# VOLUME AND BRIGHTNESS CONTROL USING HAND GESTURES

**A MINI PROJECT REPORT**

***Submitted by***


**BHUVANIKA S (9517202109011)**
**RAJAKUMARI S (9517202109042)**
**SUJI S (9517202109051)**


***in partial fulfillment for the award of the degree of***

**BACHELOR OF TECHNOLOGY**

**IN**

**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**


**MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI**


**ANNA UNIVERSITY : CHENNAI 600 025**

**OCTOBER 2024**

# MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

## AUTONOMOUS

### DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

## BONAFIDE CERTIFICATE

Certified that this project report **"VOLUME AND BRIGHTNESS CONTROL HAND GESTURES "** is the bonafide work of **"BHUVANIKA S (9517202109011), RAJAKUMARI S(9517202109042),SUJI S(9517202109051)"** who carried out the mini project in 19AD784 - Image Analysis and Computer Vision Laboratory during the seventh semester July 2024 –October 2024 under my supervision.

**SIGNATURE**

**Dr.E.Emerson Nithiyaraj,**
**M.E.,Ph.D Assistant Professor,**
Artificial Intelligence and Data Science,
Mepco Schlenk Engineering College,
Sivakasi – 626 005.
Virudhunagar District.

**SIGNATURE**

**Dr.J.Angela Jennifa Sujana,**
**M.TECH.,Ph.D Professor & Head,**
Artificial Intelligence and Data Science,
Mepco Schlenk Engineering College,
Sivakasi – 626 005.
Virudhunagar District.

# TABLE OF CONTENTS

# ABSTRACT

The Hand Gesture-Based Control System is an innovative approach to enhancing human-computer interaction by leveraging real-time hand gesture recognition. This system uses a webcam to capture live video of the user's hands, with OpenCV and Media Pipe processing the frames to detect hand landmarks and recognize specific gestures. The recognized gestures are mapped to control system functions such as adjusting screen brightness and controlling the system volume. The project integrates various tools including OpenCV for image processing, Media Pipe for gesture recognition, PyAuto GUI for system control, and a brightness control module for screen adjustments.

By providing a touchless interface, this project not only improves user accessibility but also presents a more hygienic and intuitive way of interacting with computers. Its design eliminates the need for physical contact with input devices, which makes it particularly useful in public or medical settings where minimizing touch is critical for hygiene. Additionally, the system offers flexibility by supporting multiple hand gestures, allowing users to interact naturally without the need for external devices or complex configurations.

Potential applications range from assistive technologies for individuals with physical impairments to smart home controls, gaming, and immersive experiences. Its low-cost hardware requirements, combined with powerful software capabilities, make it a versatile and impactful solution in the field of gesture-based interaction systems. This system paves the way for future advancements in hands-free technology, offering broader accessibility and ease of use in everyday computing tasks.

# ACKNOWLEDGEMENT

First and foremost we **praise and thank "The Almighty",** the lord of all creations, who by his abundant grace has sustained us and helped us to work on this project successfully.

We really find unique pleasure and immense gratitude in thanking our respected management members**,** who is the backbone of our college.

A deep bouquet of thanks to respected Principal **Dr.S.Arivazhagan M.E.,Ph.D.,** for having provided the facilities required for our mini project.

We sincerely thank our Head of the Department **Dr. J. Angela Jennifa Sujana M.TECH., Ph.D.,** Professor & Head, Department of Artificial Intelligence and Data Science, for her guidance and support throughout the mini project.

We extremely thank our project coordinators **Dr.E.Emerson Nithiyaraj M.E., Ph.D,** Assistant Professor and **Mrs.A.Antony Shyamalin M.E., M.S.,** Assistant Professor (Sl.G)**,** Department of Artificial Intelligence and Data Science, who inspired us and supported us throughout the mini project.

We extend our heartfelt thanks and profound gratitude to all the faculty members of Artificial Intelligence and Data Science department for their kind help during our mini project work.

We also thank our parents and our friends who had been providing us with constant support during the course of the mini project work.

# CHAPTER 1

# INTRODUCTION

## 1.1 OVERVIEW OF PROJECT

The Hand Gesture-Based Control System is a contactless, intuitive interface designed to manage system functions like brightness and volume through hand gestures. Using technologies such as Media Pipe for hand detection, OpenCV for video capture, PyAuto GUI for system interaction, and Screen Brightness Control, the system interprets gestures in real-time. For the left hand, the distance between the thumb and index finger adjusts screen brightness, while the right hand's vertical thumb position controls volume. The application runs in full screen mode and uses Media Pipe's hand-tracking model to accurately detect up to 21 hand landmarks, ensuring smooth performance even in varying lighting conditions or when the hand is partially obscured. The system is compatible with standard webcams, making it both accessible and cost-effective.

This system offers potential use cases in areas like home automation, public space interaction, and accessibility for users with physical limitations. In environments such as hospitals, the contactless design enhances hygiene by reducing the need for physical touch, while in smart homes, gesture control increases convenience. Future developments could include additional gestures for controlling media or switching applications, integrating voice control for multimodal interactions, and using machine learning to personalize the system based on user preferences. These enhancements would expand its applicability to gaming, virtual reality, and other interactive fields

## 1.2 PROBLEM STATEMENT

As technology becomes increasingly integrated into daily life, the demand for intuitive and accessible interaction methods grows, especially in environments where physical contact with devices may be impractical or impossible. Traditional input methods, such as keyboards, mice, and touchscreens, can be cumbersome or unfeasible in scenarios like cooking, exercising, or for users with physical disabilities. This project addresses the need for a contactless control system that allows users to interact with devices using hand gestures, offering an accessible and convenient solution for adjusting common system functions like screen brightness and volume. By utilizing computer vision and hand gesture recognition, this system aims to provide a seamless user experience that enhances interaction with digital devices while promoting a hygienic and user-friendly environment.

## 1.3 LITERATURE REVIEW

[1] Pavan et al. a real-time gesture recognition system for Arduino platforms using OpenCV and Media Pipe. Analyzing live video streams, the system interprets hand gestures to dynamically control LEDs and servomotors. Bridging the gap between users and technology, it provides an intuitive interface for hardware manipulation. Through computer vision, gesture recognition algorithms, and Arduino programming, the project advances interactive applications in augmented reality and robotics. The abstract encapsulates its essence, emphasizing real-time gesture recognition's focus and its application in seamlessly controlling LEDs and servomotors for Arduino-based systems, assisting with the evolving field of human-computer interaction.

