Vizieye- A Multi Model Personal AI Assistant

Deep Neural Networks

Submitted by

Dhruv Kajla, Mukund, Navroop, Shail Sharma

2210993781, 2210993818, 2210993719, 2210993837

Semester: 5th

BE-CSE (Artificial Intelligence)

Submitted To

Mr. Vivek Parmar

Prof , Department of CSE(AI),

CUIET, Chitkara University

CHITKARA UNIVERSITY INSITUTE OF ENGINEERING & TECHNOLOGY

CHITKARA UNIVERSITY, RAJPURA

December, 2024

Table of Contents

1.Introduction

In the modern era, technology is evolving at an unprecedented pace, influencing every aspect of our lives, from education and healthcare to entertainment and personal assistance. One such innovation is **Vizieye**, a multi-modal personal assistant designed to empower users through advanced gesture recognition, voice commands, and object detection. Developed using deep neural networks (DNNs) and cutting-edge tools like Python, Flask, and OpenCV, Vizieye bridges the gap between human interaction and machine efficiency.

The motivation behind creating Vizieye stems from the challenges faced by individuals in seamlessly interacting with technology. With the increasing reliance on digital systems, there is a growing demand for intuitive interfaces that accommodate diverse user needs, including those of people with disabilities. By integrating gesture recognition, voice control, and real-time object detection, Vizieye provides a hands-free, efficient, and interactive experience, redefining how users engage with technology.

The global education and professional landscape often highlight the importance of accessibility and efficiency in technology. Students and professionals alike can benefit from tools that save time, improve productivity, and simplify routine tasks. However, existing solutions often fall short of addressing the needs of individuals who require a multi-modal interaction platform. Vizieye fills this gap by offering an innovative, adaptable, and user-friendly interface, ensuring inclusivity and enhanced usability.

The project emphasizes the seamless fusion of multiple domains, including computer vision, natural language processing, and machine learning, to deliver a comprehensive personal assistant. While the primary focus is on gesture recognition and voice commands, the ongoing development of an object detection module further enhances its capabilities. These features not only cater to accessibility needs but also improve task automation, making Vizieye an indispensable tool for users in various contexts.

Through a robust and scalable architecture, Vizieye leverages Python for model implementation, Flask for backend management, and tools like OpenCV and Mediapipe for gesture and object tracking. The integration of speech recognition ensures accurate and efficient voice command execution, creating a versatile platform that adapts to diverse user preferences.

As the world moves towards intelligent systems, projects like Vizieye exemplify the potential of multi-modal interaction technologies in improving daily life. This report delves into the methodology, tools, and outcomes of Vizieye, presenting a roadmap for future enhancements and exploring its role in creating a more accessible and interactive technological ecosystem.

2.Methodology

The **Vizieye** project employs a user-focused approach to create a multi-modal personal assistant that integrates gesture recognition, voice commands, and object detection. The methodology ensures seamless interaction between the user and technology, offering intuitive features designed for accessibility and efficiency.

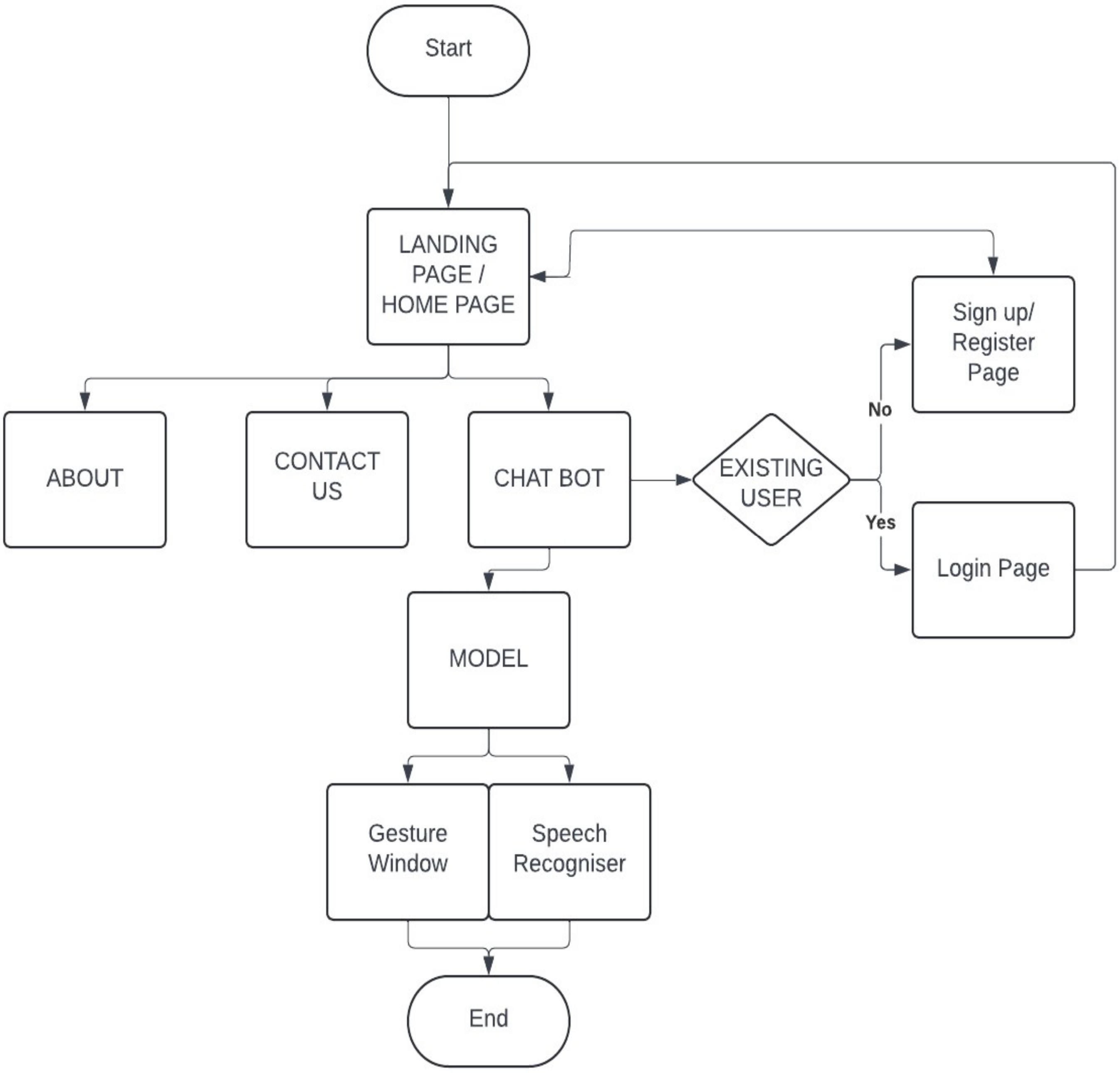
Core Functionalities

1. **Gesture Recognition**:
    - Leveraging tools like OpenCV and Mediapipe, the system detects and interprets hand gestures for tasks such as controlling media, adjusting system settings, and navigating applications.
    - This enables hands-free operation, enhancing user convenience and accessibility.
2. **Voice Commands**:
    - Using speech recognition and text-to-speech technology, Vizieye accurately processes spoken instructions, enabling users to perform tasks like opening applications, searching for information, and system management.
    - This functionality is integrated with Flask to ensure smooth backend processing.
3. **Object Detection**:
    - The project is being extended to include object detection, helping users identify objects in real time.
    - This feature enhances usability in scenarios like navigating environments or identifying items.

System Design

- **Backend Development**: Built using Flask, the system processes commands and integrates responses from different modules efficiently.
- **Multi-modal Input**: The platform combines visual and auditory inputs for versatile interaction, catering to diverse user needs.
- **Real-Time Feedback**: The system provides immediate responses to commands, ensuring a smooth and interactive user experience

3.FlowChart


Start
LANDING PAGE / HOME PAGE
Sign up/ Register Page
ABOUT
CONTACT US
CHAT BOT
EXISTING USER
No
Yes
Login Page
MODEL
Gesture Window
Speech Recogniser
End

4.Tools and Technologies

 Python: A high-level programming language used for implementing core functionalities like gesture recognition, voice commands, and object detection**.**

 Flask: A lightweight web framework used for creating the backend to process user inputs and integrate various modules.

 OpenCV: An open-source computer vision library for real-time image processing and gesture tracking**.**

 Mediapipe: A framework by Google for building multimodal interaction pipelines, used for hand and face gesture detection**.**

 Pycaw: A Python library for controlling system audio, enabling volume adjustment via gestures.

 SpeechRecognition: A library for converting spoken language into text to enable voice **command processing.**

 Text-to-Speech (pyttsx3): A Python library used to generate speech feedback for users in response to voice commands.

 TensorFlow/Keras: Machine learning frameworks used for training and implementing deep learning models like object detection**.**

 Numpy: A Python library for numerical computations, aiding in image processing and model implementation.

