, yList = [], []
    for i in range(21):
        # Getting hand coordinates for mouse movement
        x_cord[i], y_cord[i] = get_finger_coords(hand_landmarks, i)
        x_cord[i], y_cord[i] = int(x_cord[i]), int(y_cord[i])
        px, py = int(hand_landmarks.landmark[i].x * frame.shape[1]), int(
            hand_landmarks.landmark[i].y * frame.shape[0])

        xList.append(px)
        yList.append(py)
```

Slika 39. Dobivanje koordinate pojedinih zglobova

Nakon dobivanja koordinata ispituje se kroz koju ruku se iterira na slici. S obzirom na to da se lijevom rukom upravlja mišem, točnije vrhom kažiprsta, za micanje miša uvijek će biti potrebne prijašnje koordinate, zato ih se uvijek sačuva nakon izvršavanja funkcije. Također, uvijek se provjerava jesu li prijašnje koordinate različite od sadašnjih, nakon čega se koriste pomoćne funkcije. Zbog funkcije *smooth_movment* iako se pozicija promjeni, miš neće odmah skočiti do krajnje točke , nego će se kretati po inkrementima od 30% vrijednosti udaljenosti točaka, kako bi se ublažili skokovi miša.

```
# Left hand functions

if label == "Left":
    global prev_left_x, prev_left_y
    if prev_left_x != x_cord[0] or prev_left_y != y_cord[0]:
        smoothed_coords = smooth_movement( prev_coords: (prev_left_x, prev_left_y), new_coords: (x_cord[8], y_cord[8]))
        move_mouse_safely(smoothed_coords[0], smoothed_coords[1])
        prev_left_x, prev_left_y = smoothed_coords
```

Slika 40. Funkcije lijeve ruke

S obzirom na to da se koriste liste *x_cord* i *y_cord,* područje upravljanja miša na videokameri se smanji kako se ruke ne bi preklapale tijekom upravljanja računalom, što bi prouzročilo komplikacije.

Svaka funkcija koja se nalazi na desnoj ruci aktivira se kada vrh prsta bude na nižoj poziciji od nižeg zgloba iste boje. Na desnoj ruci nalazi se 8 funkcija koje uključuju: lijevi klik, desni klik, dvostruki lijevi, držanje lijevog klika, skrolanje, brz prijelaz na prošle stranice preglednika i funkcija prelaska na druge funkcionalnosti programa koje nisu povezane s mišem.

Implementira se 8 funkcija koje ovise o poziciji palca. Svaka funkcija aktivira se oponašanjem klika pojedinog prsta.

```
# Right hand functions

if label == "Right":

    if xList[4] - xList[5] < 0:
        click(yList[7], yList[8])
        double_click(yList[11], yList[12])
        left_click(yList[19], yList[20])
        hold_click(yList[15], yList[16])

        circle(frame, joint_number: 8, color: (255, 160, 5), xList, yList)
        circle(frame, joint_number: 7, color: (255, 150, 5), xList, yList)

        circle(frame, joint_number: 11, color: (100, 255, 80), xList, yList)
        circle(frame, joint_number: 12, color: (100, 255, 80), xList, yList)

        circle(frame, joint_number: 20, color: (200, 213, 48), xList, yList)
        circle(frame, joint_number: 19, color: (200, 213, 48), xList, yList)

        circle(frame, joint_number: 16, color: (168, 156, 205), xList, yList)
        circle(frame, joint_number: 15, color: (168, 156, 205), xList, yList)

    else:
        back(yList[7], yList[8])
        forward(yList[11], yList[12])
        scroll(y_cord[16], y_cord[15])
        mode_change(yList[17], yList[20])
```

Slika 41. Funkcije desne ruke

Sljedeći način rada je kontrola zvuka računala. Grananje je slično u svakom načinu rada. Na početku se provjerava trenutna jačina zvuka računala i za lakše snalaženje ispisuje se na varijablu *frame*. Ponovo se iterira kroz zglobove i spremaju se njihove koordinate u liste. Kada iterira kroz indeks desne ruke, u privremenu varijablu sprema se koordinata vrha kažiprsta, a kada se iterira po indeksu lijeve ruke izračunava se udaljenost točaka kažiprsta lijeve i desne ruke kako bi se interpolirala ta vrijednost i prenijela se u jačinu zvuka računala. Odabrane su proizvoljne vrijednosti za minimalnu i maksimalnu udaljenost. Za vizualizaciju je nacrtana linija koja prikazuje duljinu udaljenosti.

```
if label and wrist_coords and mode % 3 == 1:
    windows_volume = int(volume.GetMasterVolumeLevelScalar() * 100)
    cv2.putText(frame, "Volume: {}".format(windows_volume), org: (10, 25),
                cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1,
                color: (255, 255, 255), thickness: 2, cv2.LINE_AA)
    cv2.putText(frame, label, wrist_coords, cv2.FONT_HERSHEY_SIMPLEX,
                fontScale: 1, color: (255, 255, 255), thickness: 2,
                cv2.LINE_AA)
    mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)
    xList, yList = [], []
    for lm in hand_landmarks.landmark:
        px, py = int(lm.x * frame.shape[1]), int(lm.y * frame.shape[0])
        xList.append(px)
        yList.append(py)
```

Slika 42. Ispis trenutne jačine zvuka i dohvaćanje koordinata zglobova

```
# Left hand functions

if label == "Left":
    circle(frame, joint_number: 8, color: (200, 213, 48), xList, yList)
    points_distance = int(distance(prev_right_x[0], xList[8],
                                   prev_right_y[0], yList[8]))
    points_distance = max(120, min(points_distance, 350))
    vol = np.interp(points_distance, xp: [120, 350], fp: [-96, 0])
    if yList[18] < yList[20]:
        try:
            volume.SetMasterVolumeLevel(vol, None)
        except Exception as e:
            print(f"Error: {e}")
    circle(frame, joint_number: 20, color: (200, 213, 48), xList, yList)
    circle(frame, joint_number: 18, color: (200, 213, 48), xList, yList)
    type(prev_right_x[0])
    cv2.line(frame, pt1: (int(prev_right_x[0]), int(prev_right_y[0])),
             pt2: (xList[8], yList[8]),
             color: (200, 213, 48), thickness: 2, cv2.LINE_AA)
```

Slika 43. Funkcije lijeve ruke kod promjene jačine zvuka