[2] Arunava et al. Hand gesture recognition procedure is a computer vision technique that allows computers to recognize human hand gestures. This can be used for a variety of applications, such as controlling devices, providing feedback, and interacting with virtual reality environments. This paper proposes a novel approach to hand gesture recognition methodology using Python scripts and cv2 (OpenCV), media pipe, and time libraries. The proposed approach uses the Mediapipe library in order to detect as well as track hand landmarks in real time. The number of fingers raised is then calculated based on the detected hand landmarks. A message is then displayed on the screen based on the number of fingers detected and even a message is sent to a registered phone number. The proposed approach was evaluated using a set of data of hand gesture images. The results showed that the proposed approach was able to accurately recognize the number of fingers raised with an accuracy of 90%. The proposed approach exhibits novelty to hand gesture recognition that is simple to implement and can be used for a variety of applications. The approach is relatively accurate, making it a promising technique for hand gesture recognition.

[3] Shruti et al. Gesture recognition is the revolutionary change in the world of Virtual and Augmented reality development allowing seamless non-physical control of computerized devices to create a highly interactive and flexible hybrid reality. Volume has been controlled using buttons and switches in many devices. For enabling real-time experience as the mirrored interaction, gestures introduce a new arena in controlling features such as volume control using palm detection and hand tracking model. Gesture volume control can be utilized in various human-machine interactions such as gaming and media player control. The main objective of the study is to provide an overview of the concept of mounting digitalization utilizing for enhancing augment reality and virtual reality applications. In this proposed work, OpenCV library along with Pycaw and MediaPipe is used to

implement gestures in Human-computer-interaction. The results obtained from the study yield results that were promising.

[4] Pushpak et al. Although the modern wireless mouse or Bluetooth mouse still requires peripherals like energy cells and cards to connect to the computer, it is not entirely peripheral- free. The suggested AI virtual mouse system addresses the aforementioned issues by capturing hand motions with an external or digital camera, then improving the system's accuracy using voice assistants and hand point detection through object recognition. The system's foundation consists of machine learning techniques. To operate a computer, navigate, and execute actions like left- and right-clicking, digital hand motion can be used instead of a physical mouse. The program employs machine learning for hand identification and Python modules, for voice assistance. The suggested approach eliminates the need for human involvement and computer- controlled equipment in the battle against the spread of COVID-19 by implementing fundamental mouse operations together with brightness, volume control, and the ability to manage noise variations.

## 1.4 SOFTWARE REQUIREMENTS

- **Python**: Programming language for implementing the project.
- **OpenCV**: Library for computer vision tasks such as image capture and processing.
- **Media Pipe**: Framework for real-time hand tracking and gesture recognition.
- **NumPy**: Library for numerical operations and data manipulation.
- **PyAuto GUI**: Library for automating GUI interactions, such as volume control.
- **Screen Brightness Control**: Library for adjusting screen brightness.
- **Matplotlib** : Library for plotting and visualizing data or debugging gesture recognition results.
- **IDE/Text Editor**: Development environment (e.g., PyCharm, Visual Studio Code).
- **Operating System**: Compatible OS (Windows, macOS, or Linux).
- **Webcam**: For capturing video input to detect hand gestures.
- **Anaconda**: For managing packages and environments (if preferred).
- **Pip**: Package manager for installing and managing Python libraries.
- **Git**: Version control system to track project progress and collaborate.
- **Virtual Environment (venv)**: For creating isolated Python environments to manage dependencies.

# CHAPTER 2

# PROPOSED METHODOLOGY

## 2.1 BLOCK DIAGRAM

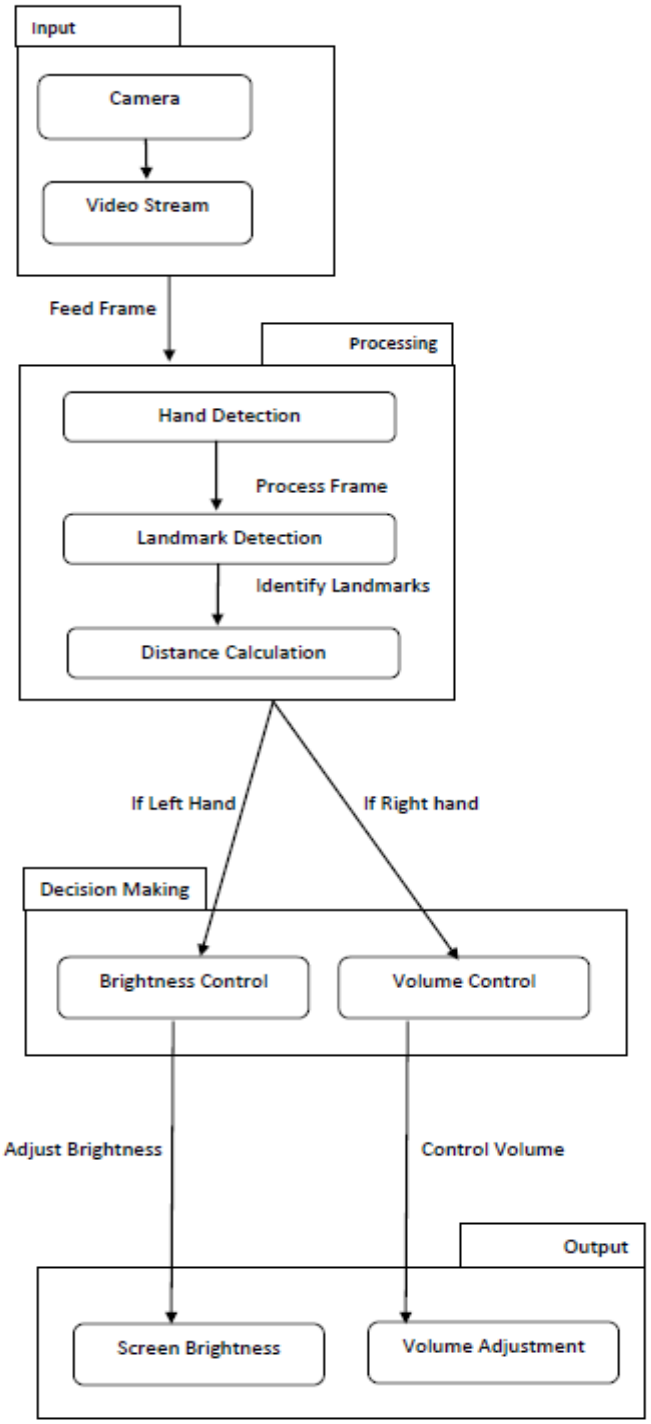


Figure 2.1- Block diagram

The block diagram represents a gesture-based system for controlling screen brightness and volume using hand movements detected by a camera. The input begins with a video stream captured in real time through the camera. This video feed is continuously monitored, and specific frames are processed to detect the presence of a hand. The primary objective of this stage is to identify whether a hand is in view and initiate further processing to determine the gesture being made. This process is foundational for gesture-based control systems, allowing users to interact with their devices without physical contact, making the interface more intuitive and touch-free.

