International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Landmark-Driven Touchless Audio Control Using OpenCV and MediaPipe

Jayesh Balu Sonar[1], Akshita Santosh Patil[2], Prajakta Yuvraj Borse[3], Jagdish Nandu Jadhav[4], Aditya Jagdish Shinde[5]

[1,2,3,4,5]Department of Computer Science Shri Shivaji Vidya Prasarak Sanstha's, Bapusaheb Shivajirao Deore College of Engineering, Dhule, Maharashtra, India

Abstract

Touchless human-computer interaction provides a natural way to control digital systems without relying on physical input devices. This paper presents a landmark-driven audio inter- action framework that uses real-time hand gesture recognition for desktop volume control. The proposed prototype uses OpenCV for webcam frame acquisition and interface rendering, Medi- aPipe for hand landmark detection, and Windows-compatible audio control libraries for executing system-level audio opera- tions. Hand landmark relationships are used to classify gesture states and map them to commands such as volume adjustment, mute/unmute control, and boosted audio response. The system also includes a dashboard interface that displays gesture status, hand detection state, volume percentage, frame-rate behavior, and interaction history. A pilot evaluation was structured using lighting variation, camera distance, and real-time performance as key parameters. The results indicate that the prototype performs effectively under normal lighting and medium camera distance, while accuracy decreases in low-light and long-distance conditions. In addition to the implemented desktop prototype, the paper discusses browser extension readiness as a future direction for web-based media control. The proposed system demonstrates the feasibility of a lightweight, low-cost, and webcam-based touchless audio-control interface for multimedia interaction and accessibility-oriented applications.

Keywords: Hand gesture recognition, MediaPipe, OpenCV, touchless interaction, audio control, human-computer interaction, browser media control.

INTRODUCTION

Human-computer interaction has gradually moved beyond traditional input devices such as keyboards, mice, and physical buttons. Camera-based gesture interaction provides a more natural way to control digital systems, especially in multime- dia, accessibility, smart device control, virtual environments, and touchless interface scenarios. Hand gestures are partic- ularly useful because they are expressive, intuitive, and can be detected using standard RGB cameras without requiring specialized external sensors. Camera-based gesture recognition has been widely studied as a computer-vision-based HCI approach for natural and touchless interaction [7], [8].

Audio control is a common desktop operation that is usually performed through keyboard shortcuts, mouse-based controls, or hardware volume keys. Although these methods are effec- tive, they still require physical contact with an input device. In situations such as presentations, media playback,

remote interaction, accessibility support, or hands-free computing, touchless audio control can provide a more flexible interaction method.

This paper presents a real-time touchless audio interaction framework that uses OpenCV and MediaPipe to detect hand landmarks from webcam frames and maps gesture states to audio-control operations. The implemented prototype supports volume adjustment, mute/unmute operation, boosted audio response, real-time dashboard visualization, and frame-rate monitoring. Instead of presenting the system only as a ba- sic volume controller, this work frames the prototype as a lightweight human-computer interaction module that can be extended toward browser media control and custom gesture- based interaction.

The major contributions of this work are as follows:

- A real-time webcam-based touchless audio interaction prototype using hand landmark tracking.
- A lightweight rule-based gesture-to-command mapping approach for volume adjustment, mute/unmute control, and boosted audio response.
- A dashboard-oriented feedback interface for visualizing gesture status, detection state, volume percentage, and frame-rate behavior.
- A pilot evaluation framework covering lighting variation, camera distance, and real-time performance.
- A browser-extension-ready architecture for extending the system toward web-based media control.

Related Work

Hand gesture recognition has been widely explored as a natural mode of human-computer interaction (HCI). Tradi- tional approaches commonly relied on image segmentation, contour extraction, skin-color filtering, edge detection, or Haar-cascade-based recognition. These techniques are com- putationally simple and can be implemented using standard computer vision libraries; however, their performance is of- ten affected by illumination changes, complex backgrounds, hand orientation, camera resolution, and segmentation errors. For real-time systems, these limitations become more visible because the hand region must be detected continuously across multiple video frames.

Recent research has shifted toward landmark-based and learning-based approaches. MediaPipe Hands introduced an efficient on-device hand tracking pipeline that combines palm detection with a hand landmark model. The palm detector first identifies the hand region, and the landmark model then estimates structured hand key points from the detected region. This approach reduces the dependency on manual segmentation and provides a consistent representation of finger positions, which is useful for gesture interpretation [1]. The availability of hand landmarks allows developers to design gesture-control systems using geometric relationships between points such as finger tips, joints, and relative hand positions.