```
# Right hand functions

if label == "Right":
    circle(frame, joint_number: 8, color: (200, 213, 48), xList, yList)
    if xList[5] < xList[4]:
        mode_change(yList[17], yList[20])
        circle(frame, joint_number: 5, color: (255, 160, 5), xList, yList)

        circle(frame, joint_number: 20, color: (200, 213, 48), xList, yList)
        circle(frame, joint_number: 17, color: (200, 213, 48), xList, yList)
    circle(frame, joint_number: 4, color: (255, 160, 5), xList, yList)

    prev_right_x[0], prev_right_y[0] = xList[8], yList[8]
```

Slika 44. Funkcije desne ruke kod promjene jačine zvuka

Posljednje funkcije koda su funkcije vezane uz tipkovnicu. Cilj ovog načina rada je upisivanje slova u računalo i ostvarivanje ostalih funkcija vezanih uz manipulaciju teksta. S obzirom na to da se ne može koristiti jednostavan način upisivanja slova kao kod aktivacija funkcija miša zbog ograničenog broja aktivacija pomoću samih koordinata, koristi se Keras model za prepoznavanje jedinstvenih pozicija.

Budući da se koristi Keras model, prvo se slika mora obraditi na način kao i kod prikupljanja podataka kako bi model lakše prepoznao geste.

Za lakše pisanje desnoj se ruci dodjeljuju dodatne funkcije poput brisanja, kretanja lijevo ili desno po tekstu i pritisak na tipku enter.

```
# Right hand functions

if label == "Right":
    prev_right_y[0], prev_right_y[1] = yList[8], yList[7]

    if xList[5] < xList[4]:
        key_press(yList[11], yList[12], key: "left", num_tracker: 9)
        key_press(yList[15], yList[16], key: "right", num_tracker: 10)


        mode_change(yList[17], yList[20])
    else:

        key_press(yList[19], yList[20], key: "backspace", num_tracker: 11)
        key_press(yList[11], yList[12], key: "enter", num_tracker: 12)
```

Slika 45. Funkcije desne ruke kod upravljanja tipkovnice

Lijeva ruka služit će za pozicioniranje ruke, kako bi se s nje očitala slova. Za prepoznavanje slova koristi se TensorFlow bibliotekom. Pomoću prethodno definirane varijable *Model* dobije se pretpostavka o kojem je slovu riječ. Dobije se niz predikcija te pomoću maksimalnog postotka preklapanja dobije se indeks. Tada se pomoću indeksa dobije malo slovo.

```
prediction = Model.predict(Data)
indexVal = np.argmax(prediction)
label = list_labels[indexVal]
label_split = label.split(" ")
last = label_split.pop()
```

Slika 46. Pretpostavka pozicije ruke

I na kraju se ispituju položaji zglobova i ako je uvjet ispunjen pritisne se slovo.

```
if (prev_right_y[1] < prev_right_y[0] and tracker[7] == 0 and
        prev_right_y[1] != 0 and prev_right_y[0] != 0):
    pyautogui.press(last.lower())
    tracker[7] = 1
elif prev_right_y[1] > prev_right_y[0] and tracker[7] == 1:
    tracker[7] = 0
```

Slika 47 Pritisak željene tipke

Na kraju, ako su uvjeti ispunjeni, program se prekida. Kao posljednji korak, poziva se funkcija *main* koja aktivira ostatak koda.

```
        # Display the camera frame
        cv2.imshow( winname: "Camera", frame)

        if cv2.waitKey(1) & 0xFF == ord('Q') or not cv2.getWindowProperty( winname: "Camera", cv2.WND_PROP_VISIBLE):
            break

    cap.release()
    cv2.destroyAllWindows()


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

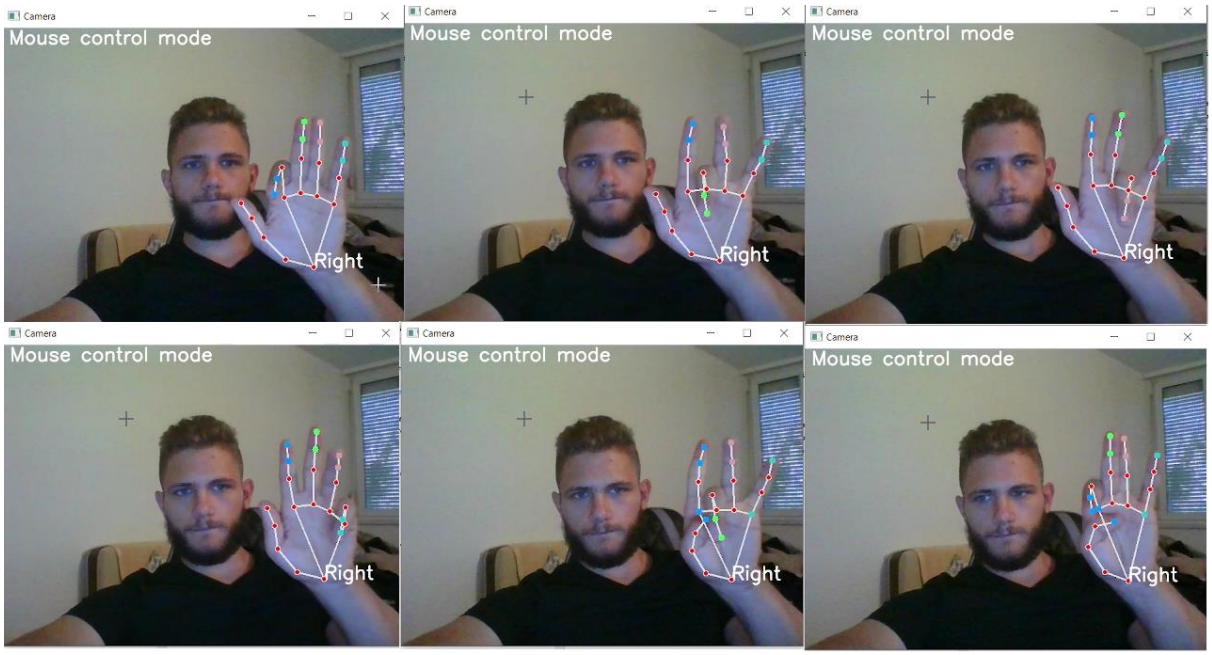
Slika 48. Uvjeti prestanka rada programa

5. EVALUACIJA PROGRAMA

U nastavku će se program izložiti normalnim i neprikladnim uvjetima za bolje razumijevanje rada koda.