Once the hand is detected, the next phase involves identifying key landmarks or points on the hand that help in interpreting the gesture. Hand landmarks include the position of the fingers, the palm, and other key points that allow the system to understand the gesture's meaning. In this phase, distance calculation between landmarks is performed to accurately assess the gesture. This calculation is vital in distinguishing between various hand positions and movements, such as determining if the hand is open or closed, or whether the hand is moving toward or away from the camera. These calculations directly inform the system whether the action is related to volume or brightness control.

In the decision-making phase, the system categorizes the detected gesture based on the hand used. If the left hand is detected, the system interprets this as a command for brightness control, allowing the user to adjust the screen's brightness based on specific hand movements. On the other hand, if the right hand is detected, the system triggers the volume control mechanism, where hand movements adjust the audio output of the device. The output stage then translates these gestures into actual device adjustments—either changing the brightness level or adjusting the volume. This gesture-based system provides a natural and efficient way for users to control their devices, enhancing user experience by making it more accessible and interactive without physical buttons or touch inputs.

## 2.2 FLOW DIAGRAM

This flowchart outlines the steps of a hand-gesture-based control system for adjusting screen brightness and volume using real-time video capture. The process begins by initializing the necessary libraries and variables, such as those required for video processing (e.g., OpenCV) and hand detection models. Once the setup is complete, the program opens the video capture from the device's camera and creates a fullscreen window to display the live video feed.

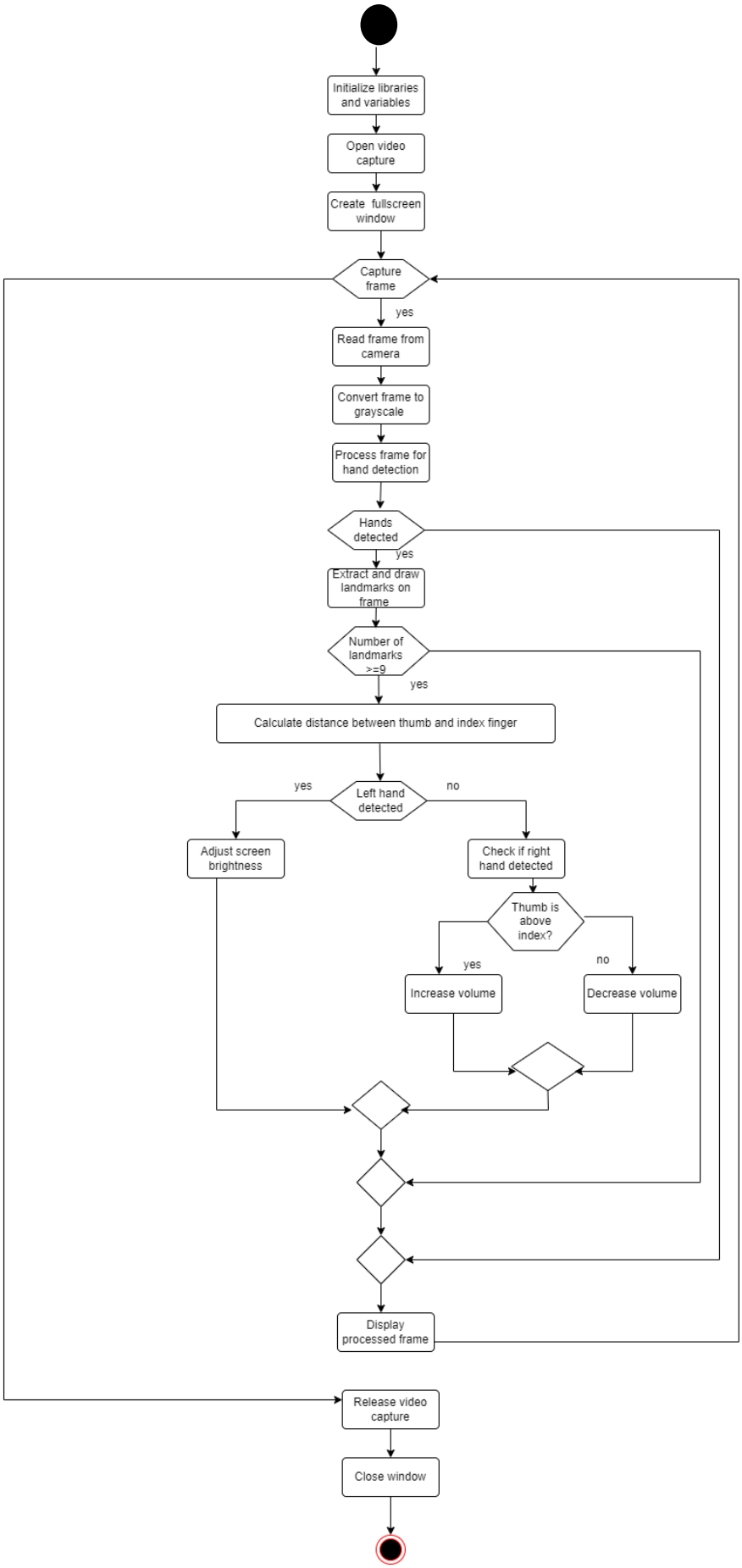


Figure 2.2- Flow diagram

The camera continuously captures frames, which are then read by the program. To simplify processing, each frame is converted to grayscale, reducing computational load and making it easier for the system to detect hands. The frame is then processed to detect the presence of hands using a hand-detection model, such as MediaPipe Hands. If no hands are detected, the system loops back to capture the next frame. When hands are detected, the program extracts key landmarks from the hand, such as the thumb and index finger, and draws them on the frame for visual feedback. The system checks if at least 9 landmarks are detected to ensure accurate calculations for further processing.

Next, the distance between the thumb and index finger is calculated, as this distance will determine the specific action, such as adjusting volume or screen brightness. If the detected hand is the left hand, the system adjusts the screen brightness based on the calculated distance. Larger distances between the thumb and index finger increase the brightness, while smaller distances decrease it. If the right hand is detected, the system checks the position of the thumb relative to the index finger. If the thumb is above the index, the system increases the volume; if the thumb is below, it decreases the volume.

Throughout the process, the program continuously displays the processed frame with the detected landmarks on the fullscreen window. Once the user is done or the system finishes, it releases the video capture device (camera) to free up resources and closes the window, ending the program. This flow enables a hands-free, gesture-based interface for controlling key system features like volume and brightness.

## 2.3 SYSTEM DESIGN

This system diagram illustrates the overall architecture of a hand-gesture-based control system designed for volume and screen brightness adjustment. The system integrates both hardware and software components to process user gestures in real time and trigger corresponding actions.

