GESTURE-BASED VOLUME CONTROL USING FINGER COUNTING

(OPENCV • MEDIAPIPE • PYCAW • STREAMLIT)

Mentor: Dr. D. Bhanu Prakash (MENTOR – INFOSYS)

Presenting by: Group D

PROBLEM STATEMENT

How can we control system volume in real time using only hand gestures, without touching the machine, using commodity hardware like a basic webcam?

Why Gesture-Based Volume Control?
Traditional laptop/PC volume control methods are **slow, inefficient, and break the workflow**:

- Physical keyboard buttons wear out and are not ergonomic.
- Alt-tabbing or clicking on the volume slider interrupts productivity.
- External devices (mouse/keyboard) add delay in fast-paced tasks.

SYSTEM OVERVIEW

1. **Hand Gesture Input**
   User shows hand to the webcam.

2. Frame Capture (OpenCV)
- Reads video frames in real time
- Converts BGR → RGB for processing
- Handles frame flipping and display

3. Hand Landmark Detection (MediaPipe Hands)
- Extracts **21 precise hand landmarks**
- Identifies finger tips, joints, and orientation
- Works robustly even with motion or poor lighting

4. Finger Counting Logic
- Thumb detection using side orientation
- Remaining fingers checked using tip/base landmark positions
- Produces output: **0, 1, 2, 3, 4, or 5 fingers**

5. Volume Mapping Module
- Converts finger count → volume percentage
- Formula: *(fingers / 5) × 100*
- Output range: **0%–100%**

6. System Audio Control (PyCAW + COM)
- Communicates with Windows Audio Endpoint API
- Sets **system volume + mic volume**
- Uses COM initialization for stability

7. Live Dashboard (Streamlit)
- Shows camera feed
- Displays gestures, volume level, FPS, latency
- Plots real-time volume history

HARDWARE & SOFTWARE REQUIREMENTS

Hardware Requirements

•**Webcam**
Required for real-time hand tracking and gesture detection.

•**Windows PC/Laptop**
PyCAW uses Windows Core Audio API → volume control requires Windows.

Software Requirements

•**Python 3.8+** → Core runtime
•**OpenCV** → Video capture, frame processing
•**MediaPipe** → 21-point hand landmark detection
•**PyCAW** → System & mic volume control through COM audio API
•**Streamlit** → Real-time dashboard UI
•**NumPy / Pandas** → Data operations + graph plotting
•**Comtypes** → Initialize COM for PyCAW (avoids crashes)

KEY LIBRARIES USED

LIBRARY	PURPOSE
OpenCV	Captures video frames from the webcam, Handles BGR ↔ RGB conversion, Performs frame flipping & drawing operations, Enables real-time processing at stable FPS
MediaPipe Hands	Detects **21 hand landmarks** with high accuracy, Works in real time on CPU, Provides tip/base landmark positions used for finger counting
PyCAW	Interfaces with **Windows Audio Endpoint API,** Controls system volume & microphone volume, Uses COM components like IAudioEndpointVolume.
Streamlit	Builds real-time dashboard UI, Displays camera feed, gesture labels, FPS, latency Supports history graphs and interactive controls.
NumPy & Pandas	Handles numeric volume arrays, Creates dataframes for plotting volume history Fast list operations for smooth graphing
Comtypes	Initializes COM (CoInitialize, CoUninitialize), Ensures stable communication with Windows audio API

OPENCV: ROLE IN THE PROJECT

OpenCV handles **all camera-related operations**, making real-time gesture recognition possible.

Key Responsibilities of OpenCV

•**Video Capture (`cv2.VideoCapture`)**

Reads continuous frames from the webcam with minimal latency.

