A Project report on

VOLUMEWIZARD: GESTURE-DRIVEN SOUND MASTERY

A Dissertation submitted to in partial fulfilment of the academic requirements for the award of the Degree

Bachelor of Technology
In
INFORMATION TECHNOLOGY

Submitted by

KASHISH SINGHAL 21H51A1247
NADEM AKSHAYA 21H51A1264

Under the esteemed guidance of
Mr. K. Venkateswara Rao
Associate Professor & HOD
Department of IT

Department of Information Technology

CMR COLLEGE OF ENGINEERING & TECHNOLOGY
(UGC AUTONOMOUS)
(NAAC Accredited with 'A+' Grade & NBA Accredited)
(Approved by AICTE, Permanently Affiliated to JNTU Hyderabad)
KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

DECEMBER 2023

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

(UGC AUTONOMOUS)

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the Mini Project-I report entitled **"VOLUMEWIZARD: GESTURE-DRIVEN SOUND MASTERY"** being submitted by KASHISH SINGHAL (21H51A1247), NADEM AKSHAYA (21H51A1264), in partial fulfilment for the award of **Bachelor of Technology in Information Technology** is a record of bonafide work carried out his/her under my guidance and super vision. The results embodies in this project report have not been submitted to any other University or Institute for the award of any Degree.

GUIDE

Mr. K. VENKATESWARA RAO
Associate Professor & HOD
Department of IT

HOD

Mr. K. VENKATESWARA RAO
Associate Professor & HOD
Department of IT

Acknowledgment

With great pleasure we want to take this opportunity to express my heartfelt gratitude to all the people who helped in making this project work a grand success.

We are grateful to **Mr. K. Venkateswara Rao, Associate Professor, HOD**, Department of Information Technology for his valuable technical suggestions and guidance during the execution of this project work.

We would like to thank **Mr. K. Venkateswara Rao,** Head of the Department of Information Technology, CMR College of Engineering and Technology, who is the major driving forces to complete my project work successfully.

We are very grateful to **Dr. Ghanta Deva Dasu**, Dean-Academics, CMR College of Engineering and Technology, for his constant support and motivation in carrying out the project work successfully.

We are highly indebted to **Major Dr. V A Narayana,** Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the **Teaching & Non- teaching** staff of Department of Information Technology for their co-operation

We express our sincere thanks to **Shri. Ch. Gopal Reddy**, Secretary, CMR Group of Institutions, for his continuous care.

Finally, we extend thanks to our parents who stood behind us at different stages of this Project. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project work.

KASHISH SINGHAL 21H51A1247

NADEM AKSHAYA 21H51A1264

DECLARATION

We hereby declare that results embodied in this Report of Projection **"VOLUMEWIZARD: GESTURE-DRIVEN SOUND MASTERY"** are from the work carried out by using partial fulfilment of the requirements for the award of B. Tech degree. We have not submitted this report to any other university/institute for the award of any other degree.

SIGNATURE

KASHISH SINGHAL 21H51A1247

NADEM AKSHAYA 21H51A1264

ABSTRACT

In this project we are developing a volume controller and brightness adjustment in which we are using hand gestures as the input to control the system. Essentially, the OpenCV module utilized to manage the gestures in this implementation. This system primarily employs a web camera to record or capture images and videos, and this application regulates the system's volume/brightness based on the input. The primary purpose of the system is to change its volume and brightness. Python and OpenCV are both used to implement the project. To operate a computer's fundamental functions, such as volume control and brightness adjustment , we can use hand gestures. People won't have to acquire the typically burdensome machinelike abilities as a result. These hand gesture systems offer a modern, inventive, and natural means of nonverbal communication. These systems have many different applications in human interaction. This project's goal is to discuss a volume control/brightness control system based on hand gesture detection and hand gesture recognition. A high-resolution camera is used in this system to recognize the user's gestures as input. The primary objective of hand gesture recognition is to develop a system that can recognize human hand gestures and use that information to control a device. With real-time gesture recognition, a specific user can control a computer by making hand gestures in front of a system camera that is connected to a computer. With the aid of OpenCV and Python, we are creating a hand gesture volume control system in this project. This system allows for control without using a keyboard or mouse, controlled by hand gesture.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In recent years, human-computer interaction has evolved to include innovative methods that go beyond traditional input devices. One such exciting development is the use of hand gestures to control electronic devices, offering a more intuitive and natural user experience. This project focuses on implementing a volume control system using hand gestures with the help of OpenCV (Opensource Computer Vision) and Python. OpenCV is a powerful computer vision library that provides tools for image and video processing, making it an ideal choice for gesture recognition applications. By leveraging OpenCV's capabilities, we can track and interpret hand movements in real-time, allowing users to control the volume of a device with simple gestures. In this project, we aim to elevate the hand gesture-based volume control system to include additional functionalities such as mute, unmute, and voice commands. By expanding the capabilities of our system, we can create a more comprehensive and user-friendly audio control experience.

Building upon the foundation of OpenCV and Python, we will integrate voice recognition technology to allow users to control the volume through spoken commands. This inclusion of natural language processing adds another layer of interactivity to the system, making it versatile and accessible. The mute and unmute functionalities will be implemented to provide users with a convenient way to toggle the audio output on or off using specific hand gestures. This feature enhances the overall user experience by offering quick and intuitive control over the device's sound settings. Additionally, we will incorporate voice commands for increasing and decreasing the volume. Users can simply speak commands like "increase volume" or "decrease volume" to adjust the audio levels without the need for physical interaction. This voice-controlled aspect adds a futuristic and convenient dimension to the system, catering to users who prefer a hands-free approach. Throughout this project, we will explore the integration of audio processing libraries to handle voice commands. We'll also delve into the synergy between hand gesture recognition and voice recognition, ensuring a seamless and cohesive user interface. The combined functionality of hand gestures and voice commands creates a versatile and adaptable audio control system suitable for various applications, from smart homes to interactive multimedia environments.

1.2 OBJECTIVE

This project aims to revolutionize audio control by implementing a gesture and voice-based system using OpenCV and Python. The primary objective is to create a seamless, hands-free experience for users, allowing them to intuitively adjust volume settings through hand gestures captured in real-time. In addition to conventional volume control, the system incorporates mute and unmute functionalities, enhancing user convenience. The integration of voice recognition technology introduces spoken commands for volume adjustments, further expanding the system's capabilities. By combining computer vision, audio processing, and natural language processing, the project seeks to provide a cohesive and user-friendly interface. Ultimately, this project aspires to improve the overall user experience in diverse environments, showcasing the versatility and adaptability of gesture and voice-controlled audio systems for applications ranging from smart homes to interactive multimedia setups.

CHAPTER 2

SYSTEM ANALYSIS

2.1 FUNCTIONAL REQUIREMENTS:

Certainly, here's a condensed version of the functional requirements:

1. Operations on Every Screen:

1.1 Gesture Recognition Screen:

- Display live webcam feed.
- Detect and track hand gestures in a specified region.
- Provide visual feedback for recognized gestures.

1.2 Volume Control Screen:

- Allow volume adjustment through hand gestures.
- Display current volume level.
- Include gestures for muting and unmuting.

1.3 Brightness Adjustment Screen:

- Enable brightness control via hand gestures.
- Display current brightness level.
- Implement gestures for increasing and decreasing brightness.

2. Data Handling Logic:

2.1 Gesture Data:

- Capture, process, and store hand gesture data.
- Implement real-time and accurate recognition algorithms.
- Securely store timestamped gesture data.

2.2 System Configuration Data:

- Manage configurable settings.
- Allow customization of camera properties and thresholds.
- Ensure adaptability to varying conditions.

3. System Outputs:

3.1 Gesture Log Report:

- Generate a detailed log with timestamped gesture data.
- Include gesture type and associated actions.