Starting with the hardware, the user interacts with the system by performing specific gestures in front of a webcam. The webcam captures live video frames of the user’s hand gestures and sends these frames into the software pipeline for processing.

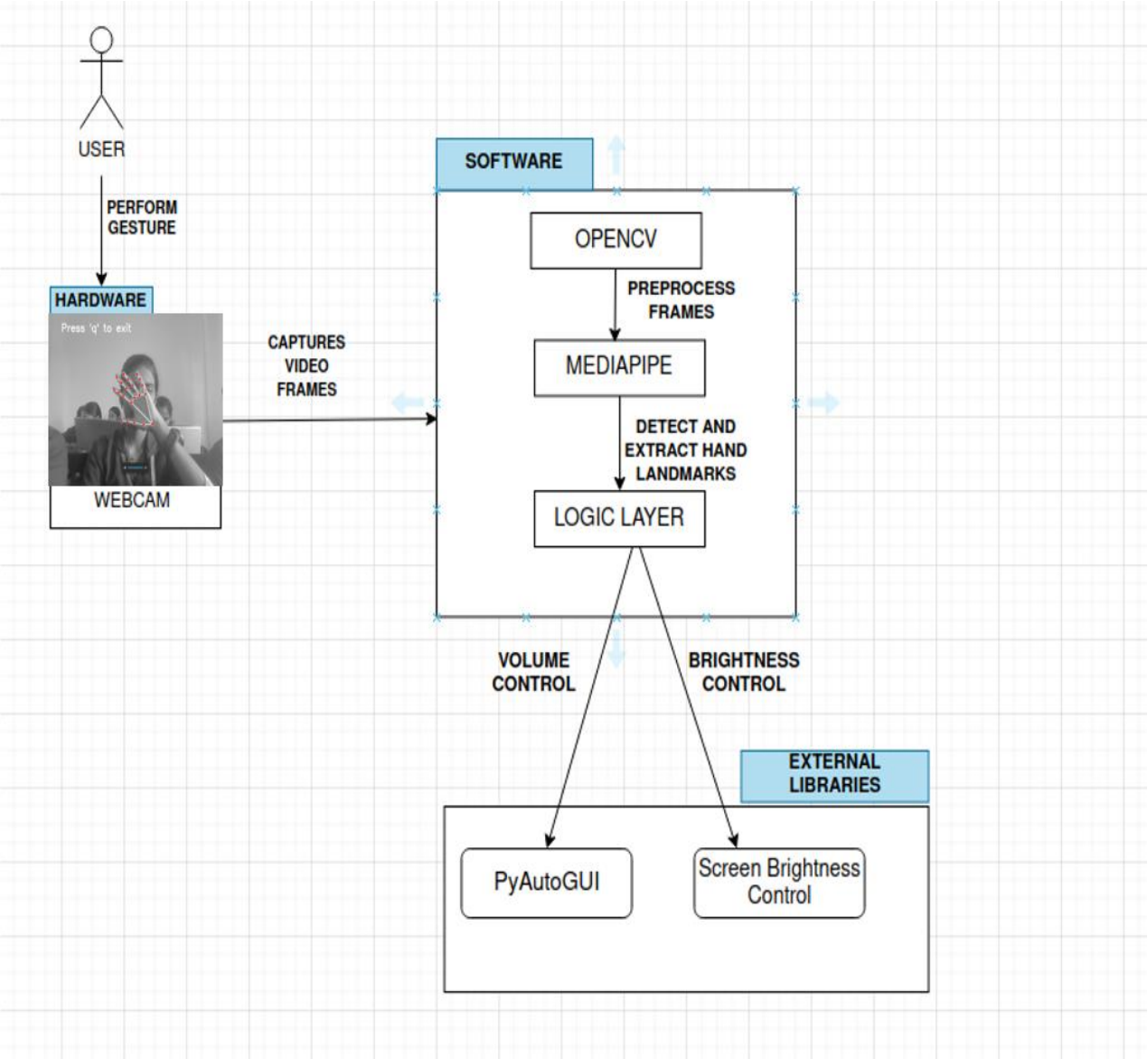


Figure 2.3- System design

The software component is responsible for analyzing the captured video frames. This begins with OpenCV, which preprocesses the incoming frames to make them suitable for gesture detection. Preprocessing may include operations such as resizing, grayscale conversion, or other image enhancements to prepare the frames for accurate analysis.

Once the frames are preprocessed, they are passed into Media Pipe, which handles hand detection. Media Pipe identifies and extracts hand landmarks, such as the positions of the fingers and palm. This extraction of key points is crucial for gesture recognition, as it allows the system to analyze the relative positions and distances between the fingers, such as the thumb and index finger, to determine the user's intent.

The extracted landmarks are then fed into the logic layer, which is responsible for decision-making. Based on the detected hand gestures, the logic layer interprets whether the user wants to adjust the volume or the screen brightness. If the gesture corresponds to volume control, the system interacts with PyAuto GUI, an external library that automates system volume adjustments. For brightness control, the logic layer uses another external library for screen brightness control.

Thus, the system effectively combines video capture, hand detection, and gesture recognition with external libraries to offer a hands-free interface for controlling volume and screen brightness based on user gestures.

## 2.4 MODULES DESCRIPTION

### OpenCV

OpenCV is an open-source computer vision library that provides essential tools for image processing and analysis, facilitating the import of images and video streams for processing. It is particularly useful for implementing methods such as bounding boxes to outline the boundaries of silkworms in images. Additionally, OpenCV enables the drawing and displaying of results, allowing users to visualize detected silkworms and their classifications effectively. Its rich set of functionalities makes it a valuable resource for developing applications that require accurate object detection and analysis.

### PyAutoGUI

PyAutoGUI is a cross-platform Python module that enables programmatic control of the mouse and keyboard, allowing for the automation of graphical user interface interactions. This library is particularly useful for automating repetitive tasks, such as clicking buttons, typing text, and navigating applications. In your hand gesture recognition project, PyAutoGUI is utilized to adjust the system's volume based on detected hand gestures, facilitating a hands-free control experience. With its simple API, PyAutoGUI makes it easy for developers to create automation scripts that enhance productivity and improve user experience.

### Screen Brightness Control

**Screen Brightness Control** is a Python library that allows for easy adjustment of screen brightness on multiple operating systems, including Windows, macOS, and Linux. It enables developers to programmatically modify brightness levels, enhancing user experience in applications that require dynamic adjustments. In your hand gesture recognition project, this library is utilized to change screen brightness based on detected hand gestures, providing hands-free control. By

measuring the distance between specific fingers, the application can interpret user intent to either increase or decrease brightness, making it more convenient and accessible for users to interact with their devices. Its simplicity and cross-platform compatibility make it an essential tool for integrating brightness control features into Python applications.