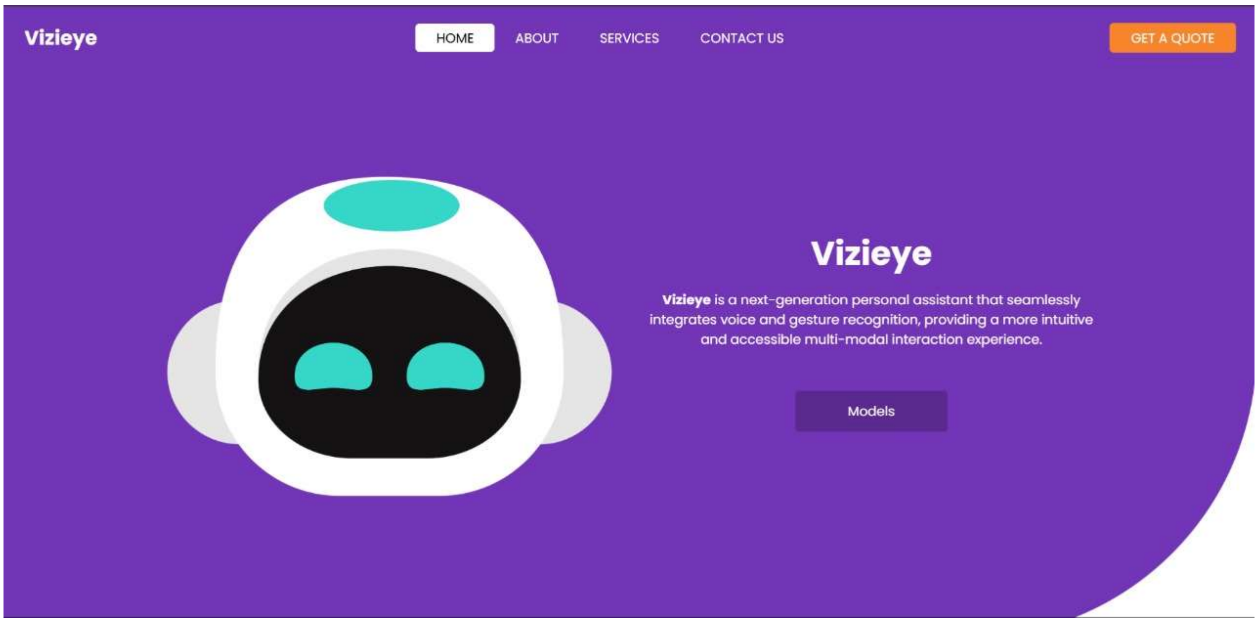
 Matplotlib: A plotting library for visualizing data during testing and debugging.
  VS Code: An integrated development environment (IDE) used for writing and debugging code.

 Webcam: Used as a hardware input device for capturing real-time gestures and object images.

 Microphone: Hardware for capturing voice inputs for speech recognition functionality.

  Jupyter Notebook: Used for testing machine learning models and debugging code interactively.

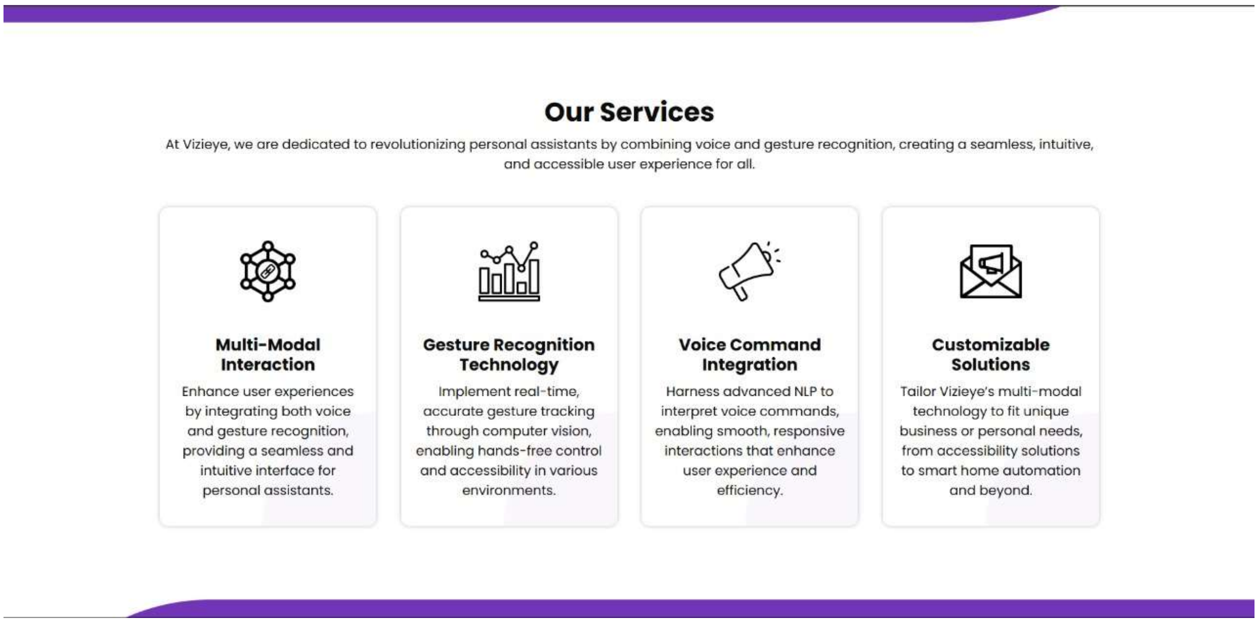
5.Major Findings/Outcomes/Output/Results



Vizieye

Vizieye is a next-generation personal assistant that seamlessly integrates voice and gesture recognition, providing a more intuitive and accessible multi-modal interaction experience.

Fig: - 1



Our Services

At Vizieye, we are dedicated to revolutionizing personal assistants by combining voice and gesture recognition, creating a seamless, intuitive, and accessible user experience for all.

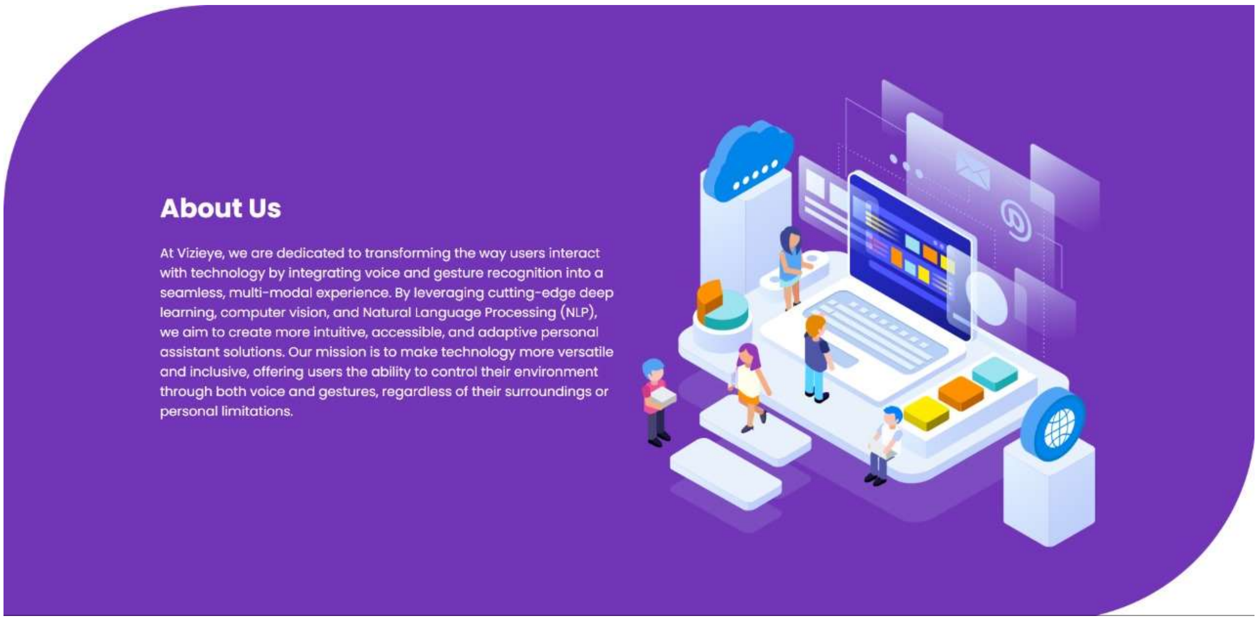
Multi-Modal Interaction
Enhance user experiences by integrating both voice and gesture recognition, providing a seamless and intuitive interface for personal assistants.

Gesture Recognition Technology
Implement real-time, accurate gesture tracking through computer vision, enabling hands-free control and accessibility in various environments.

Voice Command Integration
Harness advanced NLP to interpret voice commands, enabling smooth, responsive interactions that enhance user experience and efficiency.

Customizable Solutions
Tailor Vizieye's multi-modal technology to fit unique business or personal needs, from accessibility solutions to smart home automation and beyond.

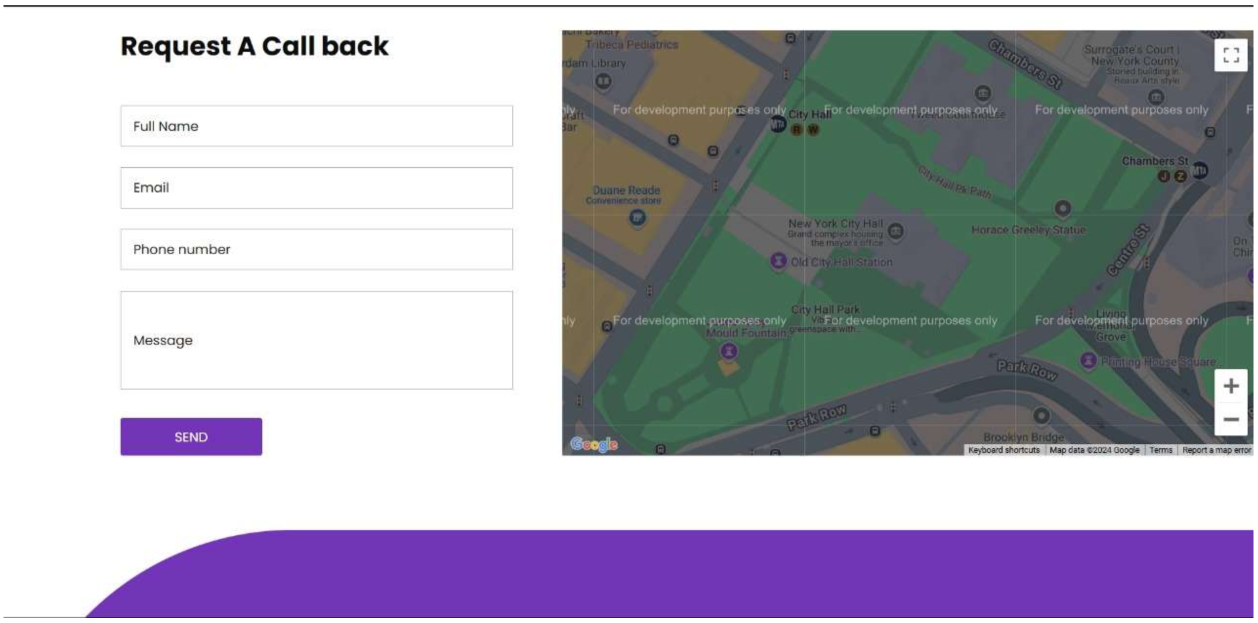
Fig: - 2



About Us

At Vizieye, we are dedicated to transforming the way users interact with technology by integrating voice and gesture recognition into a seamless, multi-modal experience. By leveraging cutting-edge deep learning, computer vision, and Natural Language Processing (NLP), we aim to create more intuitive, accessible, and adaptive personal assistant solutions. Our mission is to make technology more versatile and inclusive, offering users the ability to control their environment through both voice and gestures, regardless of their surroundings or personal limitations.