5.1 Prvi primjer – normalni uvjeti korištenja

Prvo se evaluiraju pokreti miša zajedno s osnovnim funkcijama. Prema slici 50. može se vidjeti da upravljanje funkcijama miša uspijeva bez ikakvih pogrešaka i s visokom točnošću te da se svaka funkcija izvrši samo jednom, kako je i namijenjeno.

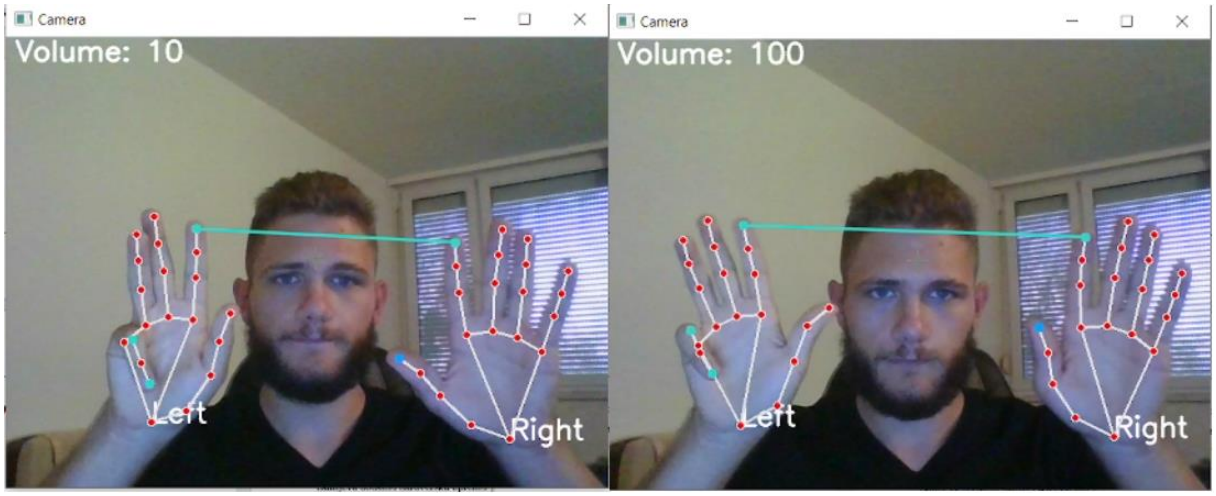


Slika 49. Primjer normalnih uvjeta kamere

```
click
double click
let go
hold
left click
back
forward
scroll down
```

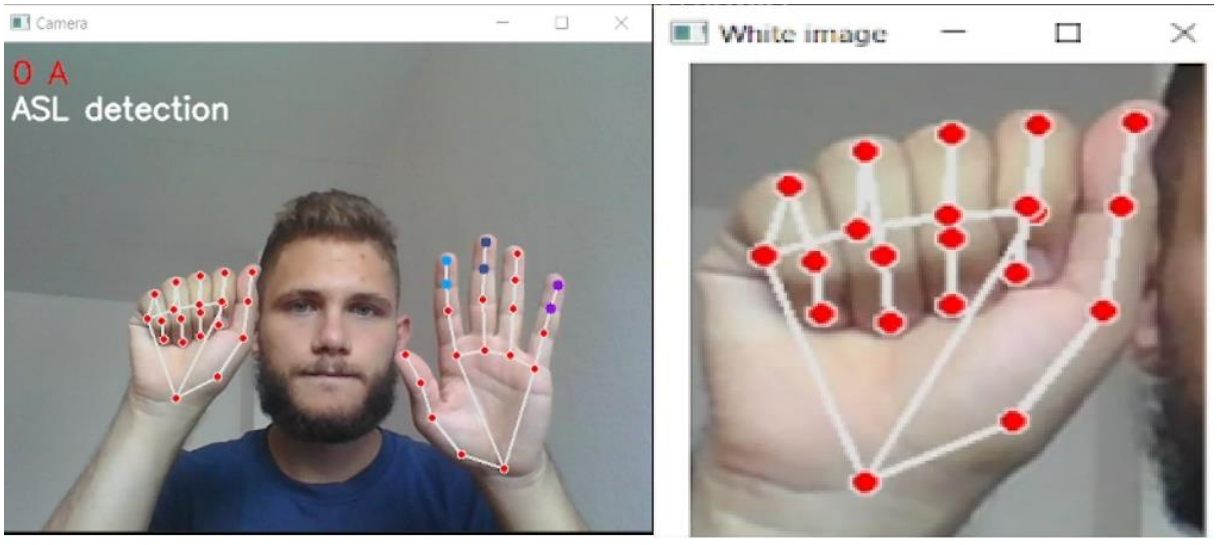
Slika 50. Izvršene naredbe

Sljedeća evaluacija vrši se na dijelu programa koji upravlja jačinom zvuka. Kao i kod kontrole miša, program radi točno i precizno. Program ovisi isključivo o točnosti detekcije ruke pomoću MediaPipe-a, tj. ako se točno detektira ruka, program će raditi ispravno.



Slika 51. Primjer normalnih uvjeta kod upravljanja jačinom zvuka

I posljednja evaluacija odnosi se na detekciju ASL znakovnog jezika. Ovo je jedini dio koda gdje točnost i preciznost ne ovise o MediaPipe-u. Iz evaluacije je vidljivo da model nije savršen. Model ponekad mješe slova M, T i N koja su vrlo slična i mogu ih se opisati kao prekrivanje kažiprsta s određenim brojem prstiju, što otežava modelu prepoznavanje. S obzirom na to da pritisak tipkovnice ovisi o MediaPipe-u, on ispravno radi.