## 2.5 MODEL EXPLANATION

## MEDIAPIPE

MediaPipe is an open-source framework developed by Google that enables developers to create real-time, cross-platform machine learning pipelines. It is widely used in computer vision tasks, such as hand tracking, face detection, pose estimation, and object detection. MediaPipe simplifies the process of implementing these complex tasks by providing pre-trained models and a highly optimized framework that works efficiently across various devices, including desktops, smartphones, and embedded systems. One of its key features is its ability to process video streams in real-time, making it ideal for interactive applications like gesture recognition, augmented reality, and virtual try-on solutions.

MediaPipe is used to detect and extract hand landmarks in video frames captured by a webcam. It processes the frames to identify key points on the hands, such as fingertips and joints, which can then be used to interpret gestures. The framework’s high accuracy and efficiency make it an excellent choice for building applications that rely on precise, real-time hand gesture recognition. Moreover, MediaPipe's ease of use and flexibility allow developers to quickly integrate complex machine learning models into their projects without needing deep expertise in the underlying algorithms.

### Why Media Pipe for This Project?

MediaPipe is an ideal choice for this project due to its robust real-time performance, ease of integration, and ability to handle complex hand gesture recognition tasks efficiently. The framework is optimized for real-time video processing, ensuring smooth and responsive interactions, which is essential for gesture-based applications. Its pre-trained hand tracking models can accurately detect multiple hands and extract critical landmarks like fingertips and joints, significantly reducing development time while maintaining high accuracy.

Moreover, MediaPipe's cross-platform compatibility allows the project to be deployed seamlessly on a wide range of devices, including desktops, mobile phones, and embedded systems. This flexibility is crucial for ensuring the application can function on different operating systems such as Windows, macOS, Linux, and even mobile platforms like Android and iOS. MediaPipe's simple APIs make it easy to integrate into the project without requiring extensive knowledge of machine learning or computer vision, accelerating development.

Additionally, MediaPipe is resource-efficient, enabling the project to run smoothly on devices with limited processing power, such as mobile devices. Its customizable nature also allows the framework to be tailored to specific gesture recognition tasks, providing a scalable solution for current and future requirements. These reasons make MediaPipe the perfect fit for this hand gesture recognition project, balancing performance, ease of use, and flexibility**.**

## Mediapipe Architecture

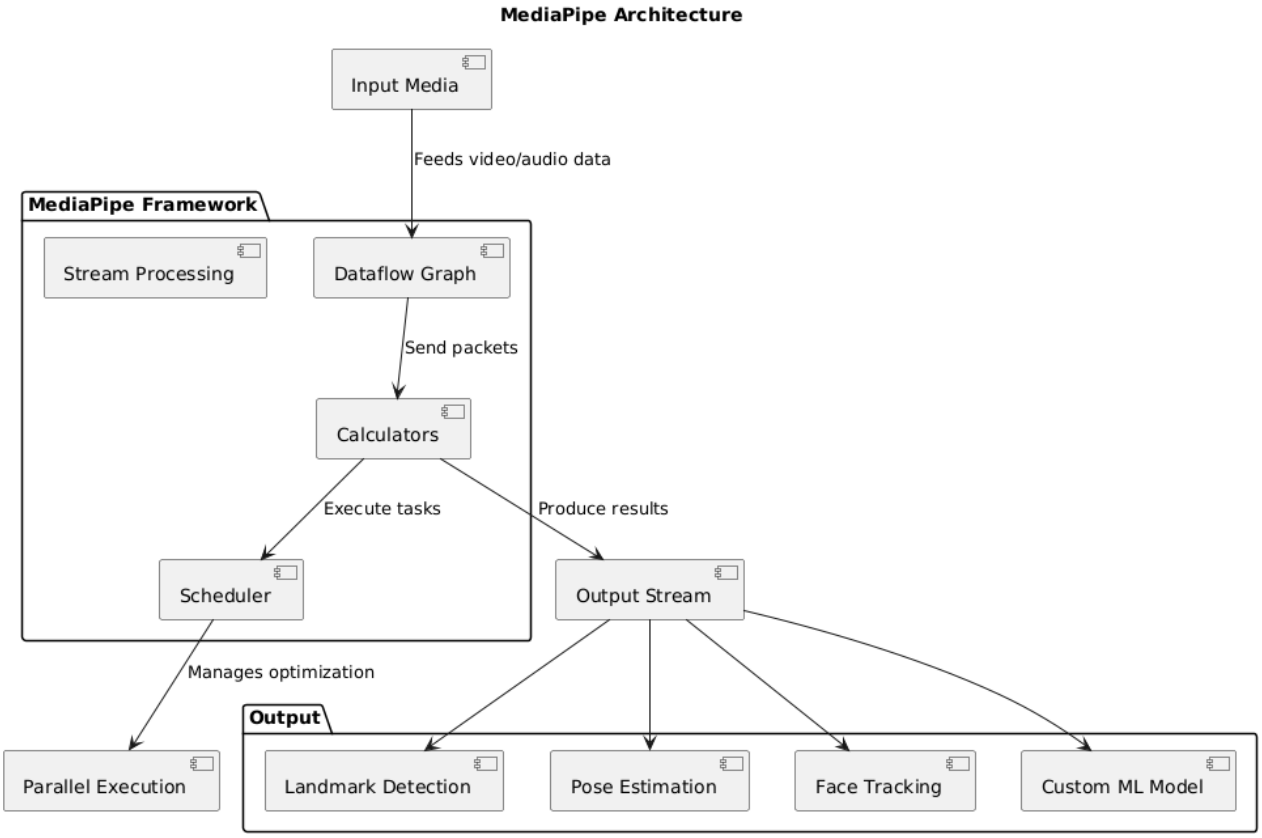


Figure 2.4-Mediapipe Architecture

MediaPipe framework is designed to efficiently handle media data (such as video or audio) and produce outputs such as landmark detection, pose estimation, or results from custom machine

learning models. The framework is modular, allowing for real-time performance and scalability across different platforms like desktop and mobile devices.