Fig: - 3



Request A Call back

Full Name

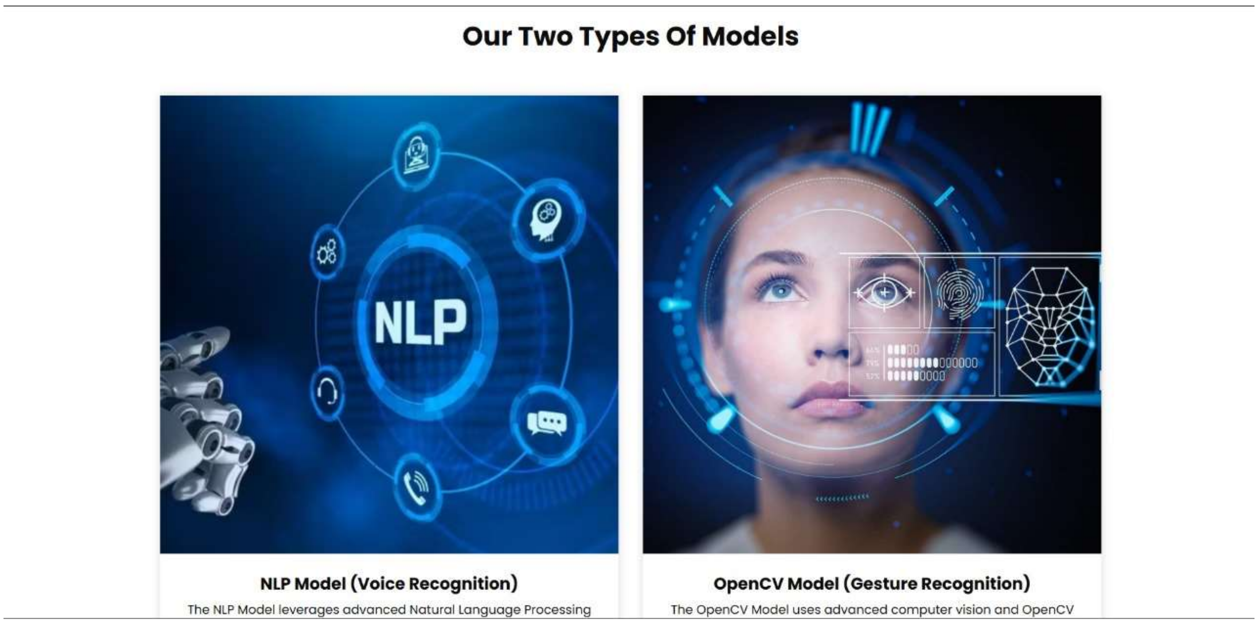
Email

Phone number

Message

SEND

Fig: - 4



Our Two Types Of Models

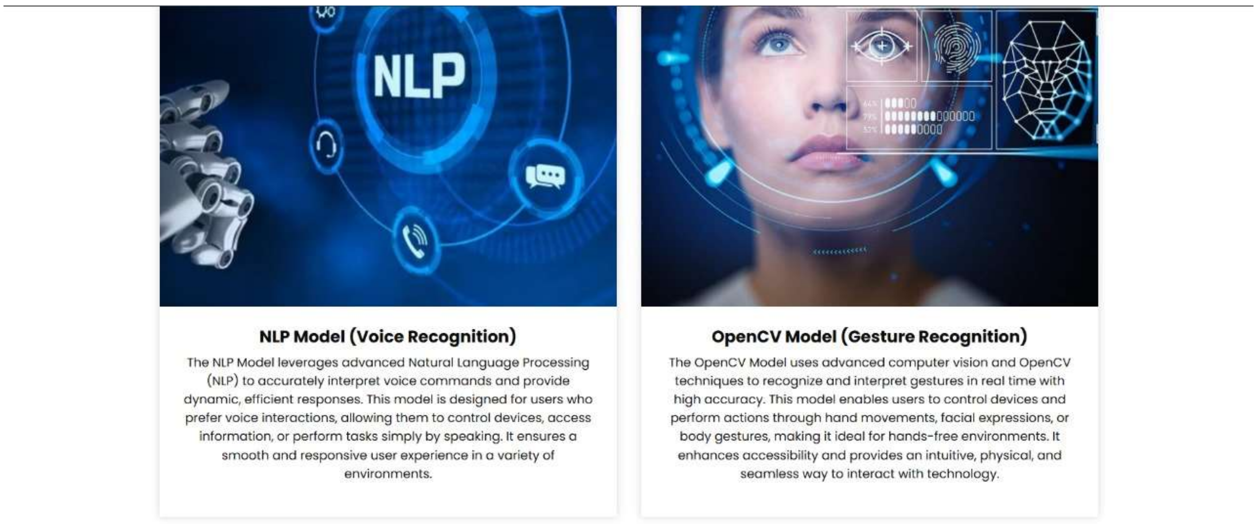
NLP Model (Voice Recognition)

The NLP Model leverages advanced Natural Language Processing

OpenCV Model (Gesture Recognition)

The OpenCV Model uses advanced computer vision and OpenCV

Fig: - 5



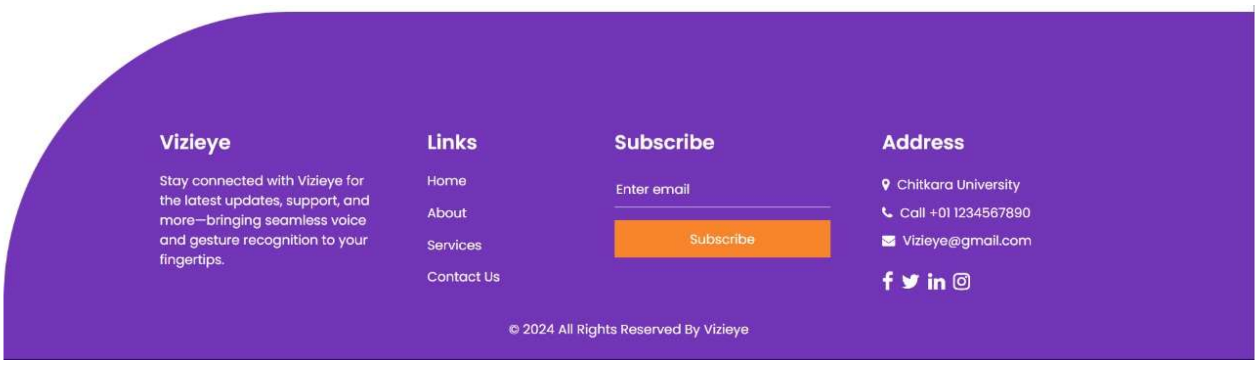
NLP Model (Voice Recognition)

The NLP Model leverages advanced Natural Language Processing (NLP) to accurately interpret voice commands and provide dynamic, efficient responses. This model is designed for users who prefer voice interactions, allowing them to control devices, access information, or perform tasks simply by speaking. It ensures a smooth and responsive user experience in a variety of environments.

OpenCV Model (Gesture Recognition)

The OpenCV Model uses advanced computer vision and OpenCV techniques to recognize and interpret gestures in real time with high accuracy. This model enables users to control devices and perform actions through hand movements, facial expressions, or body gestures, making it ideal for hands-free environments. It enhances accessibility and provides an intuitive, physical, and seamless way to interact with technology.

Fig: - 6


Vizieye

Stay connected with Vizieye for the latest updates, support, and more—bringing seamless voice and gesture recognition to your fingertips.

Links

Home
About
Services
Contact Us

Subscribe

Enter email

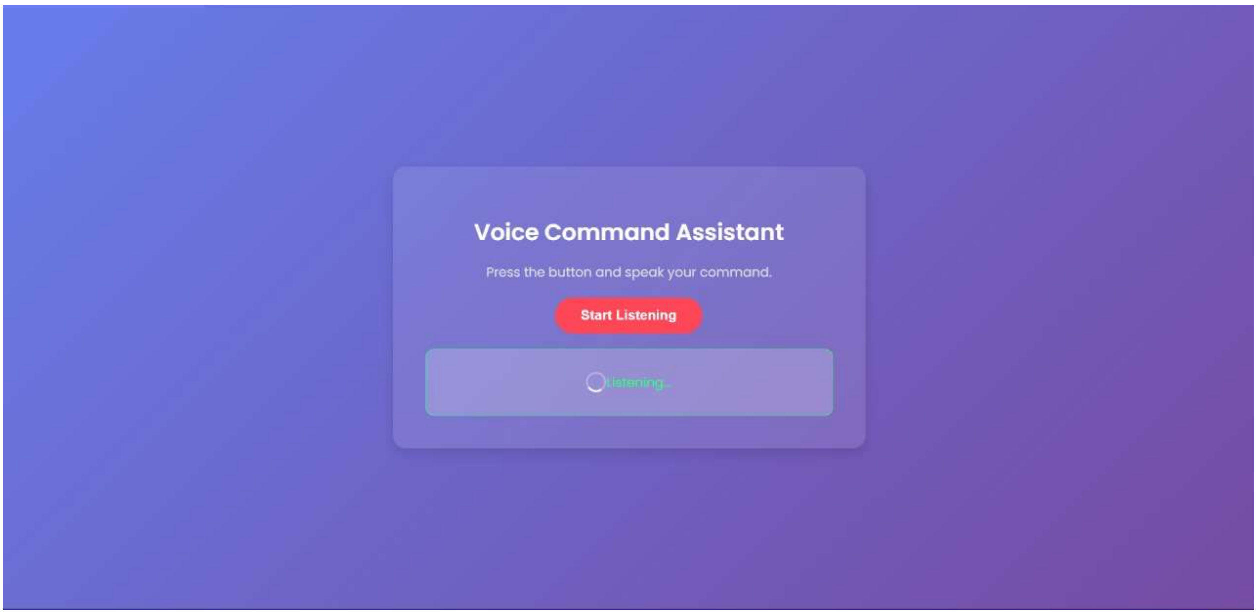
Subscribe

Address

Chitkara University
Call +01 1234567890
Vizieye@gmail.com

© 2024 All Rights Reserved By Vizieye

Fig: - 7


Voice Command Assistant

Press the button and speak your command.