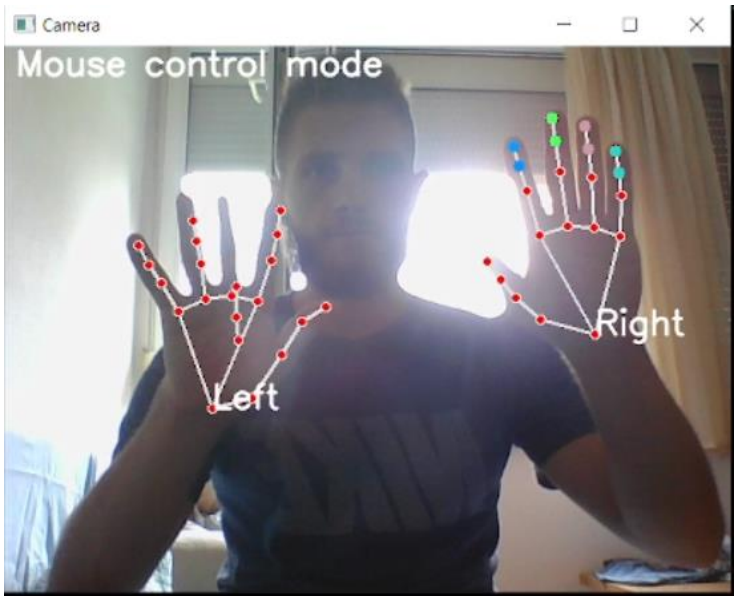


Slika 52. Primjer normalnih uvjeta kod detekcije slova

5.2 Drugi primjer – korištenje kod neprikladnih uvjeta

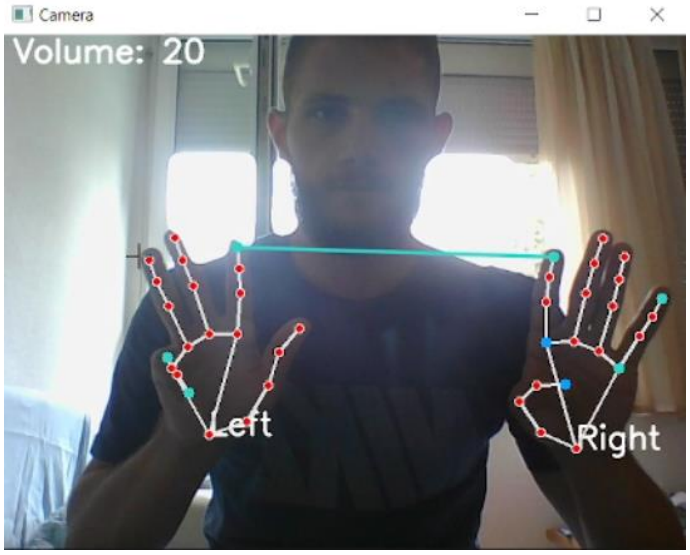
Nakon što je program ispitan pod normalnim uvjetima, slijedi ispitivanje pod manje optimalnim uvjetima.

Prvo se ponovno ispituje kontrola miša. Vidljivo je da kod lošeg osvijetljena dolazi do pogrešne detekcije ruke pomoću MediaPipe-a, što je i za očekivati jer je svijetlost prekrila većinu srednjeg prsta.



Slika 53. Primjer kontrole miša kod nepovoljnih uvjeta

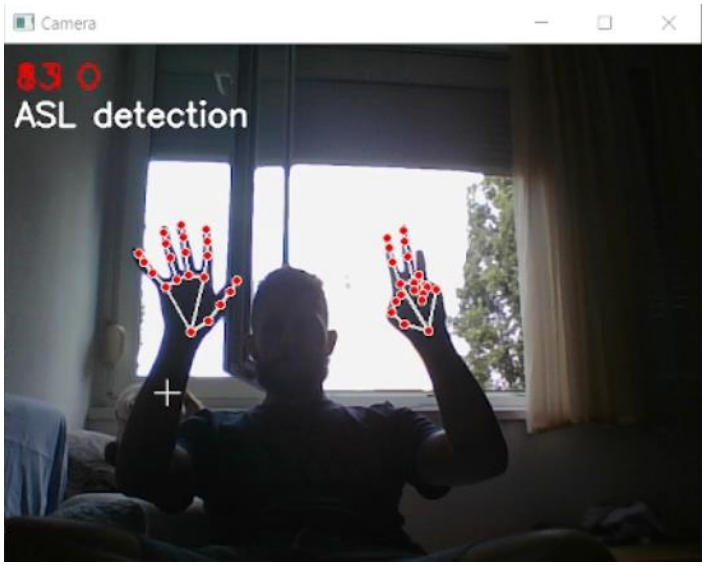
Za kontrolu zvuka, sve je u redu osim što zbog lošijeg osvjetljenja ruka može biti neuredna i tresti se, što otežava precizno namještanje zvuka.



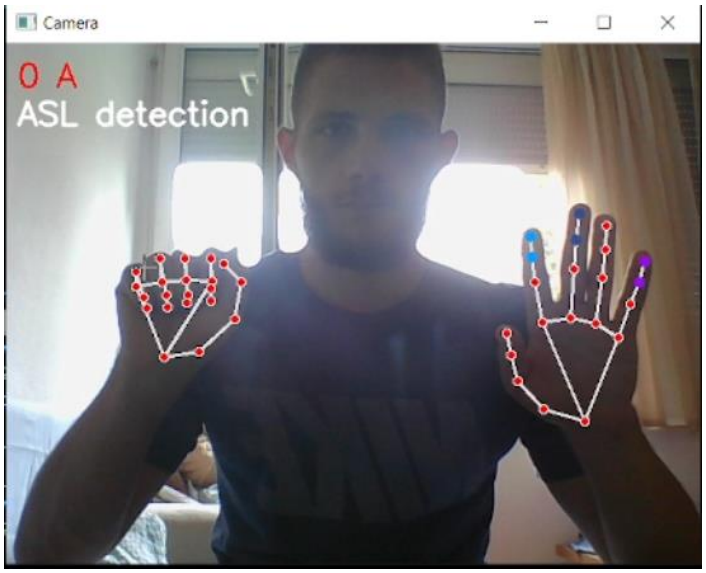
Slika 54. Primjer upravljanje jačinom zvuka kod nepovoljnih uvjeta

Kod detekcije znakovnog jezika dogodilo se da model nije mogao prepoznati desnu i lijevu ruku, što je rezultiralo nemogućnošću dodjeljivanja funkcija desnoj ruci. Ovo se dogodilo samo

na nekoliko sekundi, nakon čega se sve vratilo u normalu. U ovakvim uvjetima, neke detekcije su se poboljšale, osim naravno slova M, N i T koja su ponovno teška za prepoznavanje. Ta bi slova ljudi mogli prepoznati samo na temelju broja nadvišenih prstiju, što bi indiciralo koliko prstiju prekriva palac.