1. **Input Media**: The process begins with media input—video, audio, or both—being fed into the MediaPipe pipeline. This data can come from various sources like webcams, cameras, or pre-recorded media.
2. **MediaPipe Framework**:
    - **Stream Processing**: The input media is captured and prepared for further processing.
    - **Dataflow Graph**: This is the heart of the framework, where the media is divided into packets and flows through a directed graph. The graph dictates how the media data will be processed.
    - **Calculators**: These are the core components that perform operations on the input data, such as detecting hand landmarks or filtering noise. Each calculator is responsible for a specific task in the processing pipeline.
    - **Scheduler**: This component manages the execution of tasks in the most optimized way, enabling parallel processing where possible. The scheduler ensures that calculators execute efficiently for real-time results.
3. **Output Stream**: After processing, the results are sent to the output stream. The results can be any processed data, such as detected hand landmarks or pose estimations.
4. **Outputs**: The framework supports multiple types of outputs:
    - **Landmark Detection**: For example, detecting and tracking hand or facial landmarks.
    - **Pose Estimation**: Estimating the body pose based on video input.
    - **Face Tracking**: Identifying and tracking facial features.
    - **Custom ML Models**: MediaPipe allows developers to integrate custom machine learning models to tailor the system to specific applications.
5. **Parallel Execution**: The architecture supports parallel execution, meaning that several tasks can be processed simultaneously. This enhances the performance, making MediaPipe suitable for real-time applications like gesture or face tracking.