3.2 System Status Indicator:

- Provide real-time visual feedback on system status and recognition accuracy.
- Include user-friendly error notifications.

4. Workflows:

4.1 Gesture Recognition Workflow:

- Outline the process from capturing to recognizing hand gestures.
- Define how recognized gestures translate to system actions.

4.2 User Interaction Workflow:

- Detail user interactions, from initiation to executing control gestures.
- Provide clear feedback and guidance.

5. User Permissions and CRUD Operations:

5.1 User Roles:

- Define roles (admin, regular user) with specific permissions.
- Specify who can modify system configurations.

5.2 System Configuration Modifications:

- Authorize specific user roles for system setting modifications.
- Implement secure authentication for access.

6. Regulatory and Compliance:

6.1 Privacy and Data Protection:

- Adhere to data protection regulations.
- Clearly define handling and storage of user data.

6.2 Accessibility:

- Comply with accessibility standards.
- Implement features for diverse user abilities.

6.3 Compliance Reporting:

- Ensure availability of necessary certifications and compliance reports.

2.2 SOFTWARE REQUIREMENTS:

LIBRARY	VERSION	INSTALLATION COMMAND
Python	3.11.2	Download Python
OpenCV(cv2)	4.8.1.78	pip install opencv-python
MediaPipe	0.10.8	pip install mediapipe
ctypes	1.2.0	pip install comtypes
NumPy	1.26.2	pip install numpy
SpeechRecognition	3.10.1	pip install SpeechRecognition
screen_brightness_control	0.22.1	pip install screen-brightness-control
Pycaw	20230407	pip install pycaw

2.3 HARDWARE REQUIREMENTS:

HARDWARE	VERSION/SIZE/CAPACITY	REQUIREMENTS
Camera	2023.2311.5.0	Webcam with good resolution and frame rate
System Microphone	6.0.9049.1	Clear audio capture capabilities
Moniter	LCD/LED	Sufficient processing power for real time operation
System Speakers	6.0.9049.1	Device for playing audio with adjusted volume

CHAPTER 3

LIBRARIES

3.1 LIBRARIES

1. cv2 (OpenCV):

OpenCV, or Open-Source Computer Vision, is a comprehensive open-source library that has become a cornerstone in the field of computer vision and image processing. Developed to provide a wide range of tools and algorithms, OpenCV is instrumental in various applications, from simple image manipulation to complex computer vision tasks. In the script at hand, the cv2 module is employed for interfacing with the webcam, capturing video frames, colour space conversions, and image display. The library's versatility allows developers to seamlessly integrate it into projects, facilitating tasks such as hand landmark detection and visualization. OpenCV's continuous development and extensive documentation make it a go-to choose for developers working on projects that involve image and video processing.

2. media pipe:

Media pipe, a machine learning framework developed by Google, addresses the challenges of developing perception-based applications by providing a suite of pre-built solutions. The mp. solutions.hands module in Mediapipe is specifically utilized in this script for hand tracking. By abstracting the complexities of hand landmark detection and tracking, Mediapipe simplifies the implementation of applications requiring accurate hand gesture recognition. Its high-level interface allows developers to focus on the application's logic rather than delving into intricate details of hand tracking algorithms. With the backing of Google's expertise and resources.

MediaPipe is an open-source framework developed by Google that facilitates the development of perception-based applications. It is designed to handle various tasks related to computer vision and machine learning, making it a versatile tool for developers.MediaPipe adopts a modular architecture, allowing developers to choose and integrate specific components for their applications. These components, known as "solutions," are pre-built and cover a wide range of tasks, including face detection, hand tracking, pose estimation, and more.MediaPipe supports cross-platform development, making it suitable for applications on various devices, including smartphones, tablets, laptops, and desktops. This flexibility enhances the framework's applicability in diverse scenarios.The framework employs efficient neural network models that strike a balance between accuracy and computational efficiency. This is crucial for real-time applications, where responsiveness is a key factor.

MediaPipe provides extensive documentation, tutorials, and examples to facilitate a smooth learning curve for developers. This accessibility encourages a broader community to engage with the framework and

contribute to its development.Developers can customize and extend MediaPipe solutions to meet specific requirements. This adaptability is particularly valuable when integrating the framework into diverse projects with unique demands.The `mp.solutions.hands` module within MediaPipe is specifically dedicated to hand tracking. It utilizes advanced algorithms for hand landmark detection and tracking, simplifying the implementation of applications that rely on accurate hand gesture recognition.

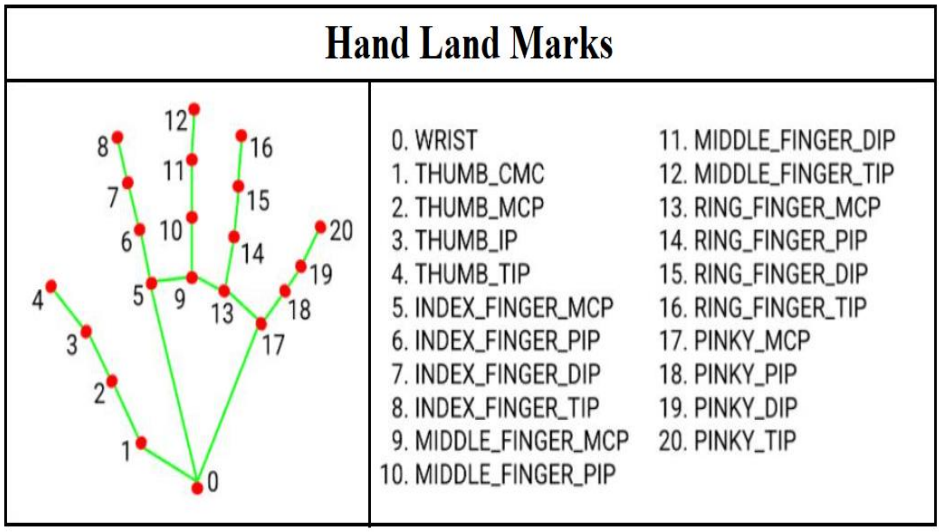

Fig 1. Hand Landmarks

3. math:

Python's standard math module serves as a robust library for mathematical operations, offering a broad range of functions. In this script, the hypot() function from the math module is employed to calculate the Euclidean distance between two points. This calculation is fundamental in hand gesture analysis, as it provides a metric for the distance between detected hand landmarks. The math module's inclusion in the Python standard library ensures its ubiquity and reliability, making it an ideal choice for mathematical computations. From basic arithmetic operations to complex mathematical functions, the math module forms an integral part of Python's computational capabilities, contributing to the accuracy and efficiency of mathematical operations in diverse applications.

4.ctypes, POINTER (ctypes):

The ctypes module in Python acts as a bridge between Python and dynamic link libraries (DLLs), allowing for the invocation of functions from shared libraries. When combined with

POINTER, it becomes a powerful tool for handling pointers and interfacing with functions in dynamic link libraries. In this script, ctypes is harnessed to interact with the Windows API, specifically for controlling audio settings. The integration of ctypes and POINTER enables seamless communication with the operating system, facilitating dynamic adjustments to audio-related parameters. This direct interaction with the Windows API showcases the flexibility and extensibility that Python provides, making it possible to integrate platform-specific functionalities effortlessly.

5. AudioUtilities, IAudioEndpointVolume (pycaw):

Pycaw, a Python library, offers a convenient interface to the Windows Core Audio API. This API allows developers to interact with audio devices, enabling functionalities such as adjusting volume levels and muting audio. In the presented script, Pycaw is utilized to control audio volume and mute settings in real-time. Leveraging functions like SetMasterVolumeLevel() and SetMute(), the script dynamically adjusts the system's audio configuration based on hand gestures. Pycaw simplifies the complexities of interacting with the Windows Core Audio API, providing a higher-level abstraction for audio-related operations. The integration of Pycaw showcases the capabilities of Python in seamlessly interfacing with underlying system functionalities.