Slika 55. Primjer pogrešne pri detekciji ruku



Slika 56. Primjer uspješne detekcije znakovnog jezika kod nepovoljnih uvjeta

6. OSVRT NA OSTVARANE REZULTATE

Rezultati evaluacije ukazuju na dobar rad programa. Naravno, program nije savršen, ima mjesta za poboljšanja. Poboljšanja koja bi se mogla napraviti u budućnosti odnose se na šaku. Naš program obrađuje cijelu sliku, dok se šaka može nalaziti pod bilo kojim kutom. Kako bi se riješile slučajne aktivacije funkcija, potrebno je dodati još jednu varijablu koja bi se obrađivala, a koja uvijek ima uspravnu ruku.

Sljedeće poboljšanje odnosi se na jačinu zvuka. Kada se prilagođava jačina zvuka, primjećuje se da promjena nije linearna, već više nalikuje kvadratnoj ili nekoj drugoj brzo rastućoj funkciji. Kada se želi smanjiti jačina zvuka na nulu, vidljivo je da je promjena između 10% i 0% jednaka opsegu promjene između 10% i 100%. Moglo bi se za svaku vrijednost duljine zapisati odgovarajuća vrijednost jačine zvuka i interpolirati polinom kako bi se dobila funkcija koja omogućuje točnije upravljanje zvukom

Posljednje poboljšanje vezano je uz detekciju znakovnog jezika. Najlakše rješenje za povećanje točnosti modela bilo bi korištenje više slika za treniranje modela. Kao što je spomenuto u prethodnim poglavljima, model za detekciju ruku treniran je na 30.000 slika, što bi, naravno, zahtijevalo mnogo više vremena u usporedbi s 400 slika koje su upotrjebljene u ovom radu. Druga mogućnost bila bi razviti potpuno jedinstvene poze ruku za svako slovo kako bi model lakše prepoznavao slova, ili pomoću BSL (britanski znakovni jezik) umjesto ASL-a. U tom slučaju obje bi ruke bile zauzete i ne bi se moglo upisivati slova u računalo kao što je to moguće s ASL-om.

Ali čak i bez tih promjena, vidi se iz rezultata da je program funkcionalan i da se može koristiti kao sučelje između čovjeka i računala.

7. ZAKLJUČAK

U ovom završnom radu razvijen je sustav za upravljanje računalom pomoću računalnog vida. Primjenom biblioteke MediaPipe za prepoznavanje gesta ruku u kombinaciji s PyAutoGUI za automatsko izvršavanje naredbi, omogućeno je beskontaktno upravljanje računalom pomoću obične kamere. Ovaj pristup čini sustav izuzetno pristupačnim i široko primjenjivim, jer ne zahtijeva dodatnu hardversku opremu.

Cilj ovog rada bio je omogućiti interakciju korisnika s računalom bez korištenja dodatnih hardverskih sredstava te gestama omogućiti jednostavne funkcije miša i tipkovnice. Ovaj način interakcije ima veliki potencijal primjene u prezentacijama ili upravljanju multimedijskim sadržajima kada korisnik nije u neposrednoj blizini računala

Rezultati rada pokazuju da je moguće implementirati upravljanje gestama ruku ne samo na računalima, već i na drugim uređajima. Buduće nadogradnje ovog sustava mogle bi uključivati optimizaciju performansi za različite platforme i personalizaciju gesti.

Kao zaključak, može se reći da je implementacija ovog sustava pokazala veliki potencijal za daljnji razvoj u području beskontaktnog upravljanja uređajima. Korištenje tehnologija poput računalnog vida i prepoznavanja gesta otvara nove mogućnosti u interakciji s tehnologijom, čineći je intuitivnijom i pristupačnijom.

LITERATURA

[1] What is MediaPipe? A Guide for Beginners: https://blog.roboflow.com/what-is-mediapipe/, dostupno dana 4. rujna 2024.

[2] MediaPipe Solutions guide: https://ai.google.dev/edge/mediapipe/solutions/guide, dostupno dana 4. rujna 2024.

[3] Hand landmarks guide: https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker, dostupno dana 4. rujna 2024.

[4] About OpenCV: https://opencv.org/about/, dostupno dana 4. rujna 2024.

[5] What Is Tensorflow: https://www.nvidia.com/en-au/glossary/tensorflow/, dostupno dana 4. rujna 2024.

[6] Using Deep Learning for Information Security – Part 2: https://www.acalvio.com/resources/blog/using-deep-learning-for-information-security-part-2/, dostupno dana 4. rujna 2024.

[7] NumPy: the absolute basics for beginners: https://numpy.org/doc/stable/user/absolute_beginners.html, dostupno dana 4. rujna 2024.

[8] PyAutoGUI's documentation: https://pyautogui.readthedocs.io/en/latest/, dostupno dana 4. rujna 2024.

[9] Aurélien Géron, Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2019.

[10] Reinforcement Learning: https://aigents.co/learn/Reinforcement-Learning, dostupno dana 4. rujna 2024.

[11] Szeliski, R., *Computer Vision: Algorithms and Applications*. Springer, 2010.

[12] Optičko prepoznavanje znakova: https://sh.wikipedia.org/wiki/Opti%C4%8Dko_prepoznavanje_znakova, dostupno dana 4. rujna 2024.

[13] Design of 3D Modeling Face Image Library in Multimedia Film and Television: https://www.researchgate.net/figure/Schematic-diagram-of-image-sequence-and-camera-movement_fig1_355340074, dostupno dana 4. rujna 2024.

[14] Visage Technologies — Bilo kuda, računalni vid svuda: https://medium.com/penkala-blog/visage-technologies-bilo-kuda-ra%C4%8Dunalni-vid-svuda-8d98a48f9412, dostupno dana 4. rujna 2024.

[15] Teachable Machine: Approachable Web-Based Tool for Exploring Machine Learning Classification: https://research.google/pubs/teachable-machine-approachable-web-based-tool-for-exploring-machine-learning-classification/, dostupno dana 4. rujna 2024.