Custom hand gesture recognition has also been studied for applications where predefined gesture sets are not sufficient. Uboweja et al. presented an on-device real-time custom hand gesture recognition framework that allows users to train and deploy gesture recognition models with limited samples per gesture [2]. This direction is important because practical ges- ture interfaces often require application-specific commands. For example, a media-control system may need gestures for volume adjustment, mute, play, pause, and browser-level control, while an accessibility-focused interface may require personalized gestures based on user comfort.

Temporal gesture recognition methods extend static landmark-based systems by analyzing sequences of

frames. Biswas et al. proposed a MediaPipe with LSTM architecture for real-time gesture recognition, where MediaPipe is used for landmark extraction and LSTM is used to model gesture movement over time [3]. Such temporal models are suitable for dynamic gestures such as swipes, circular motions, or repeated tap-like movements. However, they usually require datasets, training time, and additional computational complexity com- pared with rule-based landmark mapping.

Several works have applied hand gesture recognition to vol- ume control. Arun et al. presented a hand gesture recognition and volume-control system using OpenCV-based techniques and hand gesture processing [4]. Tamilkodi et al. also dis- cussed gesture recognition for volume control and broader media interaction commands such as play, pause, volume up, volume down, and resume [5]. These works show that gesture- based media control is a practical application area. However, many existing systems are presented mainly as direct volume- control implementations and provide limited discussion about dashboard feedback, extension-ready architecture, browser- level interaction, or future custom gesture personalization.

The proposed work builds on the landmark-based direction by using MediaPipe hand landmarks with OpenCV-based real- time processing. Unlike conventional single-purpose volume- control demonstrations, this work frames the prototype as a touchless audio interaction framework. The implemented sys- tem supports desktop audio control through gesture mapping, visual dashboard feedback, mute/unmute operation, boosted audio response, and real-time performance monitoring. In addition, the paper discusses a browser-extension-ready archi- tecture that can extend the prototype toward web-based media control in future versions.

TABLE I
COMPARISON OF RELATED WORK

Work	Technique	Focus	Limitation
MediaPipe Hands [1]	Palm detection and landmark estimation	Real-time hand tracking	Does not directly define audio-control commands
Custom HGR [2]	Landmark embedding and custom classifier	User-defined ges- ture recognition	Requires gesture samples and model setup
MediaPipe + LSTM [3]	Landmark se- quence model- ing	Dynamic gesture recognition	Higher complexity than rule-based mapping
Volume Con- trol System [4]	OpenCV- based gesture processing	System volume control	Limited extension and dashboard discussion
Gesture Media Control [5]	Gesture-to- command mapping	Media and volume control	Less focus on browser- extension readiness
Proposed Work	Landmark- driven rule mapping	Desktop audio control with extension readiness	Current prototype does not train custom gestures

Problem Statement and Objectives

Most desktop audio-control systems depend on physical interaction through keyboards, mouse-based sliders, touch- pads, or hardware volume buttons. Although these methods are reliable, they are not always convenient in situations where hands-free control is required. During presentations, video playback, online meetings, accessibility-focused usage, or distance-based computer operation, physical input devices may interrupt the interaction flow. A touchless gesture-based audio-control system can reduce this dependency by allow- ing the user to control audio behavior through natural hand movements.

The central problem addressed in this work is the design of a lightweight, real-time, camera-based audio interaction system that does not require external sensors, gloves, depth cameras, or additional hardware. The system should be able to detect hand landmarks from a live webcam stream, interpret selected gestures, and convert those gestures into system-level audio commands. At the same time, the system should provide visual feedback so that the user can understand whether the hand is detected, which gesture is active, and how the audio level is changing.

The objectives of the proposed work are as follows:

- To develop a real-time webcam-based audio-control pro- totype using OpenCV and MediaPipe.
- To detect and process hand landmark coordinates for gesture interpretation.
- To map selected hand gestures to audio operations such as volume adjustment, mute/unmute control, and boosted audio response.
- To provide a dashboard interface for displaying gesture status, hand detection state, volume percentage, frame-

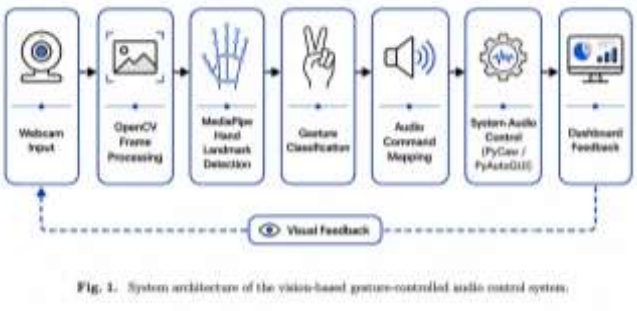

Fig. 1. Proposed architecture of the touchless audio interaction framework.

rate behavior, and interaction history.