6.numpy:

NumPy, an abbreviation for z script, NumPy's np.interp() function plays a pivotal role in mapping hand gestures to specific volume levels. This interpolation mechanism establishes a smooth relationship between detected hand positions and the desired audio volume, enhancing the precision and responsiveness of the system.

NumPys Proficiencies array operations significantly contributes to the performance of the code, particularly in tasks involving complex numerical computations. Its broad range of functionalities and optimizations makes NumPy a cornerstone for numerical computing in Python.

7.speech_recognition (SpeechRecognition):

The SpeechRecognition library serves as a user-friendly interface to various speech recognition engines, providing a convenient tool for incorporating voice recognition capabilities into Python scripts. In the script under consideration, a Recognizer object from SpeechRecognition is utilized to capture and interpret spoken commands. By leveraging Google's speech recognition service, the library enables the

script to comprehend and respond to voice commands effectively. The SpeechRecognition library abstracts the intricate details of speech recognition engines, offering a straightforward and accessible interface for developers to integrate voice-based functionalities into their applications. This ease of use, coupled with support for multiple recognition engines, makes SpeechRecognition a valuable asset for projects involving voice interaction.

8.os:

The os module, a core component of Python, provides a wealth of functionalities for interacting with the operating system. In the script, os is employed for the graceful termination of the application upon receiving an exit command. The line os.kill(os.getpid(), signal.SIGINT) sends a SIGINT signal, initiating a controlled shutdown of the program. The os module enhances the script's robustness by allowing proper cleanup before program termination, contributing to the overall stability of the application. Whether it's managing files, directories, or processes, the os module serves as a versatile tool for developers working on diverse applications across different operating systems.

9.signal:

The signal module in Python provides mechanisms for handling signals, which are notifications sent process to notify it of specific events. In the script, signal.SIGINT represents command.

The interrupt signal, and it is employed to gracefully terminate the application in response to an exit com. The module enhances the robustness of the script by allowing proper cleanup and resource release before program termination. Effectively managing signals is crucial for the overall stability and reliability of the application, especially in scenarios where user-initiated interruptions need to be handled gracefully

10.screen_brightness_control:

The screen_brightness_control library stands as a cross-platform solution for controlling screen brightness. It offers a unified interface for adjusting brightness levels on different operating systems. In the script, the library is leveraged to retrieve the current screen brightness (get_brightness()) and dynamically set it based on hand gestures (set_brightness()). This integration enhances the user experience by allowing intuitive control over screen illumination. The library abstracts the platform-specific details of brightness control, providing a consistent interface for developers to incorporate brightness adjustments seamlessly into their applications. The ability to interact with screen brightness in a uniform manner across different platforms adds a layer of convenience for developers aiming to create applications with cross-platform support

CHAPTER 4

SYSTEM DESIGN

4.1 PROPOSED SOLUTION:

The methodology described outlines the system flow for a Python program that uses hand gestures or the range between fingers to control and adjust the volume range of a connected computer system or operating device. The program begins by taking input using voice commands like volume or brightness from user. After taking input, based on user requirments volume or brightness the program runs. For example, if user says volume, then a camera dialog box opens to capture images of the user's gestures. It utilizes the Media pipe library to detect and recognize the user's hands and their landmarks. The program analyses the pre-processed data using Python libraries to recognize specific hand gestures or finger ranges made by the user. For example, it can identify the gap between the index finger and thumb to indicate an increase or decrease in volume range as well as mute/unmute. By accessing the coordinates of specific hand landmarks, such as the tip of the thumb and index finger, the program calculates the distance between them. The program provides visual feedback to the user by displaying a volume bar that reflects the current volume range or the effect of their gestures or finger ranges. We will see a bar showing the increase and decrease in volume as soon as we increase or decrease the gap between our finger and thumb increases or decreases. The program continues to capture input and adjust the volume range based on the user's gestures or finger ranges until the user stops or until the program is terminated. If user wants brightness control, then, a camera dialog box opens to capture the user’s hands and their landmarks. The program analyses the data using python libraires to recognize specific hand landmarks. Here for brightness control the hand gesture is distance between Pinky finger and thumb finger. The program continues to capture input and adjust the brightness range based on the user's gestures or finger ranges until the user stops or until the program is terminated.

Formula to calculate distance for volume adjustment:

The formula used to calculate the volume in our code is based on the distance between two hand landmarks. Specifically, the distance between the tips of the thumb and index finger is used to determine the volume level. Here’s the relevant part of the code:

length=hypot (x2-x1, y2-y1): This calculates the Euclidean distance between two points (x1, y1) and (x2, y2). In this case, it’s the distance between the tips of the thumb and index finger.

vol=np. interp (length [30,350], [vol_min, vol_max]): This function performs linear interpolation. It maps the calculated distance(length) from the hand landmarks to a volume level within the

specified range. The range [30,350] corresponds to the expected range of distance between the thumb and index finger, and [vol_min, vol_max] is the range of possible volume levels. So, the volume (vol) is adjusted proportionally based on the distance between the thumb and index finger, ensuring that shorter distances result in lower volume levels, and longer distances result in higher volume levels. The vol_min

and vol_max values represent the minimum and maximum volume levels supported by the audio system, respectively.

These calculations help ensure precise volume control based on the user's gestures or finger ranges. Fig. 2 Volume Result at 0 percentage. Fig. 3 Volume Result at 99 percentage. Fig. 4 Volume Result at mute. This Fig. 2 The result shows a video playing in the background with no volume, and the screen bar indicates a volume of 0%. This occurs when the distance between the thumb tip (point 4) and the fingertip (point 8) is 0. Fig. 3 The result presents a video playing in the background with maximum volume, and the screen bar indicates a volume of 99%. This occurs when the distance between the thumb tip (point 4) and the fingertip (point 8) is at its maximum. Fig. 4 The result displays a video playing in the background with mute, and the screen bar shows a volume of 100%. This occurs when there is a significant distance between the thumb tip (point 4) and the fingertip (point 8).

Formula to calculate distance for brightness adjustment:

The formula used to calculate brightness in the provided code is based on the distance between two hand landmarks, specifically the thumb and pinky finger. The code adjusts the screen brightness based on the calculated distance between these two points. Here's the relevant part of the code:

Get the positions of the pinky and thumb

pinky_tip = (int (lm_list [20][0]), int (lm_list [20][1]))

thumb_tip = (int (lm_list [4][0]), int (lm_list [4][1]))

Calculate distance between pinky and thumb

distance = calculate_distance(pinky_tip, thumb_tip): This function calculates the Euclidean distance between the positions of the pinky and thumb tips.

Adjust brightness based on the distance

brightness_change = int((distance - 50) / 10): int((distance - 50) / 10): This part adjusts the brightness based on the calculated distance. It subtracts 50 from the distance to create a centre point and then divides by 10 to scale the change. The result is an adjustment value for brightness.

Extract the brightness value from the list

current_brightness = get_brightness(display=0)[0] : This retrieves the current brightness level from the display. The get_brightness function likely returns a list of brightness values, and [0] extracts the primary brightness value.

new_brightness = max(0, min(100, current_brightness + brightness_change)): This ensures that the new brightness level (new_brightness) stays within the valid range of 0 to 100. The max and min functions clamp the value to this range

Set the new brightness

set_brightness(new_brightness, display=0): function is used to apply the new brightness level to the display.