PRILOZI

I. Python kod – Prikupljanje podataka

II. Python kod – Upravljanje računalom

I. Python kod – Prikupljanje podataka

```
import time
import mediapipe as mp
import cv2
from pathlib import Path

import numpy as np

folder = "Data/A"
c_alphabet = 0
c_frame = 0
alphabet = list(map(chr, range(65, 91)))

mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=2, min_detection_confidence=0.5, min_tracking_confidence=0.5)

cap = cv2.VideoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)
    RGB_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    result = hands.process(RGB_frame)

    if result.multi_hand_landmarks:

        mp_drawing.draw_landmarks(frame, result.multi_hand_landmarks[0], mp_hands.HAND_CONNECTIONS)

        # Extract landmark positions
        lm_list = [(int(lm.x * 640), int(lm.y * 480)) for lm in
result.multi_hand_landmarks[0].landmark]
        x_list, y_list = zip(*lm_list)

        xmin, xmax = min(x_list), max(x_list)
        ymin, ymax = min(y_list), max(y_list)

        # Extract hand region
        hand_region = frame[max(0, ymin - 20):min(480, ymax + 20), max(0, xmin - 20):min(640, xmax +
20)]

        # Prepare the white image
        white_image = np.ones((300, 300, 3), dtype=np.uint8) * 255

        h, w = ymax - ymin + 40, xmax - xmin + 40
        if h > w:
            scale = 300 / h
            new_w = min(int(scale * w), 300)
            resized = cv2.resize(hand_region, (new_w, 300))
            x_offset = (300 - new_w) // 2
            white_image[:, x_offset:x_offset + new_w] = resized
        else:
            scale = 300 / w
            new_h = min(int(scale * h), 300)
            resized = cv2.resize(hand_region, (300, new_h))
            y_offset = (300 - new_h) // 2
            white_image[y_offset:y_offset + new_h, :] = resized

        # Display the results
        cv2.imshow('White image', white_image)
        cv2.imshow("Camera", frame)


        if c_alphabet < 26:
            if c_frame < 400:
                if cv2.waitKey(1) == ord(alphabet[c_alphabet]):
                    print(alphabet[c_alphabet])
                    cv2.imwrite(f'{folder}/Image_{time.time()}.jpg', white_image)
```

```
                    c_frame += 1
            else:
                c_alphabet += 1

                c_frame = 0
                segments = folder.split('/')
                last_segment = segments.pop()
                remaining_folder = '/'.join(segments)
                folder = remaining_folder + "/" + alphabet[c_alphabet]
                folder_path = Path(folder)
                folder_path.mkdir(parents=True, exist_ok=True)

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

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

II. Python kod – Upravljanje računalom

```
import mediapipe as mp
import tensorflow as tf
import cv2

import math
import numpy as np

import pyautogui
import autopy

from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

# Load the pre-trained model
modelPath = "Model/keras_model.h5"
labelFile = "Model/labels.txt"
Model = tf.keras.models.load_model(modelPath)
Data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

# Initialize Audio
devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
volume = interface.QueryInterface(IAudioEndpointVolume)

# Constants and Globals
pyautogui.FAILSAFE = False
tracker = np.zeros(20)
x_cord = np.zeros(21)
y_cord = np.zeros(21)
prev_right_x = [0, 0]
prev_right_y = [0, 0]
off_set = 20
mode = 0
prev_left_x, prev_left_y = 240, 240

# Load labels
if labelFile:
    with open(labelFile, "r") as f:
        list_labels = [line.strip() for line in f]
else:
    print("No Labels Found")

# Video Capture Setup
cap = cv2.VideoCapture(0)

# Resize frame for faster processing
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

# Initialize MediaPipe components
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=2, min_detection_confidence=0.5,
                       min_tracking_confidence=0.5)

screen_width, screen_height = autopy.screen.size()


# Helper functions


def mode_change(joint1, joint2):
    global mode
    if joint1 - joint2 < 0 and tracker[3] == 0:
        tracker[3] = 1
```

```
        mode += 1
    elif joint1 - joint2 > 0 and tracker[3] == 1:
        tracker[3] = 0


def distance(p1, p2, p3, p4):
    return pow(pow(p1 - p2, 2) + pow(p3 - p4, 2), 0.5)


def get_finger_coords(hand, joint):
    index_tip = hand.landmark[joint]
    # *1.5 changes dpi and -400 offsetting minimum needed value to reach edge
    return int(index_tip.x * 1.5 * 1920 - 400), int(index_tip.y * 1.5 * 1080 - 400)


def click(joint1, joint2):
    if tracker[0] == 0 and joint1 - joint2 < 0:
        print("click")
        pyautogui.click()
        tracker[0] = 1

    elif tracker[0] == 1 and joint1 - joint2 > 0:
        tracker[0] = 0


def double_click(joint1, joint2):
    if tracker[1] == 0 and joint1 - joint2 < 0:
        print("double click")
        pyautogui.doubleClick()
        tracker[1] = 1

    elif tracker[1] == 1 and joint1 - joint2 > 0:
        tracker[1] = 0


def scroll(joint1, joint2):
    if joint2 - joint1 < 0:
        print(joint1, joint2)
        if joint2 < 640:
            print("scroll down")
            pyautogui.scroll(int((640 - joint1) * 0.5))
        else:
            print("scroll up")
            pyautogui.scroll(int((640 - joint1) * 0.5))


def left_click(joint1, joint2):
    if tracker[2] == 0 and joint1 - joint2 < 0:
        print("left click")
        pyautogui.click(button='right')
        tracker[2] = 1
    elif tracker[2] == 1 and joint1 - joint2 > 0:
        tracker[2] = 0


def back(joint1, joint2):
    if tracker[4] == 0 and joint1 - joint2 < 0:
        print("back")
        with pyautogui.hold('alt'):
            pyautogui.press(['left'])
        tracker[4] = 1
    if tracker[4] == 1 and joint1 - joint2 > 0:
        tracker[4] = 0


def forward(joint1, joint2):
    if tracker[5] == 0 and joint1 - joint2 < 0:
        print("forward")
        with pyautogui.hold('alt'):
            pyautogui.press(['right'])
```