•**Frame Preprocessing**

- `cv2.flip()` → Mirror the image for natural hand interaction
- `cv2.cvtColor()` → Convert **BGR** → **RGB** for MediaPipe processing
- Maintains consistent frame format for UI display

•**Drawing Operations**

- Renders hand landmarks
- Displays detection overlays
- Provides visual feedback during gesture recognition

Why OpenCV Works Well Here

•Lightweight & fast on CPU

•Compatible with real-time applications

•Integrates smoothly with MediaPipe's input pipeline

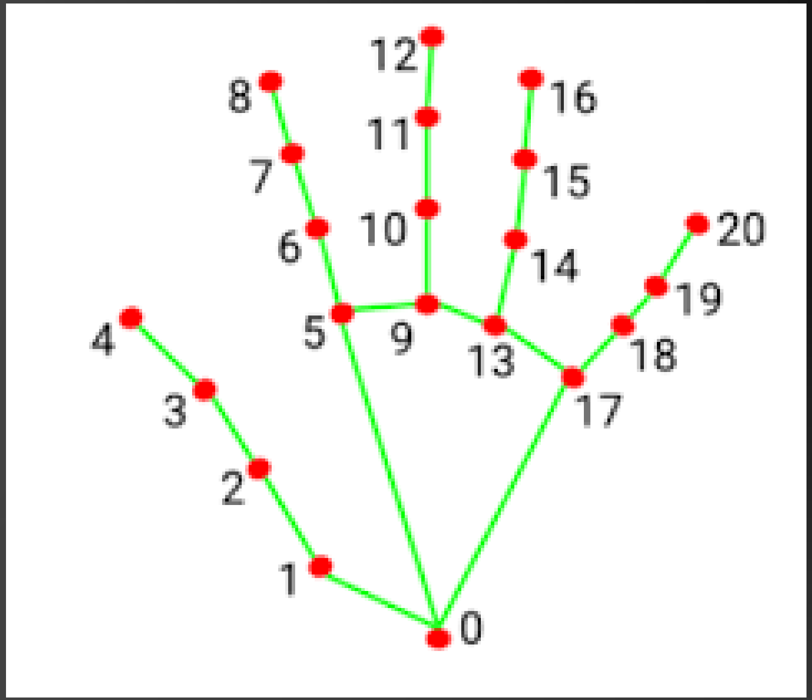
•Highly stable for continuous streaming loops

MEDIAPIPE HANDS: LANDMARK DETECTION ENGINE

MediaPipe uses a machine-learning pipeline to detect **21 precise hand landmarks** in real time using only CPU.

It provides:

- Fingertip positions
- Joint coordinates
- Hand orientation
- Tracking stability even with fast movement



Why It Is Used in This Project

- **High accuracy** compared to contour or threshold methods
- **Fast performance** (30–60 FPS on normal laptops)
- **No special hardware needed**
- Outputs normalized coordinates (x, y, z), perfect for gesture logic

Landmarks Used for Finger Counting

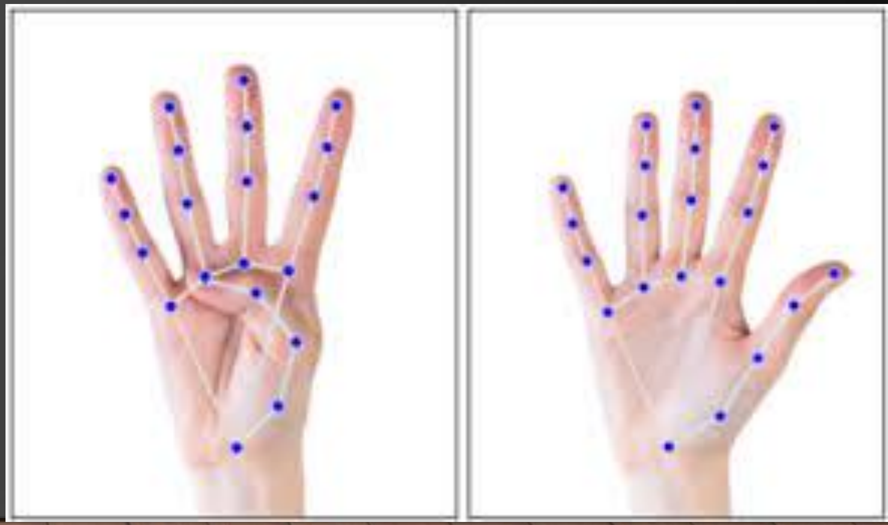
- **Thumb:** Uses landmark indices 4 (tip) & 3 (IP) + wrist (0)
- **Fingers:** Uses tip landmarks (8, 12, 16, 20)
- Compared against their base joints (6, 10, 14, 18)
- This allows deterministic detection of whether each finger is **up or down**.