These calculations help ensure precise brightness control based on the user's gestures or finger ranges. Fig. 5 Brightness Result at 0 percentage. Fig. 6 Brightness Result at 100 percentage. This Fig. 5 The result shows a screen brightness with 0. This occurs when the distance between the thumb tip (point 4) and the pinky tip(point 20) is 0. Fig. 6 The result shows a screen brightness with 100. This occurs when the distance between the thumb tip (point 4) and the pinky tip (point 20) is at its maximum.

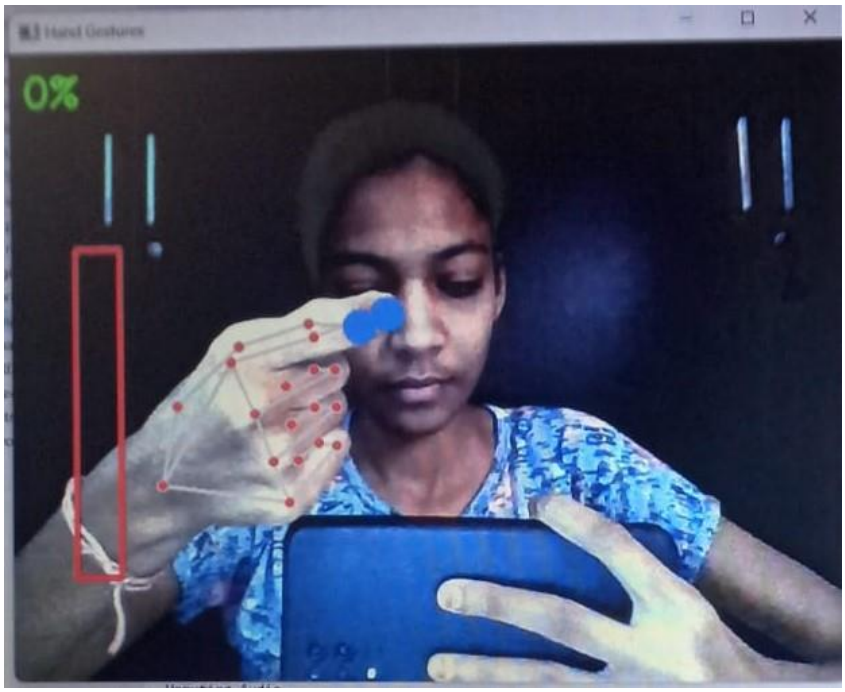

Fig 2. Volume Result at 0 percentage

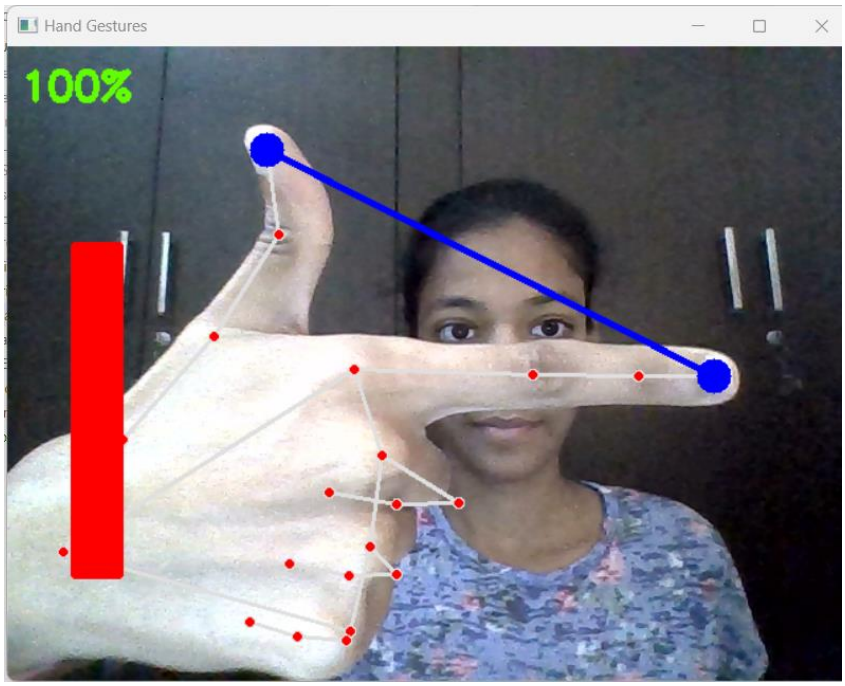

Fig 3. Volume Result at 100 percentage

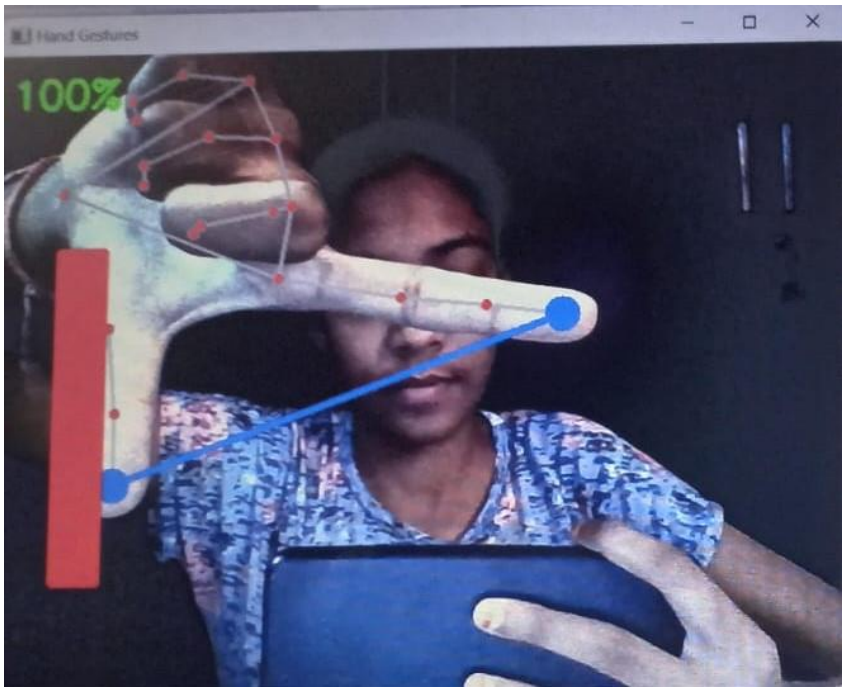

Fig 4. Volume Result at mute

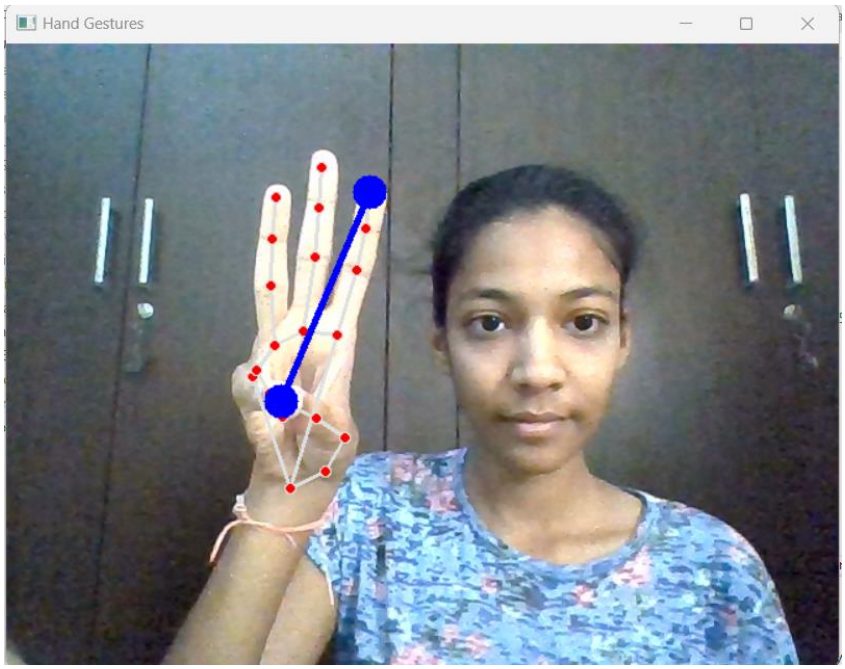

Fig 5. Brightness Result at 0 percentage

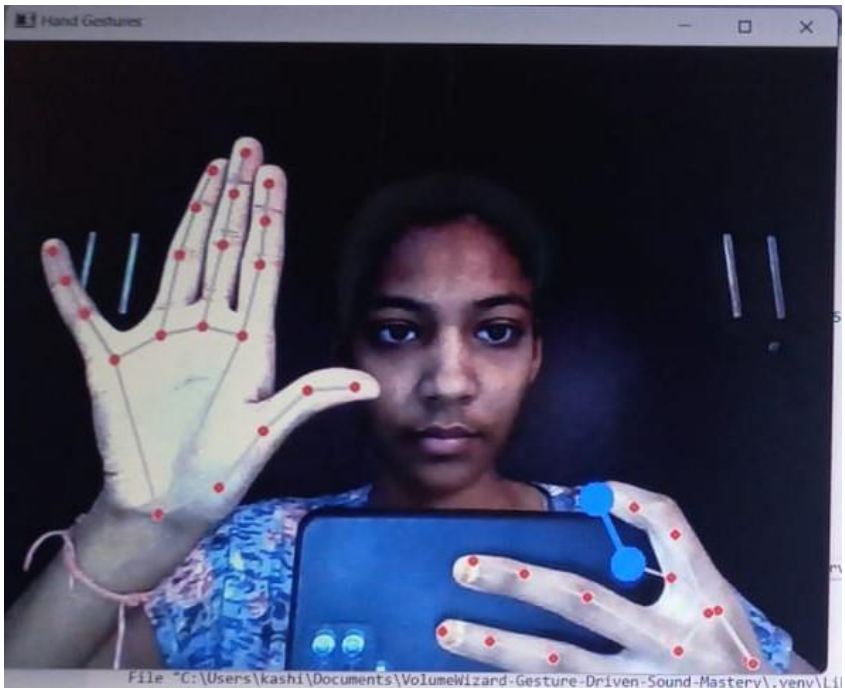

Fig 6. Brightness Result at 100 percentage

4.2 PROPOSED TECHNIQUE:

Here's a proposed technique:

1. Install Required Libraries:

Make sure you have OpenCV installed. You can install it using bash

```
pip install opencv-python
```

2. Capture Video:

Use OpenCV to capture video frames from the webcam.

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

3. Hand Detection:

Use a pre-trained hand detection model or cascade classifier to identify the region of interest (ROI) containing the hand.

4. Hand Gesture Recognition:

Once you have the ROI containing the hand, use image processing techniques or machine learning models to recognize hand gestures. You may consider using a pre-trained deep learning model or designing a custom solution based on your requirements.

5. Volume and Brightness Control:

Map specific gestures to volume control and brightness adjustment commands. For example, a swipe gesture might increase or decrease volume, while a pinch gesture could control brightness.

6. Real-Time Interaction:

Update the system's volume and brightness in real-time based on the recognized gestures.

7. Testing and Optimization:

Test the system with various hand gestures, and optimize the gesture recognition algorithm for accuracy and responsiveness.

4.3 SYSTEM ARCHITECTURE:

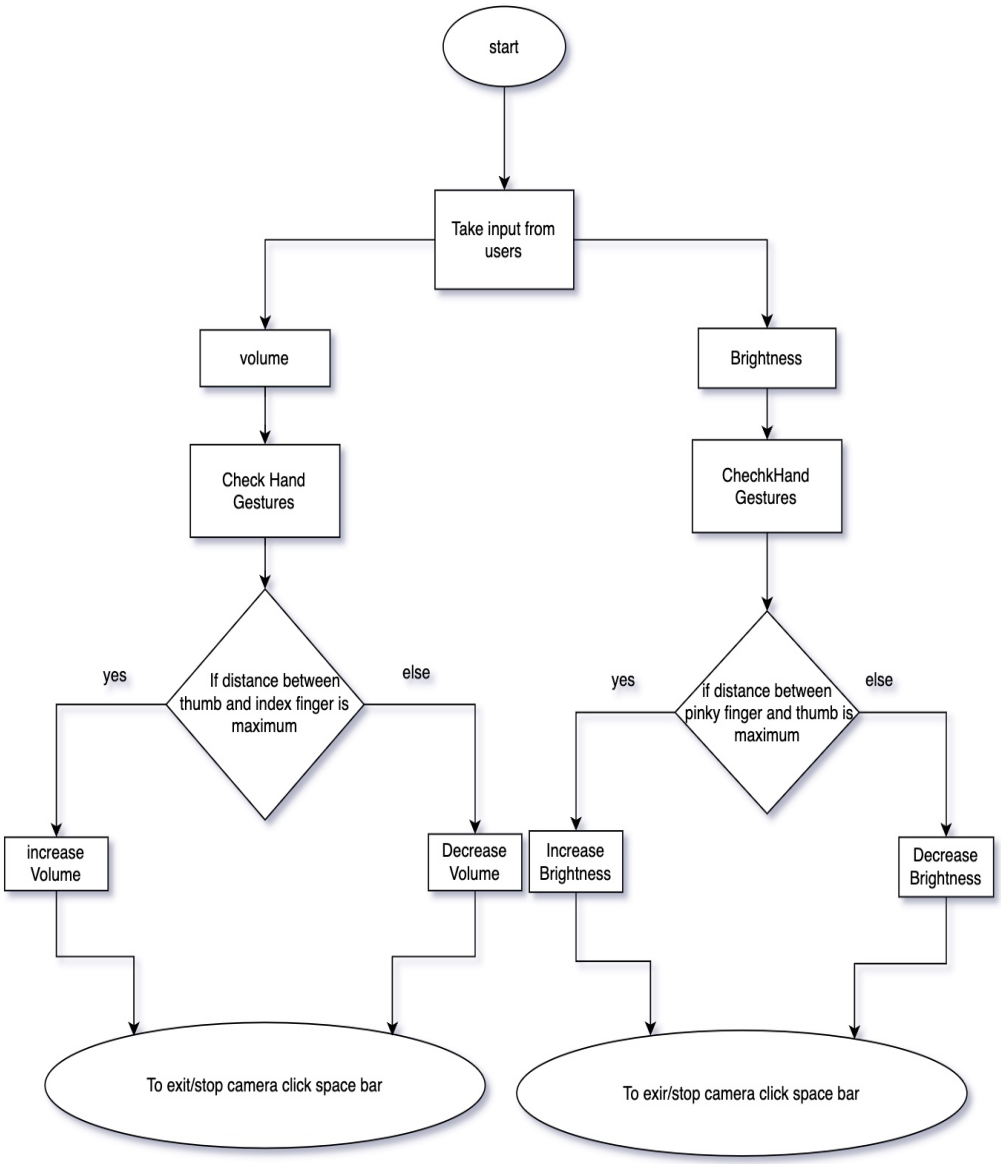

Fig 7. System Architecture

ADVANTAGES

1. User-Friendly Interaction:

- Easy Hand Movements: Navigate your computer's volume and brightness effortlessly with intuitive hand gestures, offering a user interface that feels natural.

2. Hands-Free Convenience:

- Effortless Control: Enjoy the convenience of adjusting settings without physically interacting with your computer, particularly useful during activities demanding hands-on attention.

3.Voice-Activated Commands:

- Speech Integration: Enhance control versatility by incorporating voice commands for taking input/requirements from users.