- To evaluate the prototype under different lighting and distance conditions.
- To define a browser-extension-ready architecture for ex- tending the system toward web-based media control.

The proposed system is not limited to a conventional volume-control demonstration. It is designed as a modular touchless interaction framework where the present implemen- tation acts as the desktop prototype and future modules can support browser media control, personalized gestures, and dynamic gesture recognition.

Proposed System Architecture

The proposed framework is organized into five major layers: input acquisition, landmark extraction,

gesture interpretation, command execution, and feedback visualization. Each layer performs a specific role in the interaction pipeline, which makes the system modular and easier to extend in future versions.
The first layer is the input acquisition layer, where a built- in webcam captures live video frames. These frames are processed using OpenCV. The second layer is the landmark extraction layer, where MediaPipe detects hand landmarks and provides structured coordinates of finger joints and tips. The third layer is the gesture interpretation layer, where selected landmark relationships are analyzed to identify the active gesture state. The fourth layer is the command execution layer, where recognized gestures are converted into audio-control operations. The fifth layer is the feedback visualization layer, where the dashboard displays the current volume level, gesture name, detection state, frame rate, and volume history.
The architecture follows a lightweight rule-based design for the current prototype. Rule-based interpretation was se- lected because it allows real-time response without requiring a custom training dataset. For example, the distance between selected finger landmarks can be used for volume mapping, while finger-extension states can be used to detect mute or

TABLE II
FUNCTIONAL MODULES OF THE PROPOSED FRAMEWORK

Module	Function
Video acquisition	Captures real-time frames from the webcam using OpenCV.
Landmark extrac- tion	Detects hand key points using MediaPipe.
Gesture interpreta- tion	Classifies gesture states using landmark re- lationships.
Audio control	Converts gesture commands into system- level volume actions.
Dashboard feedback	Displays gesture status, volume level, FPS, and interaction history.
Extension layer	Provides future scope for browser media control and custom gestures.

boost gestures. This keeps the system suitable for low-cost laptops and standard webcams.

A. *Core Interaction Pipeline*

The core interaction pipeline starts when the user places a hand in front of the webcam. The camera stream is captured frame by frame, and each frame is passed to the hand landmark detector. If a hand is detected, the system extracts landmark coordinates and checks predefined gesture conditions. If the gesture matches a valid command, the corresponding audio operation is triggered. If no valid gesture is detected, the system continues monitoring without changing the audio state.

B. *Gesture Command Layer*

The command layer maps hand gestures to system actions. In the implemented prototype, the L-shaped gesture is used for volume adjustment, the open-palm gesture is used for mute/unmute control, and the two-finger gesture is used for boosted audio response. This mapping can be modified in future versions

to support custom gestures, browser media commands, or accessibility-specific controls.

C. *Extension Ready Design*

A key design goal of the proposed framework is extension readiness. The implemented desktop prototype can be con- nected to a browser extension module in future work. In such a configuration, the gesture recognition layer can communicate with browser level controls and trigger media actions such as play, pause, mute, volume increase, volume decrease, and tab level audio management. This allows the system to move from desktop audio control toward web based multimedia interaction.

METHODOLOGY

The methodology of the proposed system follows a real time vision based interaction pipeline. The system captures video frames from a webcam, detects hand landmarks, interprets ges- ture states using landmark relationships, maps the recognized gesture to an audio control command, and displays feedback through a dashboard interface. The complete methodology is designed to remain lightweight so that it can run on a standard laptop without requiring external hardware.