FINGER COUNTING LOGIC

Core Idea
Finger counting is done by comparing the **tip landmark** of each finger with its **lower joint landmark**.
If the tip is above the base → finger is considered **UP**.

Thumb Detection
Thumb is special because it moves sideways, not vertically.
•Uses landmarks **4 (thumb tip)** and **3 (thumb IP)**
•Compares their **x-position** relative to **wrist (0)**
•Determines whether thumb is extended or folded based on hand orientation

Other Fingers
For index, middle, ring, pinky:
•Tip landmarks: **8, 12, 16, 20**
•Base landmarks: **6, 10, 14, 18**
•Logic:
If tip.y < base.y → Finger is UP
This makes detection stable across distances and angles.



Final Finger Count
Number of raised fingers = **sum of all detected UP fingers**
Output range: **0–5**

MAPPING FINGER COUNT TO VOLUME

The system converts the number of raised fingers directly into a volume percentage.

Resulting Mapping Table

Fingers Up	Volume Level
0	0% (Mute)
1	20%
2	40%
3	60%
4	80%
5	100%

Why This Mapping Works Well

- Fully deterministic → zero ambiguity
- Fast enough for real-time control
- No calibration needed
- Very easy for users to understand

Application of Mapping

- Same mapping is applied to **system volume** and **microphone volume**
- Uses PyCAW to set OS-level audio instantly

PYCAW & WINDOWS AUDIO API

What PyCAW Does

PyCAW (Python Core Audio Windows) provides access to the **Windows Audio Endpoint API**, allowing your program to:

- Control **system volume**
- Control **microphone volume**
- Read current volume levels
- Interact with audio devices at OS level

Key Components Used

- **IMMDeviceEnumerator**
  Finds the default input/output audio devices.
- **IAudioEndpointVolume**
  The interface responsible for setting:
  - Scalar volume (0.0 → 1.0)
  - Percent volume (0% → 100%)
- **CLSCTX_ALL**
  Required for COM component activation.

COM Initialization

The project uses:

- `CoInitialize()`
- `CoUninitialize()`

These prevent:

- COM threading errors
- Crashes when volume device is unavailable
- Failures during repeated volume updates

This is why the volume control works **smoothly** even during continuous updates.

Why PyCAW Is Essential

- Direct OS-level volume control
- Faster than using media keys
- More accurate (sets exact scalar values)
- Allows separate control of mic + speaker

STREAMLIT UI BREAKDOWN

What the UI Does

Camera Interface

- Displays live camera feed inside a glass-style card
- Provides a Start / Stop button to control capture
- Shows hand landmarks (dots) generated by MediaPipe

Volume Visualization

- Animated circular volume gauge
- Gradient progress bar showing real-time volume
- Volume updates automatically based on finger count (0–5)

Status Metrics

- Volume percentage displayed in real time
- Finger count detected through MediaPipe
- FPS shown for performance monitoring

Modern Glass UI

- Dark gradient theme
- Blurred glass cards
- Custom badges, round buttons, and animated bars

Backend and Thread Handling

- QThread manages camera and gesture processing
- Thread-safe signals prevent UI freeze
- Controlled delay ensures smooth, stable performance