Start Listening

Listening...

Fig: - 8

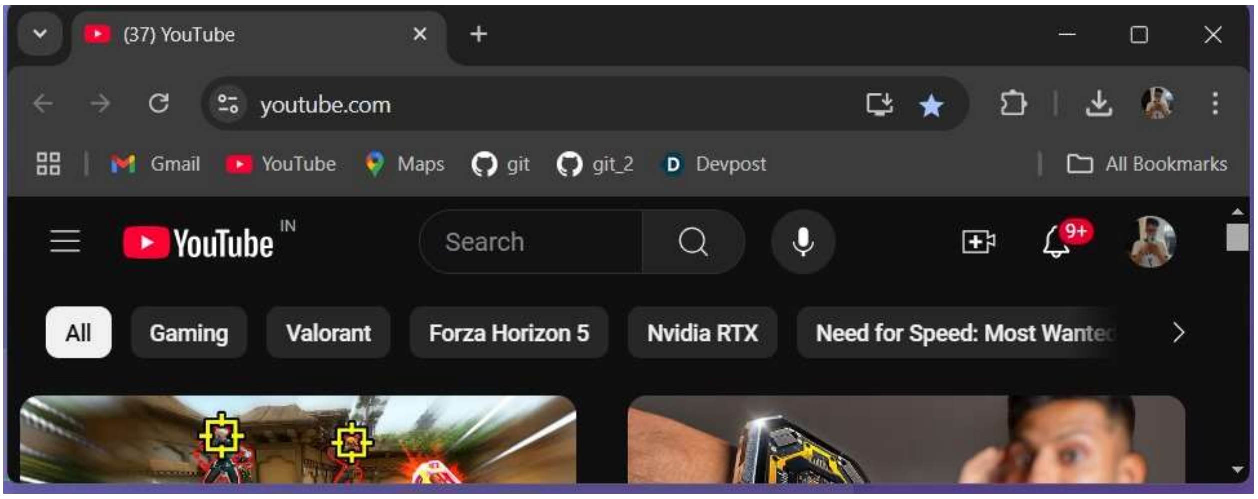
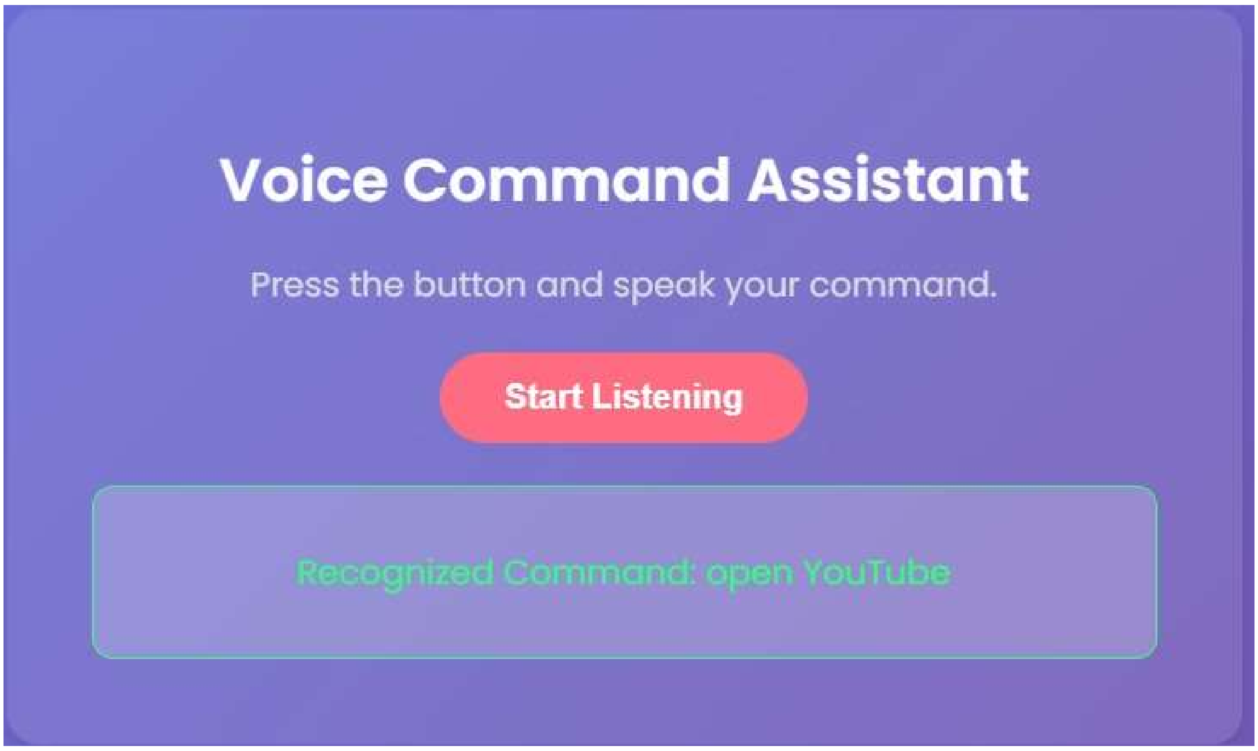


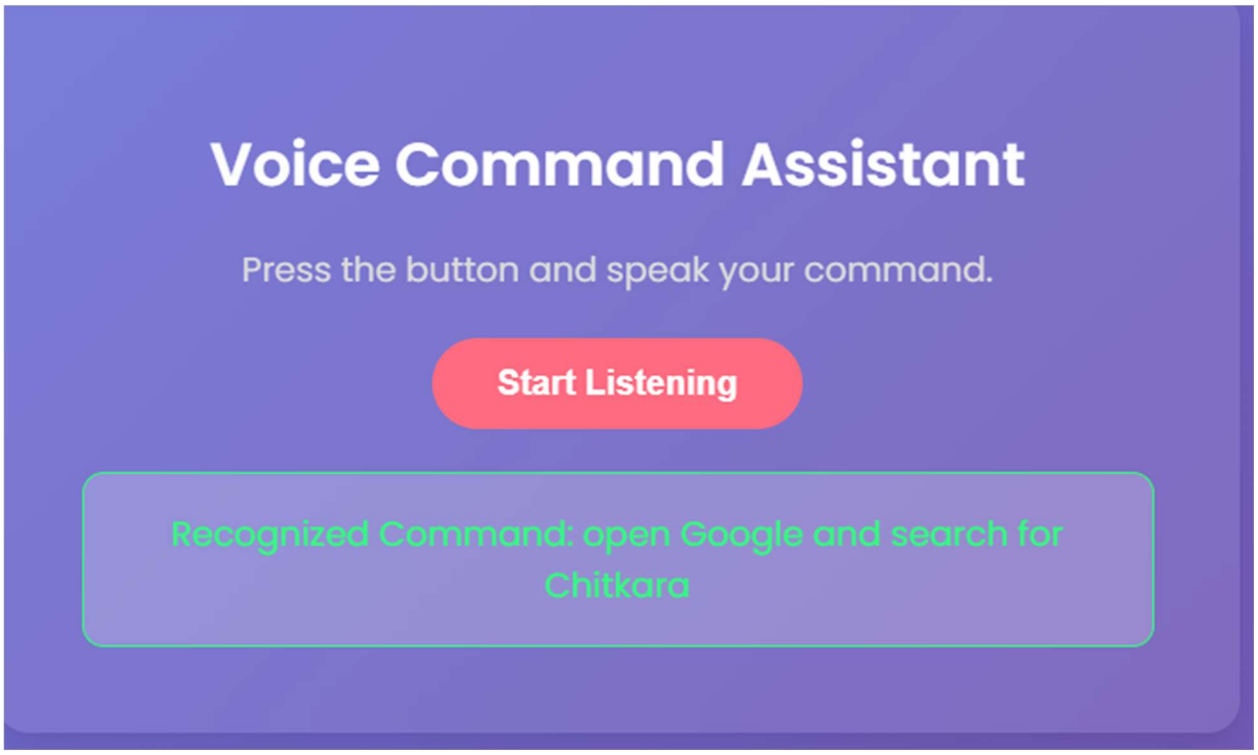
Fig: - 9


Voice Command Assistant

Press the button and speak your command.