4. No Physical Contact:

- Unlike traditional methods that involve physical contact with buttons or devices, this system operates hands-free. It brings a touch of the future by allowing you to control your computer without actually touching anything.

CHAPTER 5

IMPLEMENTATION

5.1 IMPLEMENTATION:

Visual Studio, an IDE software interpreter, is used to implement it, but the command prompt is also an option. Import the open CV Library to the Python project, which is utilized as a PC vision device, and peruse the picture, which is only available in this specific circumstance. The cross-platform framework for building multimodal applied machine learning pipelines known as Media Pipe must then be utilized. It is used for the purpose of detection. The Euclidean norm is returned by the Hypot () method. Then, at that point, to get the default sound gadget utilizing PyCAW, we utilized com types, which is an essential library and sound utility. If the video camera is open, the video capture objects will capture the data. Then Media Pipe Hands is a solution for tracking hands and fingers with high fidelity. On the off chance that the hand motion is identified, we need to draw the accompanying framework of the hand utilizing the capability. Utilize PyCAW to acquire the default audio device. After that, we connected to the necessary volume, determined the range, which is from 0 to 100, read the webcam frames, and converted the image to RGB. After the hands are detected, we will locate the key points, using cv2.circle to highlight the dots in the key points and mpDraw.draw_landmarks to connect the key points. After that, we print with the index and thumb fingertips (x1, y1, x2, y2). After that, we print the fingers after determining which ones are up. If the index and thumb fingers are close together, we lower the volume, whereas if they are far apart, we raise it. The length of the line that runs through the coordinates is then found. Draw a map using the volume range of the thumb and index fingers. Similarly, brightness control will work but with different hand gesture i.e., if thumb and pinky fingers tip are close together decrease the brightness, whereas if they are far increase the brightness.

5.2 SOURCE CODE:

```
import cv2
import mediapipe as mp
from math import hypot
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
import numpy as np
import speech_recognition as sr
from screen_brightness_control import get_brightness, set_brightness
import os
import signal

def calculate_distance(point1, point2):
    return hypot(point2[0] - point1[0], point2[1] - point1[1])

def recognize_speech():
    recognizer = sr.Recognizer()

    with sr.Microphone() as source:
        print("Say something:")
        recognizer.adjust_for_ambient_noise(source)

        try:
            audio = recognizer.listen(source, timeout=5, phrase_time_limit=5)
        except sr.WaitTimeoutError:
            print("Timeout: No speech detected.")
            return None

    try:
        return recognizer.recognize_google(audio).lower()
    except sr.UnknownValueError:
        print("Sorry, I did not hear your command.")
        return None
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech Recognition service; {e}")
        return None
```

```
def control_audio(mode):

cap = cv2.VideoCapture(0)
    mp_hands = mp.solutions.hands
    hands = mp_hands.Hands()
    mp_draw = mp.solutions.drawing_utils
    devices = AudioUtilities.GetSpeakers()
    interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
    volume = cast(interface, POINTER(IAudioEndpointVolume))
    vol_bar = 400
    vol_min, vol_max = volume.GetVolumeRange()[:2]

    while True:
        success, img = cap.read()
        if not success or img is None:
            print("Error: Empty frame.")
            break

        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        results = hands.process(img_rgb)
        lm_list = []

        if results.multi_hand_landmarks:
            for hand_landmark in results.multi_hand_landmarks:
                lm_list.extend([(id, int(lm.x * img.shape[1]), int(lm.y * img.shape[0]))
                                for id, lm in enumerate(hand_landmark.landmark)])
                mp_draw.draw_landmarks(img, hand_landmark, mp_hands.HAND_CONNECTIONS)

if lm_list:
            x1, y1 = lm_list[4][1:3]
            x2, y2 = lm_list[8][1:3]

            if mode == 'volume':
                length = hypot(x2 - x1, y2 - y1)
                vol = np.interp(length, [30, 350], [vol_min, vol_max])
                vol_bar = np.interp(length, [30, 350], [400, 150])

                print(vol, int(length))
                volume.SetMasterVolumeLevel(vol, None)

                thumb_status = detect_thumb_status(lm_list)
```

```
            if thumb_status == "thumb_up":
                print("Unmuting Audio.")
                volume.SetMute(False, None)
            elif thumb_status == "thumb_down":
                print("Muting audio.")
                volume.SetMute(True, None)

        elif mode == 'brightness':
            # Similar brightness control logic as before

    cv2.imshow('Hand Gestures', img)

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

cap.release()
cv2.destroyAllWindows()

def detect_thumb_status(hand_landmarks):
    # Function implementation here

print("Listening for voice command...")
while True:
    command = recognize_speech()

    if command:
        if "volume" in command:
            print("Controlling Volume...")
            control_audio('volume')
        elif "brightness" in command:
            print("Adjusting Brightness...")
            control_audio('brightness')
        elif "exit" in command:
            print("Exiting...")
            os.kill(os.getpid(), signal.SIGINT)
            break
        else:
            print("Command not recognized. Try saying 'volume', 'brightness', or 'exit.'")
```