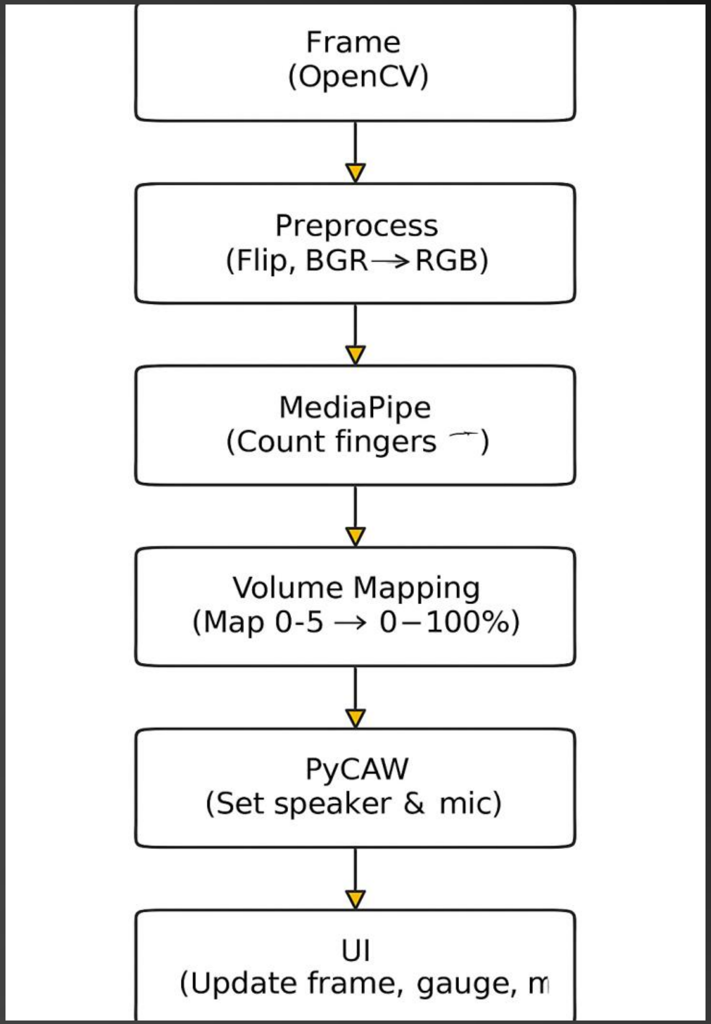
Why It Matters

- Real-time gesture recognition with volume control
- Clean, responsive animated UI
- Stable multithreaded architecture without web technologies