Recognized Command: open YouTube

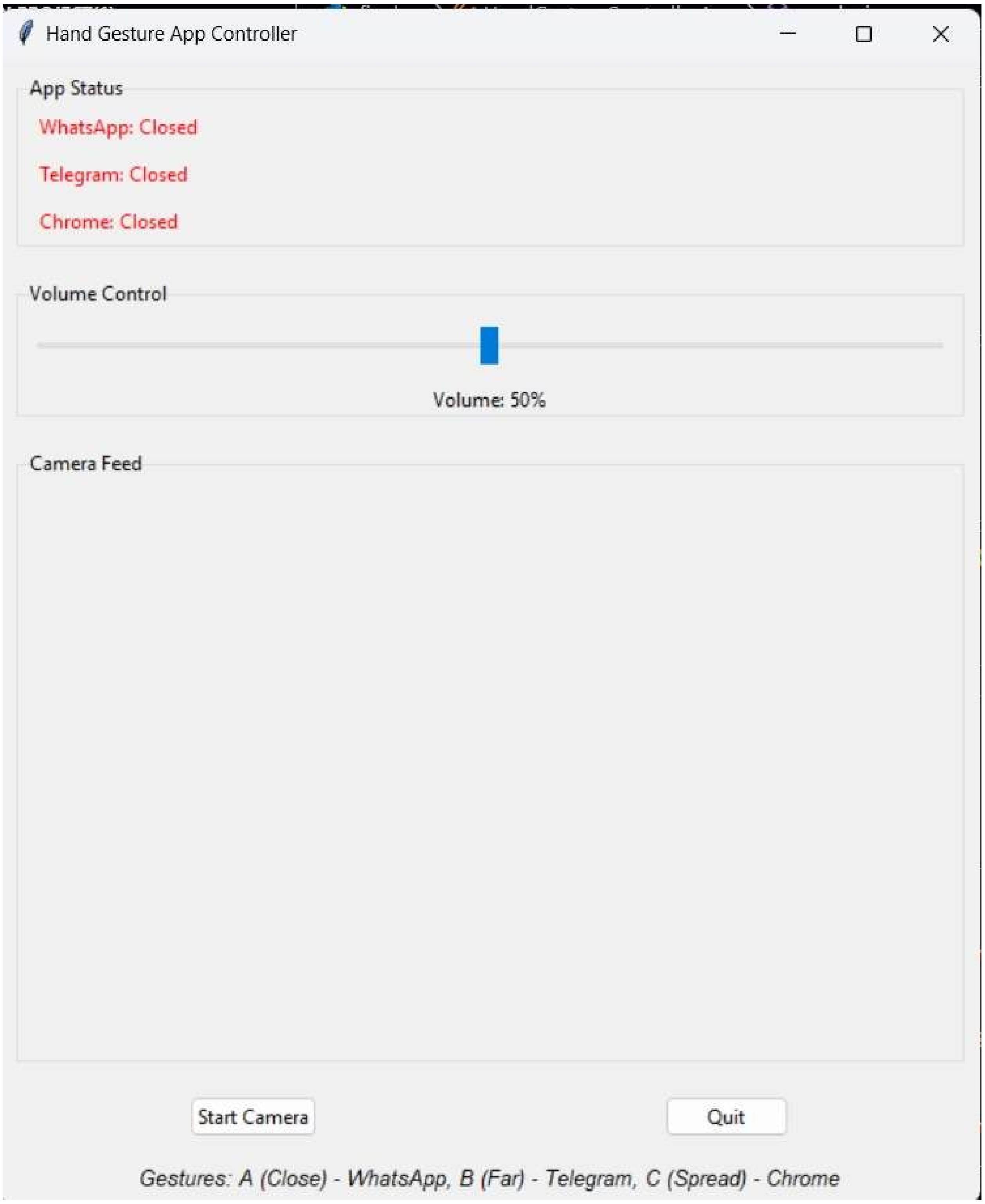
Fig: - 10


Voice Command Assistant

Press the button and speak your command.