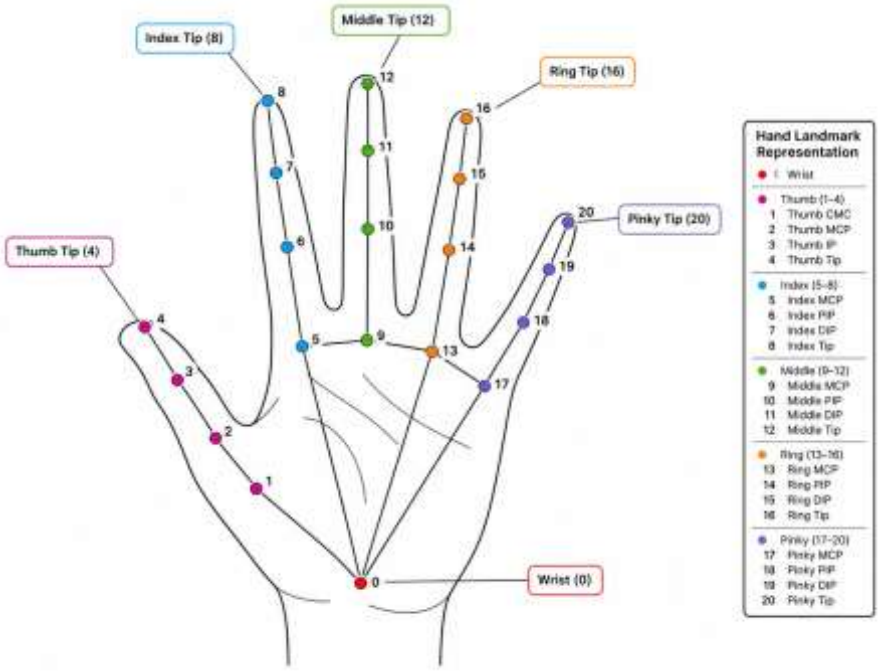

Fig. 2. Hand landmark representation used for gesture interpretation.

A. *Video Frame Acquisition*

The first stage of the system is video frame acquisition. The webcam captures a continuous stream of RGB frames. OpenCV is used to access the camera, resize the frames, flip the frame horizontally for a mirror like interaction experience, and render the graphical dashboard. The frame resolution used in the prototype is 640 × 480 pixels, which provides a balance between real time performance and sufficient hand- region visibility.

B. *Hand Landmark Detection*

After frame acquisition, each frame is processed using MediaPipe Hands. MediaPipe provides structured hand land- marks that represent important hand key points such as finger tips, finger joints, and wrist

position. The landmark-based approach avoids the need for manual skin segmentation or contour extraction and provides a more stable representation for gesture interpretation.

In the implemented prototype, the system tracks a single hand at a time. This design choice reduces computational overhead and avoids ambiguity when multiple hands appear in the camera frame. The landmark detector provides normal- ized coordinates, which are converted into pixel coordinates according to the frame size.

C. *Gesture State Classification*

TABLE III
IMPORTANT HAND LANDMARKS USED IN THE PROTOTYPE

Landmark	Index	Purpose
Wrist	0	Reference point for hand structure
Thumb tip	4	Used for distance-based volume control
Index tip	8	Used with thumb tip for volume mapping
Middle tip	12	Used for two-finger ges- ture detection
Ring tip	16	Used for finger-state checking
Pinky tip	20	Used for open-palm and gesture-state checking

TABLE IV
GESTURE CLASSIFICATION LOGIC

Gesture	Landmark-Based Condi- tion	Action
L gesture	Thumb and index finger are active while other fingers re- main folded	Adjust volume
Open palm	Index, middle, ring, and pinky fingers are extended	Toggle mute/unmute
Two-finger gesture	Index and middle fingers are extended while ring and pinky remain folded	Boost volume response
Invalid gesture	No predefined condition is satisfied	No action

two-finger gesture, where the index and middle fingers are active and the system triggers a boosted audio response.

D. *Distance-Based Volume Mapping*

For continuous volume adjustment, the system calculates the Euclidean distance between the thumb tip and index finger tip. Let (x_1, y_1) represent the pixel coordinate of the thumb tip and (x_2, y_2) represent the

pixel coordinate of the index finger tip. The distance between the two points is calculated using:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

The measured distance is then mapped to a volume range. A smaller distance indicates lower volume, while a larger dis- tance indicates higher volume. The prototype uses minimum and maximum gesture-distance thresholds to avoid unstable values.

$$V = \frac{d - d_{min}}{d_{max} - d_{min}} \times 100 \quad (2)$$

based approach. Instead of training a custom machine learning classifier, the system checks relative positions of selected landmarks to determine whether specific fingers are extended or folded. This keeps the system simple, fast, and suitable for real time execution.

The prototype identifies three primary gesture states. The first is an L-shaped gesture, where the thumb and index finger are active and used for volume adjustment. The second is an open-palm gesture, where multiple fingers are extended and the system toggles mute or unmute behavior. The third is a Here, V represents the estimated volume percentage, d represents the measured landmark distance, d_{min} represents the minimum hand-distance threshold, and d_{max} represents the maximum hand-distance threshold.