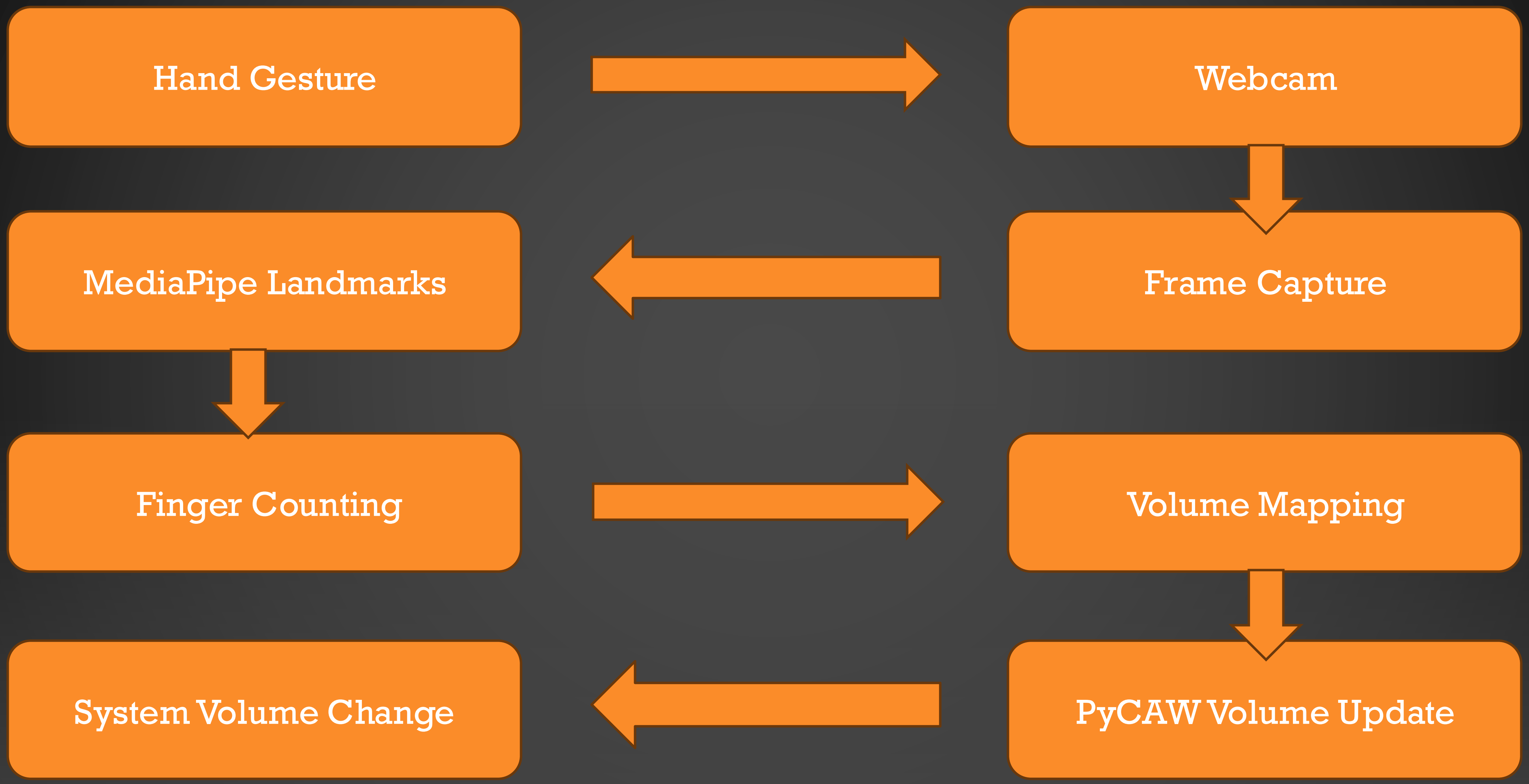
MAIN PROCESSING LOOP

Real-Time Pipeline

1. **Read frame** from webcam using OpenCV
2. **Flip horizontally** for mirror-view interaction
3. **Convert BGR → RGB** for MediaPipe compatibility
4. **Process frame in MediaPipe** to detect 21 hand landmarks
5. **Count fingers** using landmark comparison logic
6. **Map finger count → volume percentage**
7. **Update volume** (system + mic) using PyCAW
8. **Update UI** (frame, text, graphs, metrics)
9. **Introduce small sleep** to reduce CPU usage



DEMO FLOW SLIDE



USE CASES

Practical Applications

Gaming: Adjust volume without removing hands from keyboard/mouse

Music editors / DJs: Touch-free mixing

Accessibility: Helps users with motor limitations

Public kiosks / displays: Touchless control

AR/VR systems: Natural gesture-based UI

Hands-free workflow: Cooking, workshops, medical environments

CHALLENGES

Key Issues

* Lighting variations caused unstable detection

* Skin-tone background clash affected tracking

* Latency (>16ms) on older webcams

* COM errors when audio devices changed

* Multi-hand interference

* Thumb detection differences between left & right hand

How we Solved Them

* Tuned MediaPipe confidence values

* Used contrasting backgrounds

* Limited to single-hand tracking

* Added COM safe wrappers (try/except + reinit)

* Improved thumb logic using wrist comparison

* Lowered processing overhead to maintain FPS

CONCLUSION

* Hand gestures can replace traditional physical volume controls

* OpenCV + MediaPipe deliver real-time, stable hand tracking

* PyCAW provides powerful OS-level audio control

* Streamlit makes it a polished, interactive System

* This project demonstrates a clear pathway toward gesture-based interfaces used in smart devices, AR/VR, and assistive technology