Recognized Command: open Google and search for Chitkara

Fig: - 11


Hand Gesture App Controller

App Status

WhatsApp: Closed

Telegram: Closed

Chrome: Closed

Volume Control

Volume: 50%

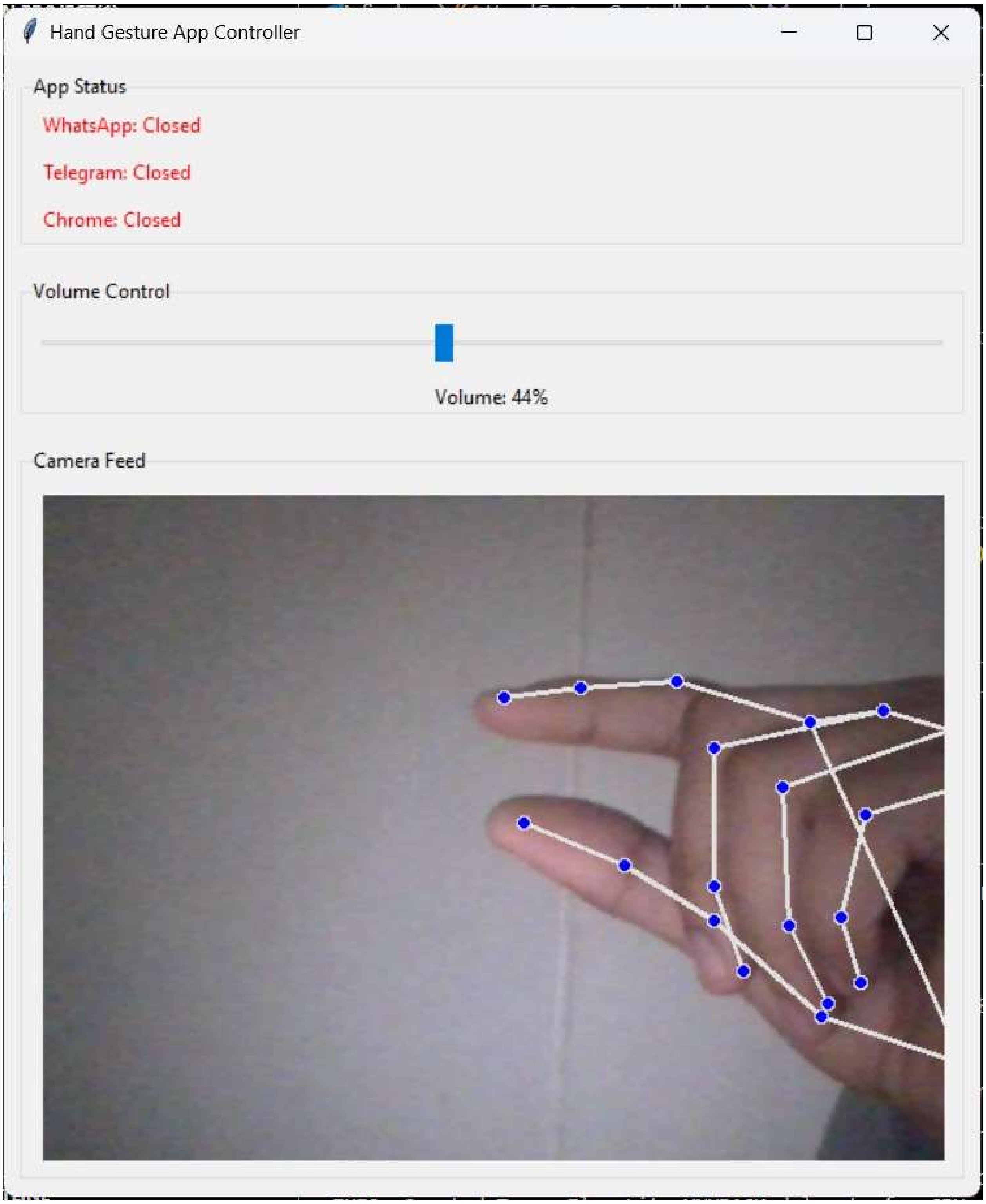
Camera Feed

Start Camera

Quit

Gestures: A (Close) - WhatsApp, B (Far) - Telegram, C (Spread) - Chrome

Fig: - 12



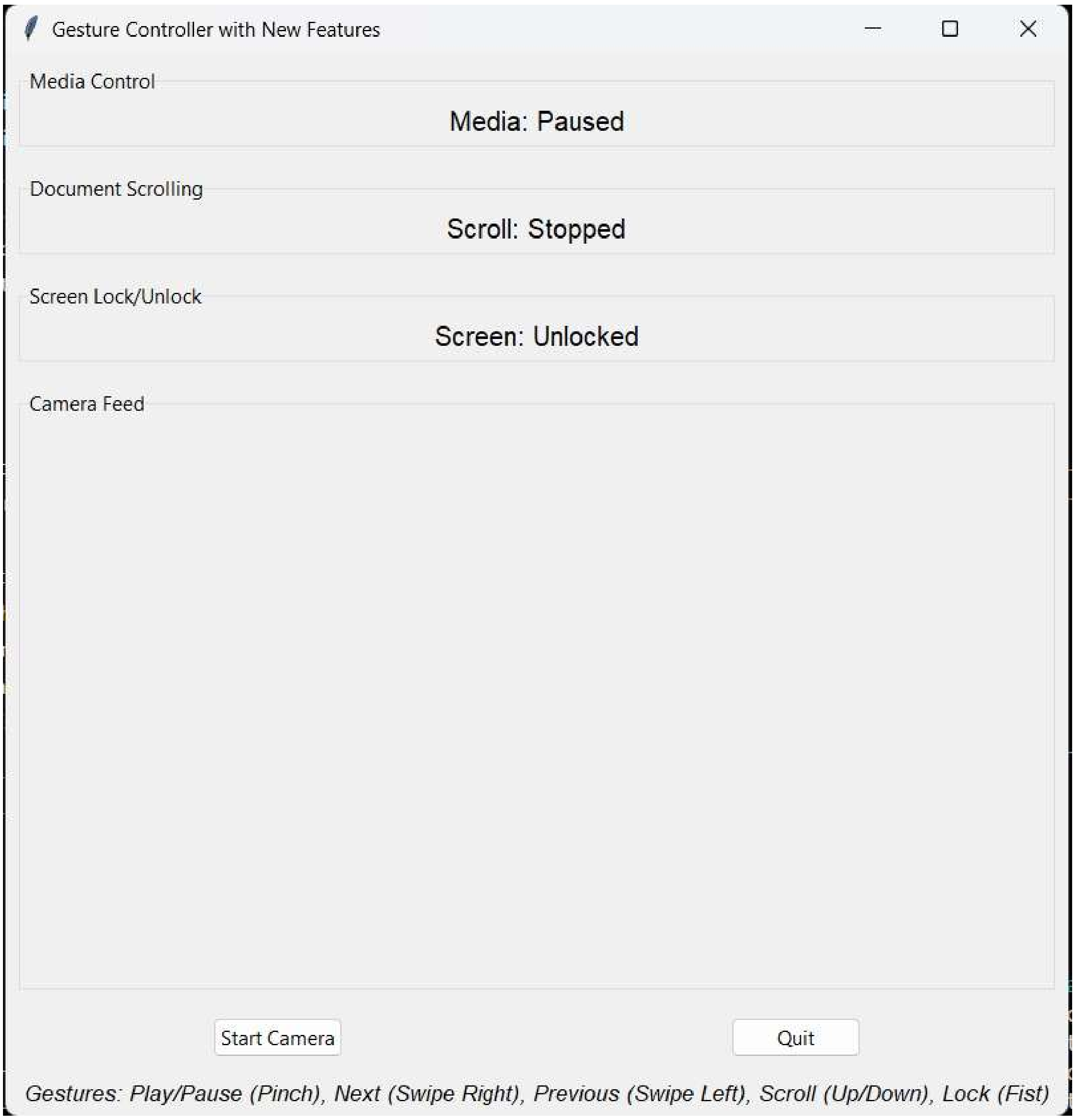
WhatsApp: Closed

Telegram: Closed

Chrome: Closed

Volume: 44%

Fig: - 13


Gesture Controller with New Features

Media Control

Media: Paused

Document Scrolling

Scroll: Stopped

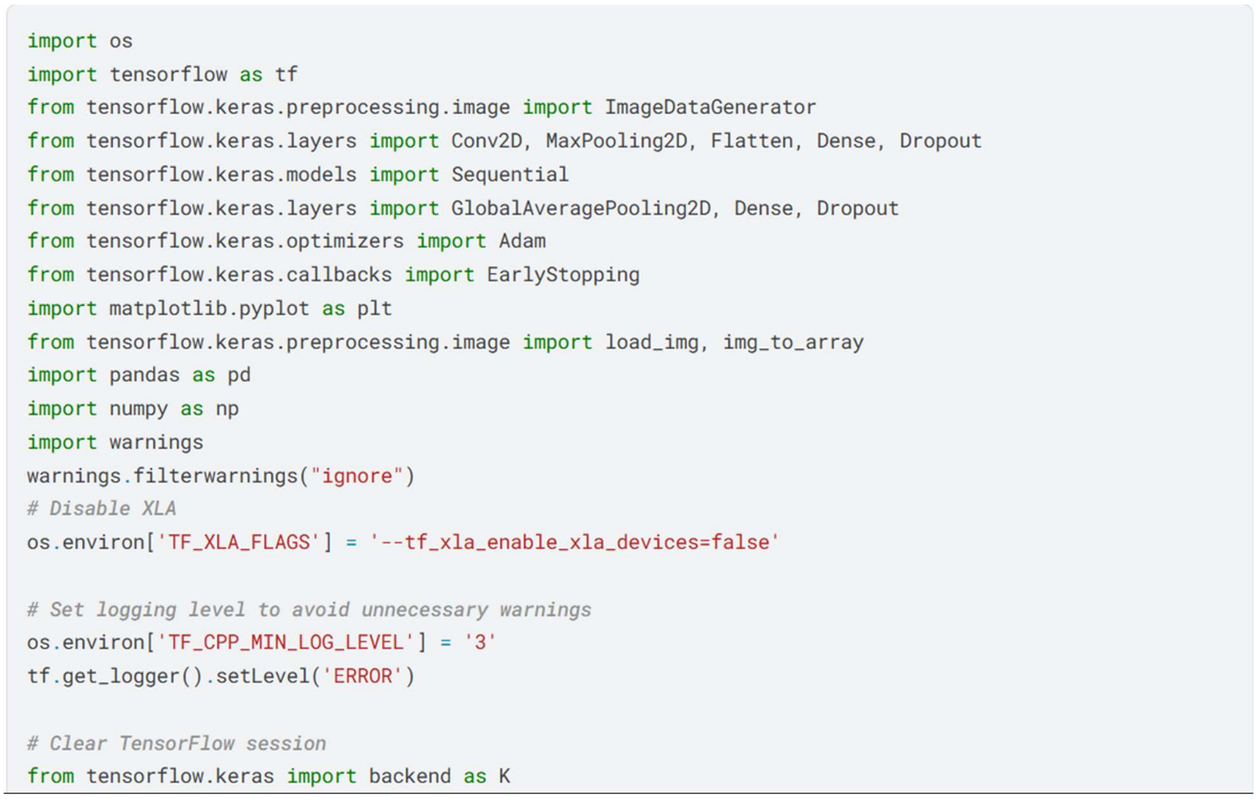
Screen Lock/Unlock

Screen: Unlocked

Camera Feed

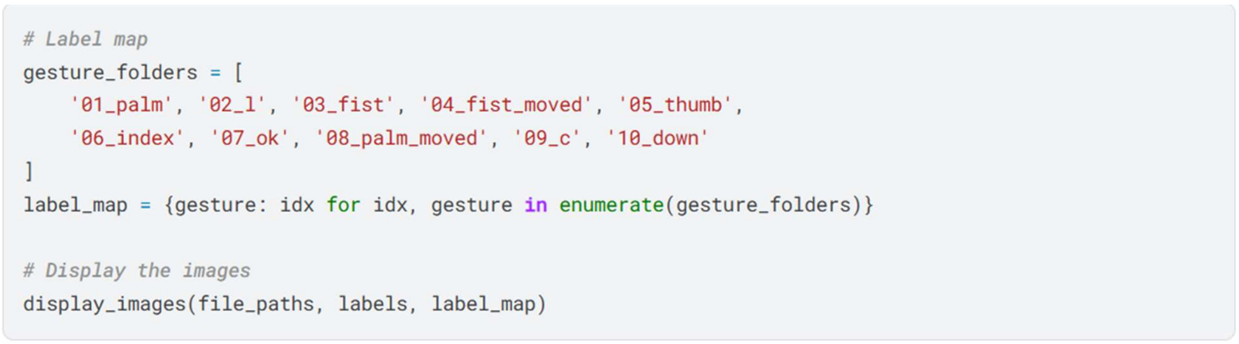
Gestures: Play/Pause (Pinch), Next (Swipe Right), Previous (Swipe Left), Scroll (Up/Down), Lock (Fist)

Fig: - 14



```python
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
# Disable XLA
os.environ['TF_XLA_FLAGS'] = '--tf_xla_enable_xla_devices=false'

# Set logging level to avoid unnecessary warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
tf.get_logger().setLevel('ERROR')

# Clear TensorFlow session
from tensorflow.keras import backend as K
```

Fig: - 15



```python
# Label map
gesture_folders = [
    '01_palm', '02_l', '03_fist', '04_fist_moved', '05_thumb',
    '06_index', '07_ok', '08_palm_moved', '09_c', '10_down'
]
label_map = {gesture: idx for idx, gesture in enumerate(gesture_folders)}

# Display the images
display_images(file_paths, labels, label_map)
```