```
        tracker[5] = 1
    if tracker[5] == 1 and joint1 - joint2 > 0:
        tracker[5] = 0


def hold_click(joint1, joint2):
    if joint1 - joint2 > 0 and tracker[6] == 1:
        print("hold")
        pyautogui.mouseUp()
        tracker[6] = 0
    elif joint1 - joint2 < 0 and tracker[6] == 0:
        print("let go")
        pyautogui.mouseDown()
        tracker[6] = 1


def enter(joint1, joint2):
    if tracker[12] == 0 and joint1 - joint2 < 0:
        pyautogui.press('enter')
        tracker[12] = 1
    if tracker[12] == 1 and joint1 - joint2 > 0:
        tracker[12] = 0


def key_press(joint1, joint2, key, num_tracker):
    if tracker[num_tracker] == 0 and joint1 - joint2 < 0:
        pyautogui.press(key)
        tracker[num_tracker] = 1
    if tracker[num_tracker] == 1 and joint1 - joint2 > 0:
        tracker[num_tracker] = 0


def get_hand_label_and_wrist_coords(index, hand, results):
    output = None
    if index == 0:
        label = results.multi_handedness[0].classification[0].label
        coords = tuple(np.multiply(
            np.array((hand.landmark[mp_hands.HandLandmark.WRIST].x,
                      hand.landmark[mp_hands.HandLandmark.WRIST].y)),
            [640, 480]).astype(int))

        output = label, coords
        return output

    if index == 1:
        print(results.multi_handedness)
        label = results.multi_handedness[1].classification[0].label
        coords = tuple(np.multiply(
            np.array((hand.landmark[mp_hands.HandLandmark.WRIST].x,
                      hand.landmark[mp_hands.HandLandmark.WRIST].y)),
            [640, 480]).astype(int))

        output = label, coords
        return output
    return output, None


def smooth_movement(prev_coords, new_coords, smoothing_factor=0.7):
    smoothed_x = int(prev_coords[0] * smoothing_factor + new_coords[0] * (1 - smoothing_factor))
    smoothed_y = int(prev_coords[1] * smoothing_factor + new_coords[1] * (1 - smoothing_factor))
    return smoothed_x, smoothed_y


def move_mouse_safely(x, y):
    global screen_height, screen_width
    x = max(0, min(screen_width - 1, x))
    y = max(0, min(screen_height - 1, y))
    autopy.mouse.move(x, y)
```

```
def circle(img, joint_number, color, x_list, y_list):
    return cv2.circle(img, (x_list[joint_number], y_list[joint_number]),
                      5, color, -1)


def main():
    while cap.isOpened():
        # frame capturing
        ret, frame = cap.read()

        frame = cv2.flip(frame, 1)
        # Convert the frame to RGB for hand detection
        RGB_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        result = hands.process(RGB_frame)

        if result.multi_hand_landmarks and ret:
            for num, hand_landmarks in enumerate(result.multi_hand_landmarks):
                label = ""
                wrist_coords = 0
                try:
                    label, wrist_coords = get_hand_label_and_wrist_coords(num, hand_landmarks,
result)
                except Exception as e:
                    print(f"Error: {e}")

                ################################################################################
                ################################################################################
                # ASL detection
                ################################################################################
                ################################################################################

                if label and wrist_coords and mode % 3 == 2:
                    cv2.putText(frame, "ASL detection ", (10, 75),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2,
                                cv2.LINE_AA)

                    mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)

                    try:
                        label, wrist_coords = get_hand_label_and_wrist_coords(num, hand_landmarks,
result)
                    except Exception as e:
                        print(f"Error: {e}")

                    # Collect landmark positions
                    xList, yList = [], []
                    for lm in hand_landmarks.landmark:
                        px, py = int(lm.x * frame.shape[1]), int(lm.y * frame.shape[0])
                        xList.append(px)
                        yList.append(py)

                    # Calculate bounding box
                    x_min, x_max = min(xList), max(xList)
                    y_min, y_max = min(yList), max(yList)

                    # Create a white image to hold the processed hand image
                    white_image = np.ones((300, 300, 3), dtype=np.uint8) * 255

                    # Resize hand region
                    White_Frame = frame[y_min - 20:y_max + 20, x_min - 20:x_max + 20]
                    ratio = (y_max - y_min + 40) / (x_max - x_min + 40)

                    # Right hand functions

                    if label == "Right":
                        prev_right_y[0], prev_right_y[1] = yList[8], yList[7]

                        if xList[5] < xList[4]:
                            key_press(yList[11], yList[12], "left", 9)
                            key_press(yList[15], yList[16], "right", 10)
```

```
                circle(frame, 11, (147, 81, 56), xList, yList)
                circle(frame, 12, (147, 81, 56), xList, yList)

                circle(frame, 15, (65, 235, 166), xList, yList)
                circle(frame, 16, (65, 235, 166), xList, yList)

                circle(frame, 4, (255, 160, 5), xList, yList)
                circle(frame, 5, (255, 160, 5), xList, yList)

                circle(frame, 20, (150, 255, 255), xList, yList)
                circle(frame, 17, (150, 255, 255), xList, yList)

                mode_change(yList[17], yList[20])
            else:

                key_press(yList[19], yList[20], "backspace", 11)
                key_press(yList[11], yList[12], "enter", 12)

                circle(frame, 11, (147, 81, 56), xList, yList)
                circle(frame, 12, (147, 81, 56), xList, yList)

                circle(frame, 8, (255, 160, 5), xList, yList)
                circle(frame, 7, (255, 160, 5), xList, yList)

                circle(frame, 20, (235, 15, 154), xList, yList)
                circle(frame, 19, (235, 15, 154), xList, yList)

        # Left hand functions

        if label == "Left":
            # white_image adjusting
            try:
                if ratio > 1:
                    k = 300 / (y_max - y_min + 40)
                    wCal = math.ceil(k * (x_max - x_min + 40))
                    imgResize = cv2.resize(White_Frame, (wCal, 300))
                    wGap = (300 - wCal) // 2
                    white_image[:, wGap:wGap + wCal] = imgResize
                else:
                    k = 300 / (x_max - x_min + 40)
                    hCal = math.ceil(k * (y_max - y_min + 40))
                    imgResize = cv2.resize(White_Frame, (300, hCal))
                    hGap = (300 - hCal) // 2
                    white_image[hGap:hGap + hCal, :] = imgResize
            except Exception as e:
                print(f"Error: {e}")

            cv2.imshow('White image', white_image)

            # Prepare the white image for prediction
            white_image_resized = cv2.resize(white_image, (224, 224))
            frame_array = np.asarray(white_image_resized)
            normalized_frame_array = (frame_array.astype(np.float32) / 127.0) - 1
            Data[0] = normalized_frame_array
            prediction = Model.predict(Data)
            indexVal = np.argmax(prediction)
            label = list_labels[indexVal]
            label_split = label.split(" ")
            last = label_split.pop()