CHAPTER 6

OUTPUT

OUTPUT

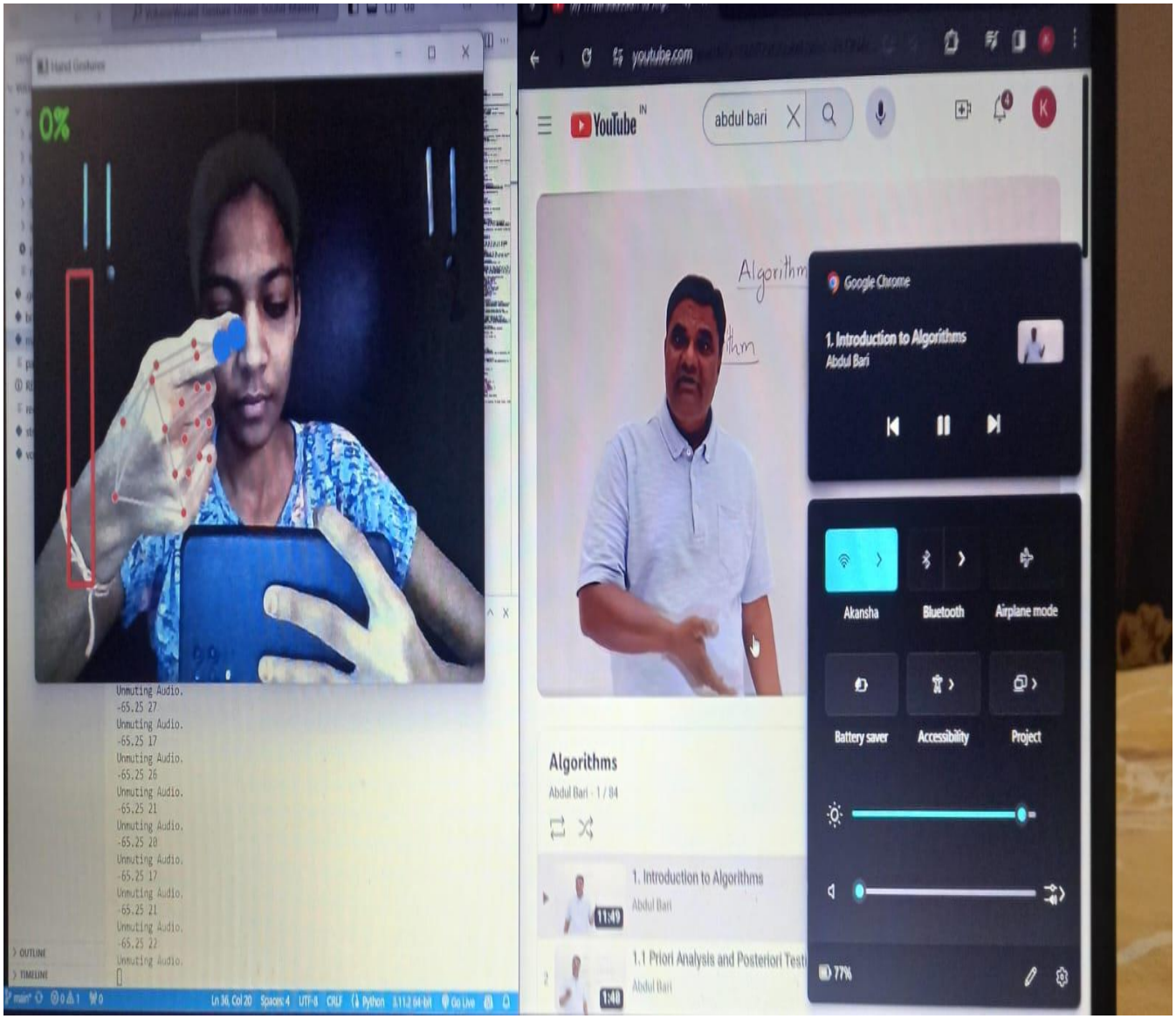

Fig 8. Minimum Level of Volume

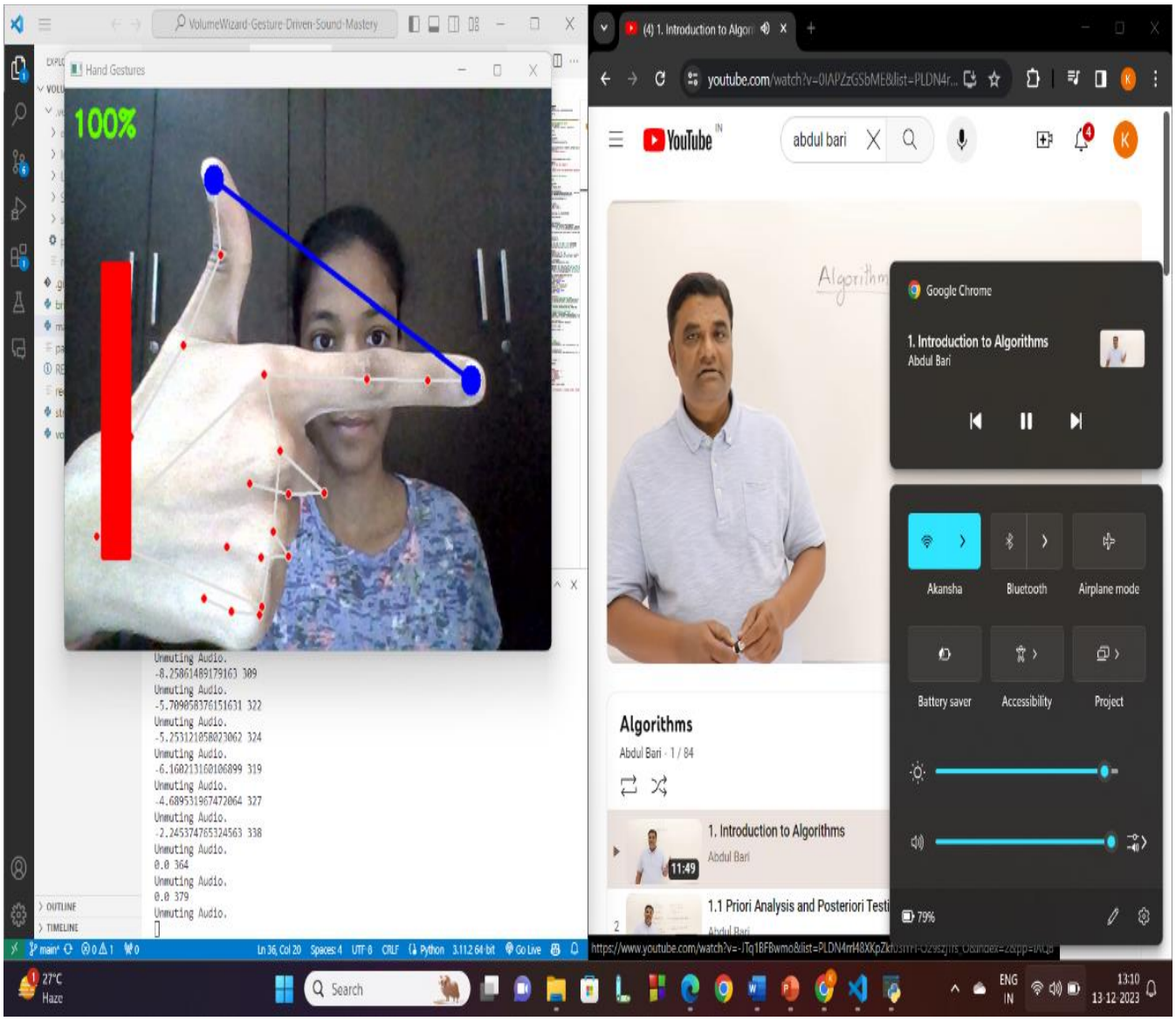

Fig 9. Maximum Level of Volume

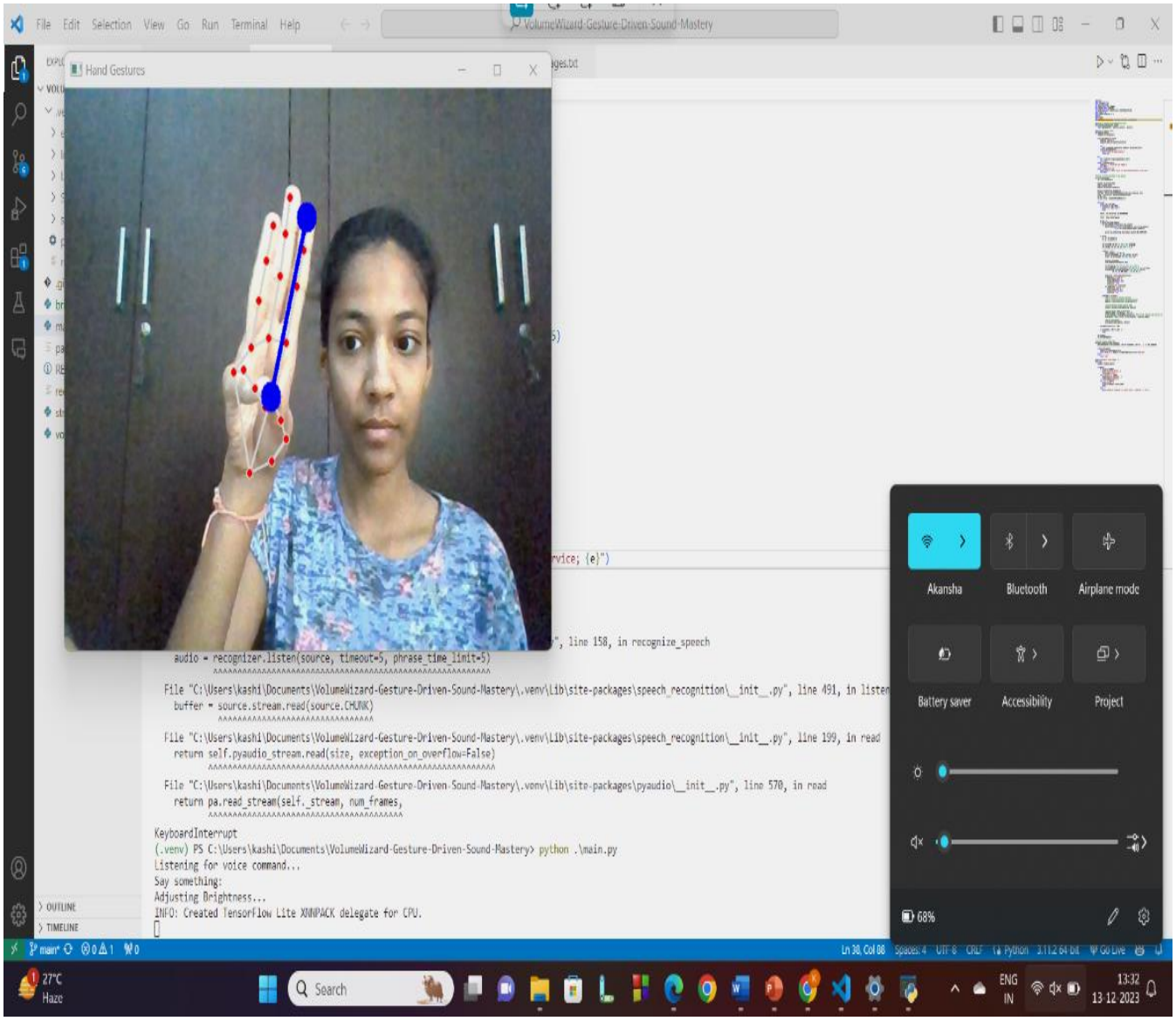

Fig 10. Minimum Level of Brightness

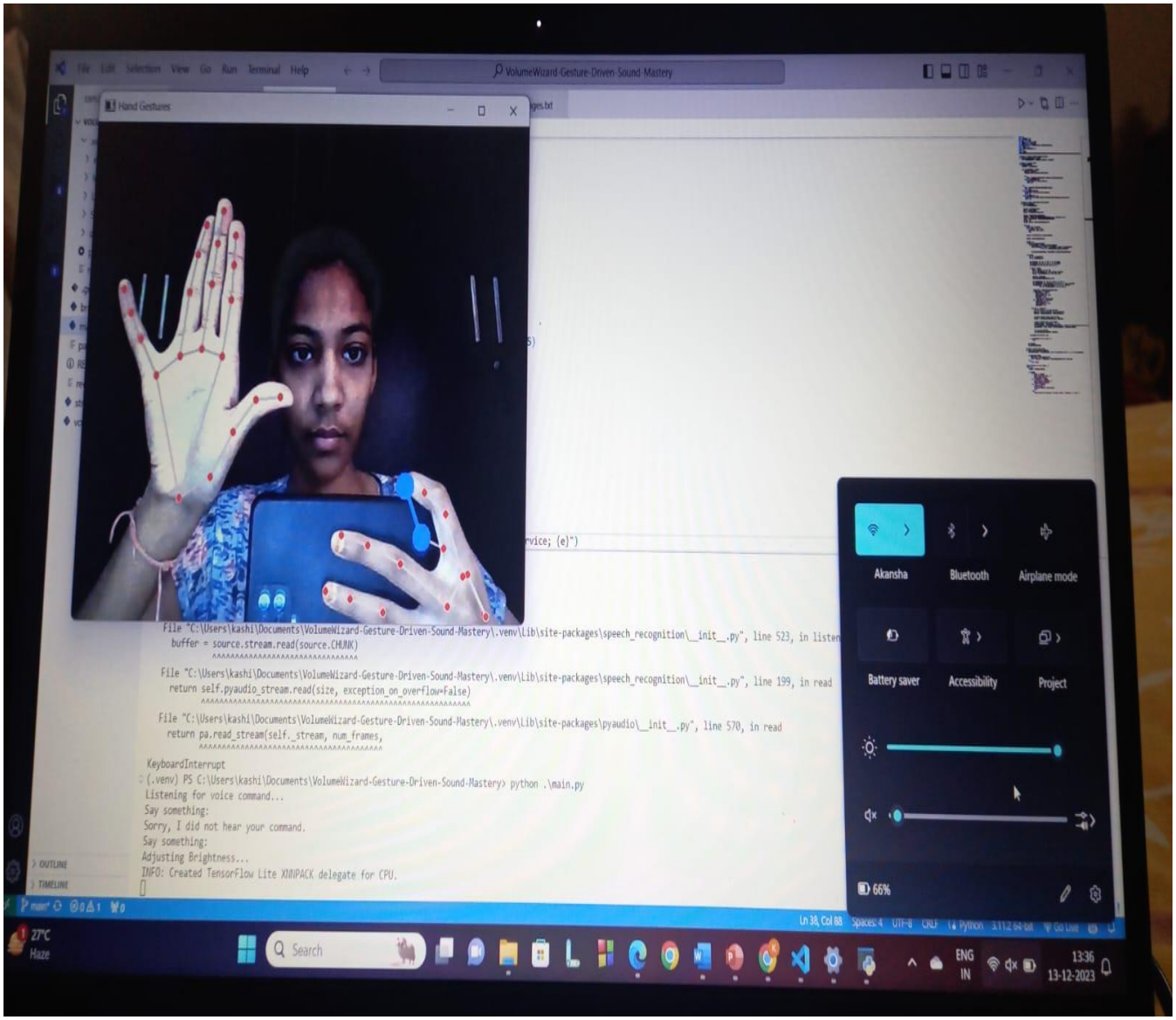

Fig 11. Maximum Level of Brightness

CHAPTER 7

CONCLUSION

CONCLUSION

This project's current application supports hand gestures as a practical and easy method of software control. A gesture-based volume controller does not require any special markers and may be used in real life on simple PCs equipped with low-cost cameras. Because it does not require a high-definition camera to recognise or capture hand gestures with exceptional clarity. This would herald in a new era of computer-human connection in which there would be no need for direct physical touch. Using gesture commands, the technology makes it simple for anybody to use a computer. It makes use of a number of algorithms and approaches, such as tracing prominent regions in pictures and estimating distance between sites. The system can track each hand's location. It makes use of a number of algorithms and approaches, such as tracing prominent regions in pictures and estimating distance between sites. The gadget can track the location of each hand's index finger and, in particular, counter tops. It is a quick and simple way to manage sound equipment that requires little physical labour. It can operate in real-time on a standard PC with low-cost cameras