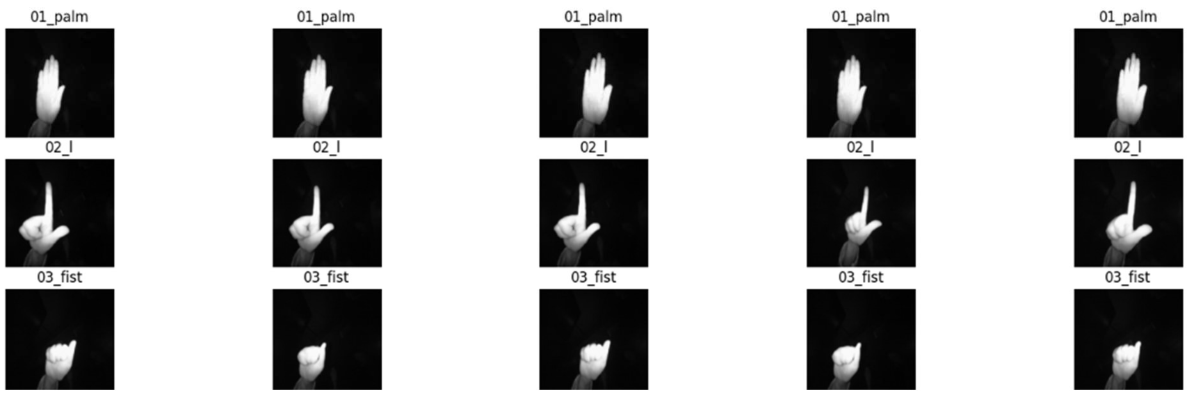
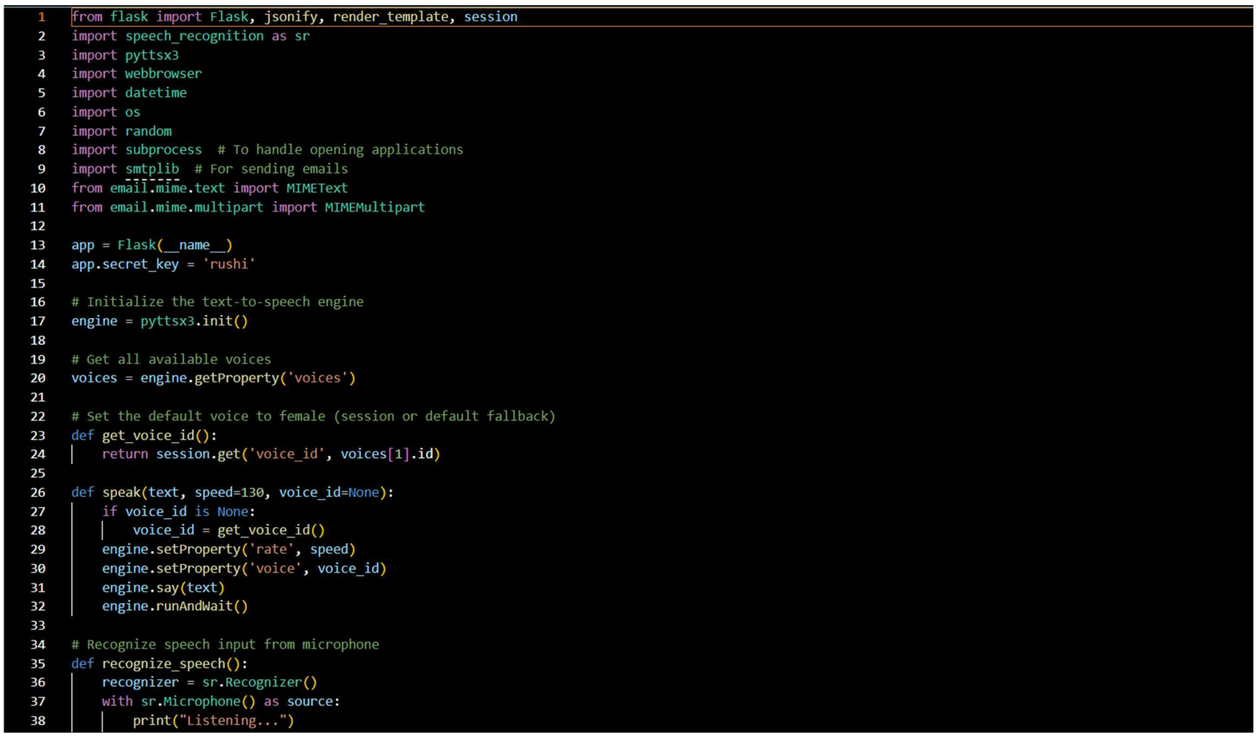
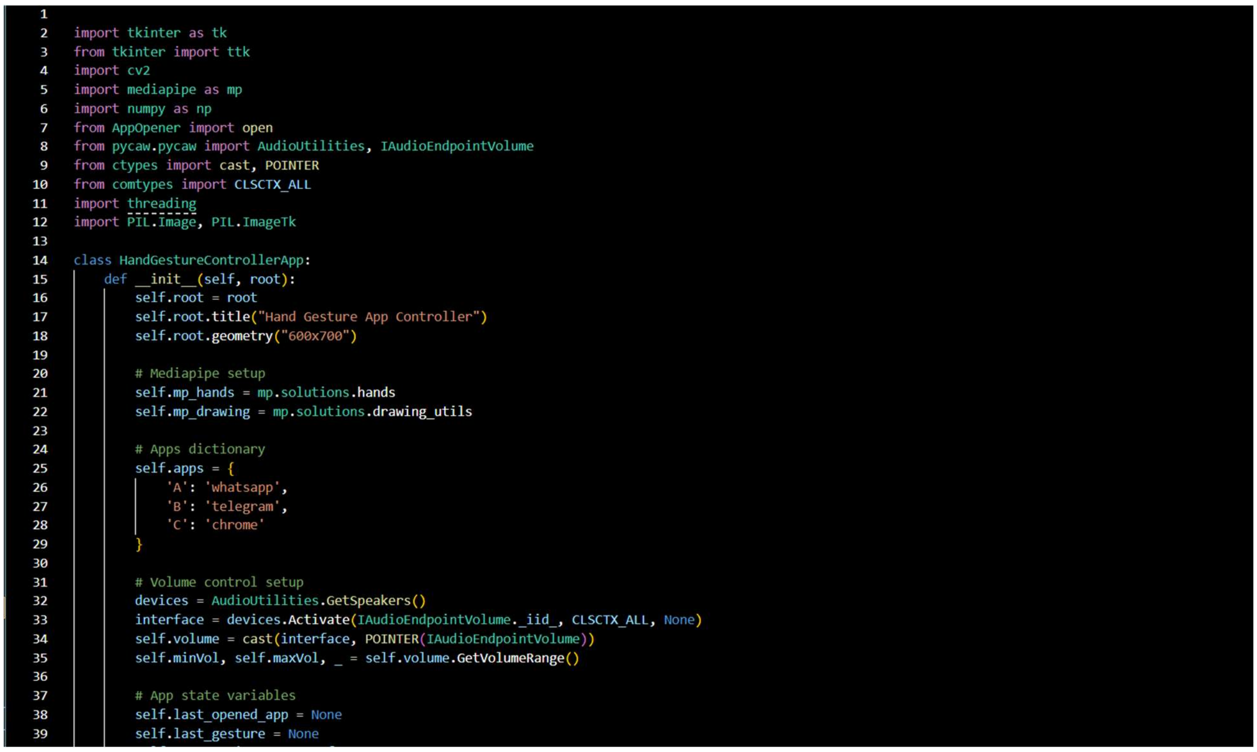


Fig: - 16



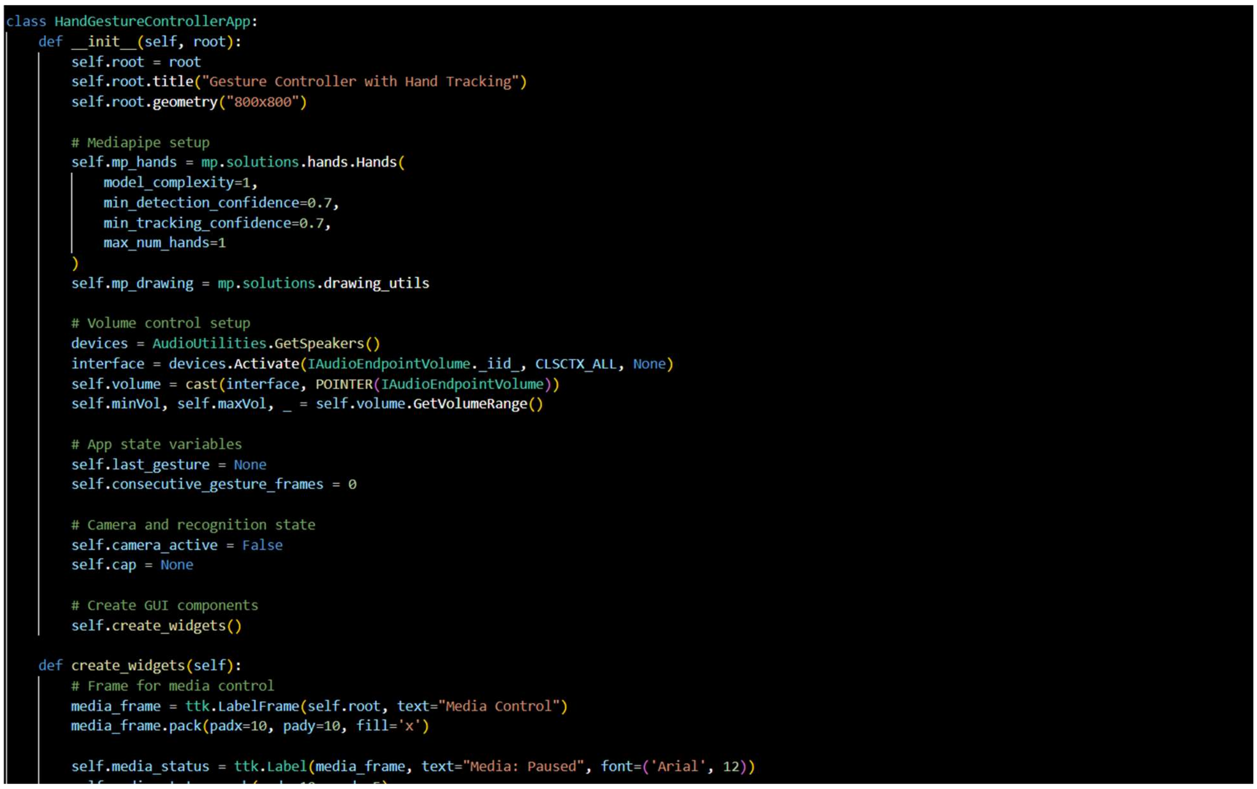
```python
from flask import Flask, jsonify, render_template, session
import speech_recognition as sr
import pyttsx3
import webbrowser
import datetime
import os
import random
import subprocess  # To handle opening applications
import smtplib  # For sending emails
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

app = Flask(__name__)
app.secret_key = 'rushi'

# Initialize the text-to-speech engine
engine = pyttsx3.init()

# Get all available voices
voices = engine.getProperty('voices')

# Set the default voice to female (session or default fallback)
def get_voice_id():
    return session.get('voice_id', voices[1].id)

def speak(text, speed=130, voice_id=None):
    if voice_id is None:
        voice_id = get_voice_id()
    engine.setProperty('rate', speed)
    engine.setProperty('voice', voice_id)
    engine.say(text)
    engine.runAndWait()

# Recognize speech input from microphone
def recognize_speech():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening...")
```

Fig: - 17



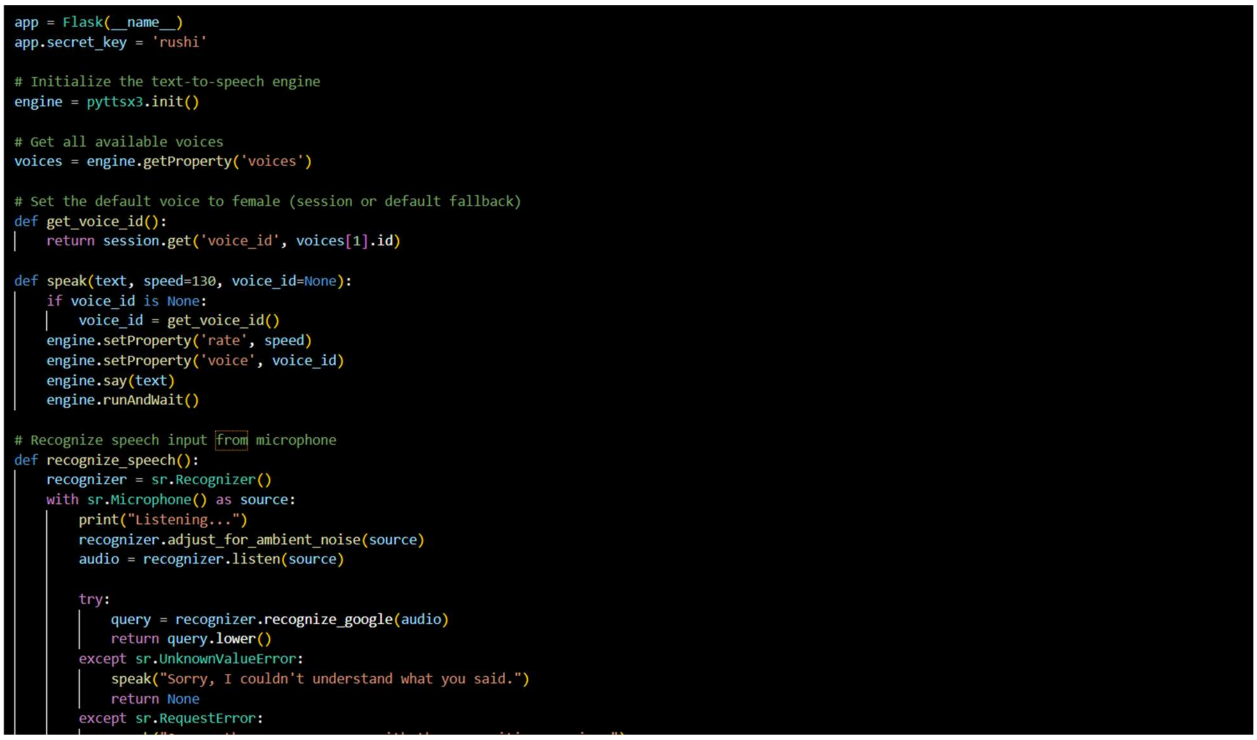
```python

import tkinter as tk
from tkinter import ttk
import cv2
import mediapipe as mp
import numpy as np
from AppOpener import open
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
import threading
import PIL.Image, PIL.ImageTk

class HandGestureControllerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Hand Gesture App Controller")
        self.root.geometry("600x700")

        # Mediapipe setup
        self.mp_hands = mp.solutions.hands
        self.mp_drawing = mp.solutions.drawing_utils

        # Apps dictionary
        self.apps = {
            'A': 'whatsapp',
            'B': 'telegram',
            'C': 'chrome'
        }

        # Volume control setup
        devices = AudioUtilities.GetSpeakers()
        interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
        self.volume = cast(interface, POINTER(IAudioEndpointVolume))
        self.minVol, self.maxVol, _ = self.volume.GetVolumeRange()

        # App state variables
        self.last_opened_app = None
        self.last_gesture = None
```