# CHAPTER 3

# RESULTS AND DISCUSSION

## 3.1 SOURCE CODE

```
from math import hypot
import cv2
import mediapipe as mp
import numpy as np
import pyautogui
import screen_brightness_control as abc
mp_hands = mp.solutions.hands
hands=mp_hands.Hands(static_image_mode=False,max_num_hands=2,min_detecti
on_confidence=0.5, min_tracking_confidence=0.5)
mp_drawing = mp.solutions.drawing_utils
cap = cv2.VideoCapture(0)
cv2.namedWindow('Hand Gesture', cv2.WND_PROP_FULLSCREEN)
cv2.setWindowProperty('HandGesture',cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_F
ULLSCREEN)
while True:
    ret, frame = cap.read()
    if not ret:
        break
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame_gray_bgr = cv2.cvtColor(frame_gray, cv2.COLOR_GRAY2BGR)
   results=hands.process(cv2.cvtColor(frame_gray_bgr,cv2.COLOR_BGR2RGB))
    lmList = []#(x, y positions of key points)
    if results.multi_hand_landmarks
        for hand_landmarks in results.multi_hand_landmarks:
            for id, lm in enumerate(hand_landmarks.landmark):
```

```
                h, w, _ = frame.  shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                lmList. append([id, cx, cy])
            mpdrawing.draw_landmarks(frame_gray_bgr,hand_landmarks,
mp_hands.HAND_CONNECTIONS)
            if len(lmList) >= 9:
                x1, y1 = lmList[4][1], lmList[4][2]
                x2, y2 = lmList[8][1], lmList[8][2]
                length = hypot(x2 - x1, y2 - y1)
                if lmList[0][1] < w/2:
                    brightness = np.interp(length, [15, 220], [0, 100])
                    abc.set_brightness(int(brightness))
                if lmList[0][1] > w/2:
                    if lmList[4][2] > lmList[8][2]:
                        pyautogui.press('volumeup')
                    else:
                        pyautogui.press('volumedown')
    cv2.putText(frame_gray_bgr,  "Press  'q'  to  exit",  (50,  50),
cv2.FONT_HERSHEY_SIMPLEX,1, (255, 255, 255), 2, cv2.LINE_AA)
    cv2.imshow('Hand Gesture', frame_gray_bgr)
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

## 3.2 RESULTS


Figure 3.1-Brightness control1

In **Figure 3.1**, the user's hand gesture causes an **increase in brightness**. The distance between the thumb and index finger is relatively larger, leading the system to interpret this as a command to raise the screen brightness, as indicated by the brightness bar on the display.


Figure 3.2- Brightness control2

**Figure 3.2** depicts a **decrease in brightness**. Here, the user's hand gesture changes slightly, with the thumb and index finger closer together, which the system detects as a signal to lower the brightness level, as shown by the reduced brightness bar.

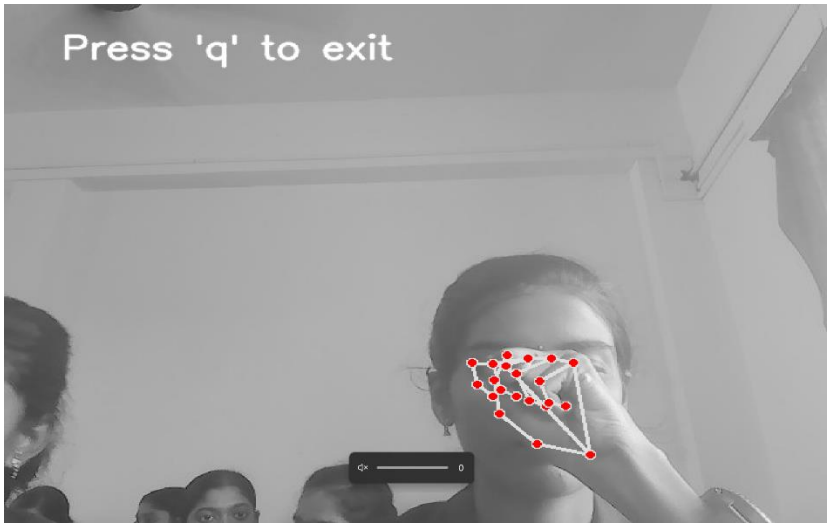


Figure 3.3-Volume control1

In **Figure 3.3**, the user's hand gesture signals a **decrease in volume**. The system detects that the index finger is placed above the thumb, indicating a command to lower the volume. This is shown by the decrease in the volume bar on the display, effectively demonstrating how hand gestures are translated into volume control.


Figure 3.4- Volume control2

n **Figure 3.4**, the user's gesture leads to an **increase in volume**. Here, the position of the thumb is above the index finger, signifying a command to raise the volume, as seen in the increased level of the volume bar on the screen. This image highlights the system's responsiveness to hand movements and its ability to accurately adjust the volume based on intuitive gestures.

## 3.3 MODEL EVALUATION

In Mediapipe, Precision (P), Recall (R) and F1 curves are used to evaluate model performance

- Precision is the ratio of true positive predictions to all predicted positives, indicating the accuracy of positive predictions.
- Recall measures the ratio of true positive predictions to all actual positives, reflecting the model's ability to identify relevant instances.
- The F1 score is the harmonic mean of precision and recall, providing a balanced measure of a model's performance, especially with imbalanced classes.

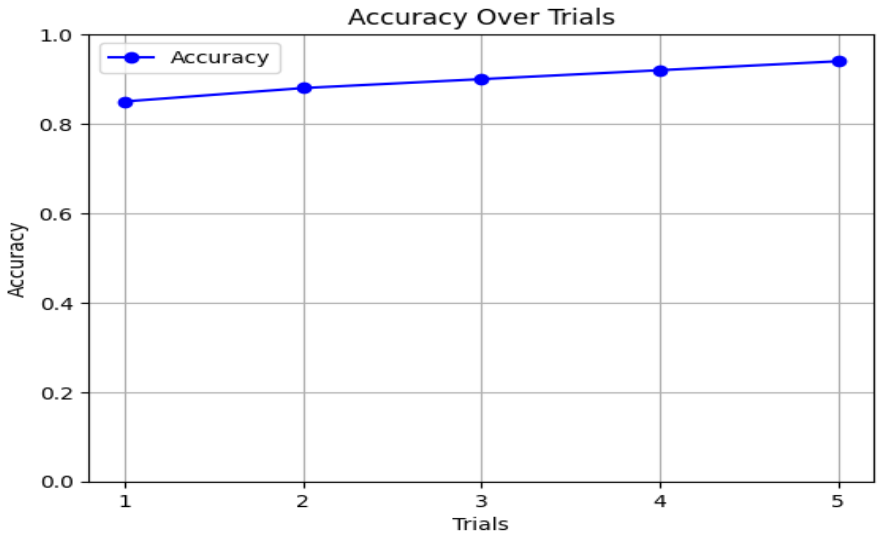


Figure 3.5-Accuracy over trials

Figure 3.5 shows the accuracy of your hand gesture control system over five trials, with accuracy improving from 0.85 to 0.94. This indicates that as you continue testing and refining the system, its ability to correctly recognize gestures improves. The upward trend suggests better performance over time, which reflects the system's increasing reliability in detecting hand gestures for tasks like controlling brightness and volume. This is important for ensuring consistent user interactions in your project.

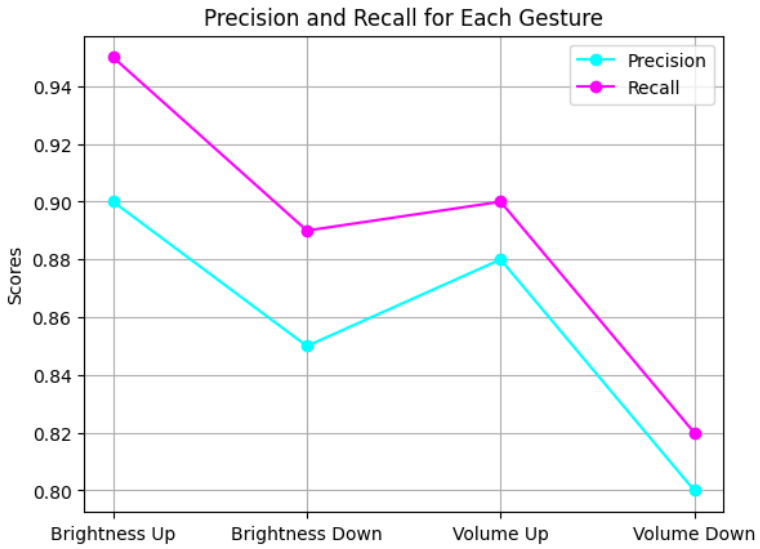


Figure 3.5-Precision and recall

Figure 3.5 represents the precision and recall scores for each gesture in your hand gesture recognition system. It compares how accurately the system detects gestures like "Brightness Up," "Brightness Down," "Volume Up," and "Volume Down." Precision measures the correctness of the detected gestures, while recall assesses how many of the correct gestures were identified. The plot shows slight variations in performance across gestures, with "Brightness Up" having higher recall and "Volume Down" scoring lower in both metrics, indicating the need for potential improvements in recognizing those gestures.

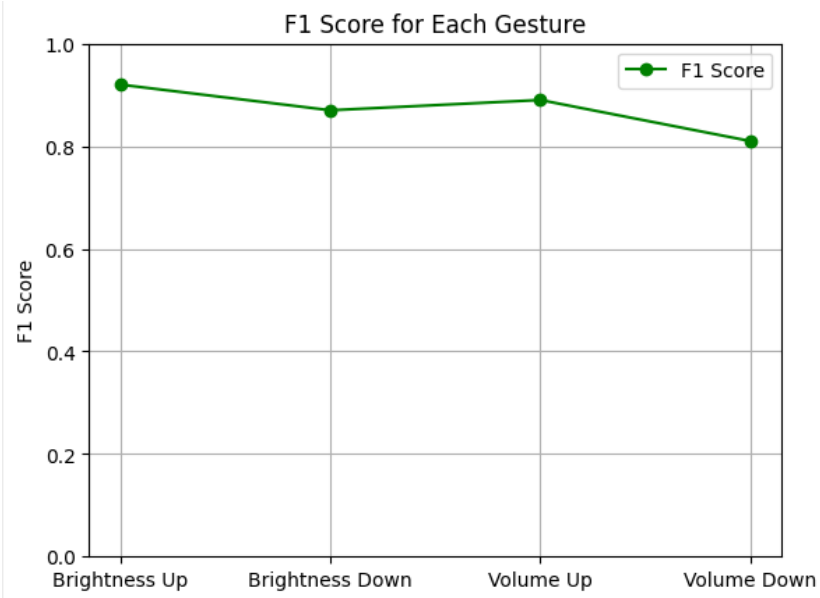


Figure 3.6 -F1 score for each gesture