CHAPTER 8

FUTURE ENHANCEMENT

FUTURE ENHANCEMENT

1. Improving Accuracy and Robustness
2. Recognizing Complex Gestures and Movement
3. Audio Control Enhancement for Bass and Treble

CHAPTER 9

REFERENCES

REFERENCES

[1] Tiwari, S., Yadav, A. L., Mishra, A., & Kukreja, D. Volume Controller using Hand Gestures.

[2] Aditya Sharma , Akshat Sethiya , Akshit Ramteke , Atharva Shinde and Prof. Shivshankar Rajput. Volume Control Using Hand Gesture

[3] Nidhishree Arun, Namratha V , Ananya Dutta , Shreenivas B. Hand Gesture Recognition and Volume Control.

[4] Aaditya Meshram, Chaitanya Thakre, Chandrabhan Rahangdale, Sagar Chouksey, Swaroop Bhoyar, Prof. Pragati Patil Volume, Brightness and Curser Controller with Hand Gesture (5), 2023

[5] Anukriti Rajput, Uttkarsh Sharma, Ashok Kumar Volume Controlled by Hand Gesture Recognition. 12(Special Issue 5),2023

[6] Dr. K. Venkatasalam, Sourav, Veluru Yashwanth Venkata Sai Chowdary, Ravella Chandu AUDIO PLAYER CONTROLLING BASED ON HAND GESTURE TECHNIQUE (6),2022

[7] Mr.V.Balu, Elamarthy Venkata Vyshnavi, Dondapati Devi Priyanka VOLUME CONTROL USING HAND GESTURES (6),2022

[8] Prof.P Ajitha, Irfan Ahmed M N, Mohith Seshan K M, Siva C Gesture Volume Control (11),2022