Fig: - 18



```python
class HandGestureControllerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Gesture Controller with Hand Tracking")
        self.root.geometry("800x800")

        # Mediapipe setup
        self.mp_hands = mp.solutions.hands.Hands(
            model_complexity=1,
            min_detection_confidence=0.7,
            min_tracking_confidence=0.7,
            max_num_hands=1
        )
        self.mp_drawing = mp.solutions.drawing_utils

        # Volume control setup
        devices = AudioUtilities.GetSpeakers()
        interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
        self.volume = cast(interface, POINTER(IAudioEndpointVolume))
        self.minVol, self.maxVol, _ = self.volume.GetVolumeRange()

        # App state variables
        self.last_gesture = None
        self.consecutive_gesture_frames = 0

        # Camera and recognition state
        self.camera_active = False
        self.cap = None

        # Create GUI components
        self.create_widgets()

    def create_widgets(self):
        # Frame for media control
        media_frame = ttk.LabelFrame(self.root, text="Media Control")
        media_frame.pack(padx=10, pady=10, fill='x')

        self.media_status = ttk.Label(media_frame, text="Media: Paused", font=('Arial', 12))
```

Fig: - 1 9



```python
app = Flask(__name__)
app.secret_key = 'rushi'

# Initialize the text-to-speech engine
engine = pyttsx3.init()

# Get all available voices
voices = engine.getProperty('voices')

# Set the default voice to female (session or default fallback)
def get_voice_id():
    return session.get('voice_id', voices[1].id)

def speak(text, speed=130, voice_id=None):
    if voice_id is None:
        voice_id = get_voice_id()
    engine.setProperty('rate', speed)
    engine.setProperty('voice', voice_id)
    engine.say(text)
    engine.runAndWait()

# Recognize speech input from microphone
def recognize_speech():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening...")
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.listen(source)

        try:
            query = recognizer.recognize_google(audio)
            return query.lower()
        except sr.UnknownValueError:
            speak("Sorry, I couldn't understand what you said.")
            return None
        except sr.RequestError:
```

Fig: - 20

6.Conclusion

Vizieye is a groundbreaking project that leverages the power of computer vision and augmented reality (AR) to create an innovative, immersive user experience. By integrating real-time object recognition, advanced image processing, and AR, Vizieye enables users to interact with their surroundings in a way that was previously reserved for science fiction. The key strength of Vizieye lies in its ability to identify and interact with objects in real time, offering a multitude of possibilities for both personal and professional applications.

The primary potential of Vizieye spans across multiple industries, each of which can benefit from its unique combination of computer vision and AR. For example:

1. **Education**: In the educational sector, Vizieye can be used to create interactive learning experiences, helping students visualize complex concepts, such as scientific processes, geographical features, and historical events, in an engaging and dynamic manner.
2. **Retail and E-commerce**: Vizieye can be used in retail environments to create virtual try-ons or provide customers with additional product details through AR. This can drive consumer engagement and improve purchasing decisions, as users can see how products look or work in their own space before buying them.
3. **Gaming and Entertainment**: Augmented reality games can use Vizieye to track and recognize real-world objects, merging the digital and physical realms to create immersive gameplay experiences. The potential for interactive storytelling and dynamic, location-based experiences is vast.
4. **Accessibility**: For individuals with visual impairments or other disabilities, Vizieye can serve as a powerful assistive tool. With object recognition and real-time feedback, it can help users navigate environments, identify objects, and even read text, making the world more accessible.
5. **Smart Environments**: Vizieye can play a key role in the development of smart homes and cities, where real-time object tracking and environmental awareness are essential. For example, it could help optimize energy consumption by detecting presence and activity within a space or assist in surveillance and security through intelligent monitoring systems.

The integration of AI and AR into Vizieye brings a level of sophistication that enables seamless interaction with the digital world. However, as with all emerging technologies, there are areas for improvement. Enhancing the accuracy, speed, and adaptability of the object recognition system will be crucial for real-world applications, especially in complex, fast-moving environments. Furthermore, addressing challenges such as ensuring data privacy, optimizing performance on various hardware platforms, and scaling the solution to handle large volumes of data will be important as the project evolves.

In conclusion, Vizieye is not just a technological advancement; it’s a tool that has the potential to transform the way we interact with both our physical and digital worlds. As the project matures and its capabilities grow, it could revolutionize industries, improve accessibility, and redefine user experiences in an increasingly connected and digital world. The future of Vizieye is filled with exciting possibilities, as it stands at the intersection of cutting-edge AI, AR, and human interaction, making technology more accessible, intuitive, and impactful for everyone

7. Future Scope

Vizieye, combining computer vision and augmented reality (AR), has vast potential for growth across various sectors. Here are key areas of its future development:

1. **Enhanced Object Recognition**: Future advancements in AI could lead to even more accurate and context-aware object recognition, allowing Vizieye to understand complex environments and user interactions better.
2. **Healthcare Applications**: Vizieye could be used in medical diagnostics, real-time surgeries, and health monitoring, offering AR overlays and precise object tracking for healthcare professionals.
3. **Gaming and AR**: Vizieye could enhance gaming experiences by enabling real-world object interaction and providing mixed-reality gameplay, making digital and physical spaces merge seamlessly.
4. **Smart Cities**: In urban development, Vizieye could play a key role in monitoring infrastructure, improving public safety, and supporting smart city applications like energy optimization and intelligent transportation systems.
5. **Retail and E-commerce**: Vizieye could transform online and in-store shopping with interactive AR try-ons, personalized product recommendations, and a more immersive customer experience.
6. **Education**: Vizieye could enhance learning by creating interactive, AR-based educational tools that make complex topics like science, history, and geography more engaging and understandable.
7. **Accessibility**: Vizieye has the potential to assist individuals with disabilities by providing real-time translation, navigation aids, and object recognition to help them interact with the world more easily.
8. **Cross-Platform Integration**: The future of Vizieye includes seamless integration across multiple devices like smartphones, wearables, and AR glasses, making it more accessible and versatile for different use cases.
9. **Personalization**: By learning from user behaviors, Vizieye could offer personalized experiences in real-time, adapting its responses and interactions to suit individual preferences.
10. **Privacy and Security**: Addressing privacy concerns and ensuring secure data handling will be crucial as Vizieye scales and collects more personal data in sensitive applications like healthcare and public safety.

As Vizieye continues to evolve, it holds the potential to transform industries, enhance user experiences, and create new possibilities for real-time, interactive applications. The future of Vizieye is rich with opportunities, but it will require ongoing advancements in AI, scalability, and ethical considerations.

8.References

Abadi, M., et al. (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Retrieved from https://www.tensorflow.org

Bradski, G. (2000). *The OpenCV Library*. Dr. Dobb's Journal of Software Tools. Retrieved from https://opencv.org

Google Developers. (2023). *Mediapipe: Cross-Platform Framework for Building Perception Pipelines*. Retrieved from https://google.github.io/mediapipe
Flask Documentation. (2023). *Flask: A Microframework for Python*. Retrieved from https://flask.palletsprojects.com

Python Software Foundation. (2023). *Python Programming Language*. Retrieved from https://www.python.org

Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement*. Retrieved from https://arxiv.org/abs/1804.02767

Dutta, A., et al. (2022). *Human-Computer Interaction Using Gesture Recognition and Voice Commands*. International Journal of Computer Vision Research.

Microsoft Developers. (2023). *Pycaw: Python Core Audio Wrapper*. Retrieved from https://github.com/AndreMiras/pycaw

Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, pp. 2825–2830.

OpenAI. (2023). *Advanced Conversational AI Models*. Retrieved from https://www.openai.com

9. Appendix

 Code Snippets:

- Gesture recognition code using Mediapipe and OpenCV.
- Voice command recognition code using the SpeechRecognition library.

 User Interface Design:

- Description of the main interface with webcam live feed and voice input section.
- Additional sections including settings and help guides.

 System Operation Flowchart:

- User input (gesture or voice recognition).
- Processing the input and triggering appropriate actions.
- Feedback output to the user.

 System Requirements:

- Hardware: Webcam, microphone, device specifications (8GB RAM, Intel i5 processor).
- Software: Python, libraries (OpenCV, Mediapipe, SpeechRecognition, etc.), supported OS.

 Testing and Evaluation:

- Gesture recognition accuracy tests.
- Voice command recognition accuracy tests.

 Future Enhancements:

- Integration of object detection for real-time recognition.
- Expansion of system support for mobile and wearable platfor