Figure 3.6 shows the F1 scores for each gesture in your hand gesture recognition system, including "Brightness Up," "Brightness Down," "Volume Up," and "Volume Down." The F1 score combines both precision and recall to provide a balanced metric of performance. The system achieves a high F1 score for "Brightness Up" (0.92), but scores slightly lower for "Volume Down" (0.81). This indicates that while the system performs well overall, there may be room for improvement in detecting certain gestures, especially those related to volume control.

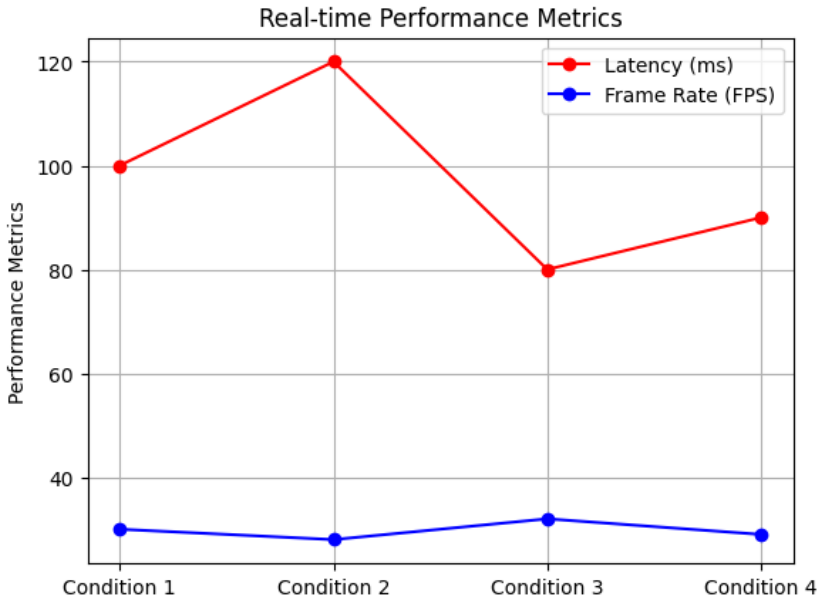


Figure 3.7-F1 Real-time performance metrics

Figure 3.7 illustrates the real-time performance metrics of your hand gesture recognition system under four different conditions. The red line represents latency (in milliseconds), showing how quickly the system processes hand gestures, while the blue line indicates the frame rate (in FPS), reflecting how smoothly the system runs. In Condition 2, the latency spikes to 120 ms, causing slower gesture response, but the frame rate remains relatively stable across all conditions, hovering around 30 FPS. This suggests that while the system maintains smooth video processing, latency fluctuations can affect responsiveness in certain scenarios.

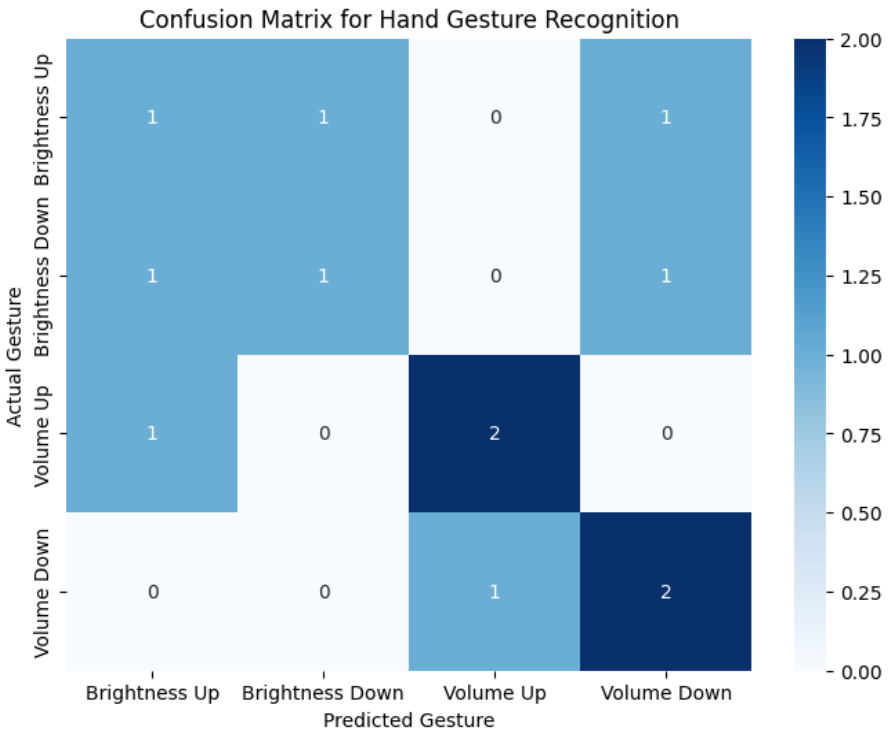


Figure 3.8-F1 Real-time performance metrics

In the Hand Gesture-Based Control System, the confusion matrix shown represents the performance of the gesture recognition model by comparing actual hand gestures (like "Brightness Up," "Brightness Down," "Volume Up," and "Volume Down") with the predicted gestures. The matrix highlights correct predictions (diagonal cells) and misclassifications (off-diagonal cells). For example, "Brightness Down" has been misclassified as "Volume Down" in one instance, while "Volume Up" is correctly predicted twice. This matrix helps to evaluate and improve the system's accuracy in interpreting gestures and can guide future optimizations in hand tracking or gesture classification algorithms.

# CHAPTER 4

# CONCLUSION

The Hand Gesture-Based Control System demonstrates a significant advancement in user interaction with digital devices by leveraging computer vision and gesture recognition technologies to create a contactless and intuitive interface. By integrating key components such as MediaPipe for real-time hand tracking, OpenCV for video processing, and PyAutoGUI for system control, the system translates hand gestures into commands for tasks like adjusting screen brightness and controlling volume. This not only enhances accessibility for users with physical limitations but also promotes a hygienic interaction method suitable for various environments, including hospitals, public spaces, and shared workstations. With its ability to function using a standard webcam and simple hardware, the system is scalable and can be easily deployed across different platforms, making it accessible to a wide audience.

Looking ahead, potential enchancement could involve adding more gestures for a wider range of functions, improving recognition accuracy in diverse lighting conditions, and incorporating multimodal interactions such as voice commands or facial recognition. The system could also evolve to recognize more complex hand movements, like swipes or multi-hand gestures, opening up possibilities for immersive experiences in virtual or augmented reality environments. Overall, this project contributes to the growing field of gesture recognition and human-computer interaction, paving the way for more seamless and efficient user experiences while offering new, intuitive ways to interact with technology.

# REFERENCES


[1] Salihbašić and T. Orehovački, "Development of Android Application for Gender Age and Face Recognition Using OpenCV" in 2019 42nd International Convention on Information and Communication Technology Electronics and Microelectronics (MIPRO), Opatija, Croatia, pp. 1635-1640, 2019.

[2] V. K. Bhanse and M. D. Jaybhaye, "Face Detection and Tracking Using Image Processing on Raspberry Pi", *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 1099-1103, 2018.

[3] Rafael C. Gonzalez, Digital image processing, Pearson education india

[4] S. N. Karishma and V. Lathasree, "Fusion of skin color detection and background subtraction for hand gesture segmentation", *International Journal of Engineering Research and Technology*, vol. 3, no. 2, pp. 1835-1839, 2014.

[5] Paul Viola and Michael Jones, "Rapid object detection using a boosted cascade of simple features", *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1, 2001.

[6] Ravindra Bula, Dipalee Golekar, Rutuja Hole et al., "Sign Language Recognition based on Computer Vision", *International Journal of Creative Research Thoughts (IJCRT)*, vol. 10, no. 5, May 2022, ISSN 2320-2882.

[7] Sherin Mohammed Sali and V H Preetha, "Hand Gestures - Virtual Mouse for Human Computer Interaction", *International Conference on Smart Systems and Inventive Technology (ICSSIT 2018)*.

[8] G. R. S. Murthy and R. S. Jadon, "A Review Of Vision Based Hand Gestures Recognition", *International Journal of Information Technology and Knowledge Management*, vol. 2, no. 2, pp. 405-410, July-December 2009.

[9] S. U. Dudhane, "Cursor control system using hand gesture recognition", *IJARCCE*, vol. 2, no. 5, 2013.

[10] K. P. Vinay, "Cursor control using hand gestures", *International Journal of Critical Accounting*, vol. 0975-8887, 2016.