            # Display the prediction label
            cv2.putText(frame, str(label), (10, 40),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

            if (prev_right_y[1] < prev_right_y[0] and tracker[7] == 0 and
                    prev_right_y[1] != 0 and prev_right_y[0] != 0):
                pyautogui.press(last.lower())
                tracker[7] = 1
            elif prev_right_y[1] > prev_right_y[0] and tracker[7] == 1:
```

```
                tracker[7] = 0

            ##########################################################################
            ##########################################################################
            # Volume adjustment
            ##########################################################################
            ##########################################################################

            if label and wrist_coords and mode % 3 == 1:
                windows_volume = int(volume.GetMasterVolumeLevelScalar() * 100)
                cv2.putText(frame, "Volume: {}".format(windows_volume), (10, 25),
                            cv2.FONT_HERSHEY_SIMPLEX, 1,
                            (255, 255, 255), 2, cv2.LINE_AA)
                cv2.putText(frame, label, wrist_coords, cv2.FONT_HERSHEY_SIMPLEX,
                            1, (255, 255, 255), 2,
                            cv2.LINE_AA)
                mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)
                xList, yList = [], []
                for lm in hand_landmarks.landmark:
                    px, py = int(lm.x * frame.shape[1]), int(lm.y * frame.shape[0])
                    xList.append(px)
                    yList.append(py)

                # Left hand functions

                if label == "Left":
                    circle(frame, 8, (200, 213, 48), xList, yList)
                    points_distance = int(distance(prev_right_x[0], xList[8],
                                                   prev_right_y[0], yList[8]))
                    points_distance = max(120, min(points_distance, 350))
                    vol = np.interp(points_distance, [120, 350], [-96, 0])
                    if yList[18] < yList[20]:
                        try:
                            volume.SetMasterVolumeLevel(vol, None)
                        except Exception as e:
                            print(f"Error: {e}")
                    circle(frame, 20, (200, 213, 48), xList, yList)
                    circle(frame, 18, (200, 213, 48), xList, yList)
                    type(prev_right_x[0])
                    cv2.line(frame, (int(prev_right_x[0]), int(prev_right_y[0])),
                             (xList[8], yList[8]),
                             (200, 213, 48), 2, cv2.LINE_AA)

                # Right hand functions

                if label == "Right":
                    circle(frame, 8, (200, 213, 48), xList, yList)
                    if xList[5] < xList[4]:
                        mode_change(yList[17], yList[20])
                        circle(frame, 5, (255, 160, 5), xList, yList)

                        circle(frame, 20, (200, 213, 48), xList, yList)
                        circle(frame, 17, (200, 213, 48), xList, yList)
                    circle(frame, 4, (255, 160, 5), xList, yList)

                    prev_right_x[0], prev_right_y[0] = xList[8], yList[8]

            ##########################################################################
            ##########################################################################
            # Mouse control
            ##########################################################################
            ##########################################################################
            if label and wrist_coords and mode % 3 == 0:
                cv2.putText(frame, "Mouse control mode", (10, 25),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2,
                            cv2.LINE_AA)
                mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)

                cv2.putText(frame, label, wrist_coords, cv2.FONT_HERSHEY_SIMPLEX,
                            1, (255, 255, 255), 2,
```

```
                                cv2.LINE_AA)
                    xList, yList = [], []
                    for i in range(21):
                        # Getting hand coordinates for mouse movement
                        x_cord[i], y_cord[i] = get_finger_coords(hand_landmarks, i)
                        x_cord[i], y_cord[i] = int(x_cord[i]), int(y_cord[i])
                        px, py = int(hand_landmarks.landmark[i].x * frame.shape[1]), int(
                            hand_landmarks.landmark[i].y * frame.shape[0])

                        xList.append(px)
                        yList.append(py)

                    # Left hand functions

                    if label == "Left":
                        global prev_left_x, prev_left_y
                        if prev_left_x != x_cord[0] or prev_left_y != y_cord[0]:
                            smoothed_coords = smooth_movement((prev_left_x, prev_left_y),
(x_cord[8], y_cord[8]))
                            move_mouse_safely(smoothed_coords[0], smoothed_coords[1])
                            prev_left_x, prev_left_y = smoothed_coords

                    # Right hand functions

                    if label == "Right":

                        if xList[4] - xList[5] < 0:
                            click(yList[7], yList[8])
                            double_click(yList[11], yList[12])
                            left_click(yList[19], yList[20])
                            hold_click(yList[15], yList[16])

                            circle(frame, 8, (255, 160, 5), xList, yList)
                            circle(frame, 7, (255, 150, 5), xList, yList)

                            circle(frame, 11, (100, 255, 80), xList, yList)
                            circle(frame, 12, (100, 255, 80), xList, yList)

                            circle(frame, 20, (200, 213, 48), xList, yList)
                            circle(frame, 19, (200, 213, 48), xList, yList)

                            circle(frame, 16, (168, 156, 205), xList, yList)
                            circle(frame, 15, (168, 156, 205), xList, yList)

                        else:
                            back(yList[7], yList[8])
                            forward(yList[11], yList[12])
                            scroll(y_cord[16], y_cord[15])
                            mode_change(yList[17], yList[20])

                            circle(frame, 5, (255, 160, 5), xList, yList)
                            circle(frame, 4, (255, 160, 5), xList, yList)
                            circle(frame, 8, (255, 160, 5), xList, yList)
                            circle(frame, 7, (255, 160, 5), xList, yList)

                            circle(frame, 11, (100, 255, 80), xList, yList)
                            circle(frame, 12, (100, 255, 80), xList, yList)

                            circle(frame, 20, (200, 213, 48), xList, yList)
                            circle(frame, 17, (200, 213, 48), xList, yList)

                            circle(frame, 16, (168, 156, 205), xList, yList)
                            circle(frame, 15, (168, 156, 205), xList, yList)

                        prev_right_x[0], prev_right_x[1] = x_cord[4], x_cord[8]

        # Display the camera frame
        cv2.imshow("Camera", frame)

        if cv2.waitKey(1) & 0xFF == ord('Q') or not cv2.getWindowProperty("Camera",
```

```
cv2.WND_PROP_VISIBLE):
            break

    cap.release()
    cv2.destroyAllWindows()


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