E. *Audio Control Execution*

After gesture recognition, the command is passed to the audio-control module. The system modifies desktop audio behavior through Windows-compatible audio-control libraries and automation utilities. To prevent repeated triggering, a

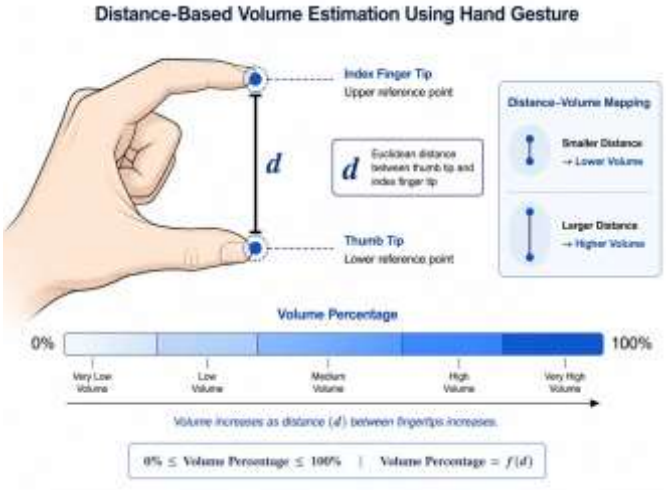

Fig. 3. Distance-based volume estimation using selected finger landmarks.

small cooldown interval is applied between consecutive com- mands. This improves interaction stability and avoids sudden repeated volume changes.

F. *Dashboard Feedback Visualization*

Real-time visual feedback is important for usability. The prototype includes a dashboard interface that shows the de- tected gesture, current volume percentage, hand detection state, frame-rate behavior, and

volume history. This feedback allows the user to immediately verify whether the system has correctly interpreted the gesture.

G. *Overall Algorithm*

The complete workflow of the proposed system is summa- rized as follows:

1. Start webcam and capture real-time video frames.
2. Preprocess each frame using OpenCV.
3. Detect hand landmarks using MediaPipe.
4. Extract selected landmark coordinates.
5. Determine finger states using landmark relationships.
6. Classify the gesture state.
7. If the gesture is valid, map it to an audio-control command.
8. Execute the corresponding volume, mute, or boost op- eration.
9. Display gesture status, volume percentage, FPS, and history on the dashboard.
10. Repeat the process until the application is closed.

IMPLEMENTATION DETAILS

The implemented prototype was developed using Python and follows a modular structure for real-time video processing, gesture interpretation, audio-control execution, and dashboard visualization. The main implementation file contains the com- plete interaction pipeline, including webcam initialization, MediaPipe hand tracking, OpenCV-based interface rendering, gesture-condition checking, and system audio manipulation.

TABLE V
HARDWARE AND SOFTWARE SPECIFICATION

Parameter	Specification
Device	Lenovo IdeaPad 3 laptop
Processor	Intel Core i5 class processor / equiva- lent
RAM	8 GB
Operating System	Windows 11 64-bit
Programming Language	Python
Python Version	Python 3.10
Camera	Built-in HD webcam
Input Resolution	640 × 480 pixels
Dashboard Resolution	1000 × 600 pixels
Main Libraries	OpenCV, MediaPipe, NumPy, PyCaw, PyAutoGUI

OpenCV is used because it provides practical computer vision support for video capture, frame processing, and real-time interface rendering [6].

A. *Development Environment*

The prototype was executed on a standard laptop envi- ronment without requiring external sensors or

specialized hardware. This makes the system suitable for low-cost and accessible deployment. Table V shows the hardware and software configuration used for the pilot implementation.

B. *Software Modules*

The prototype uses OpenCV for webcam access, image- frame handling, drawing operations, and dashboard rendering. MediaPipe is used for hand landmark detection and tracking. NumPy is used for numerical operations such as interpolation and graph plotting. PyCaw and PyAutoGUI are used for system-level audio-control operations on Windows.

The system is designed around five implementation mod- ules:

- **Camera and frame module:** Captures live frames and prepares them for hand tracking.
- **Landmark detection module:** Extracts hand landmark positions using MediaPipe.
- **Gesture logic module:** Checks finger states and classifies the active gesture.
- **Audio execution module:** Maps gestures to volume, mute, and boost operations.
- **Dashboard module:** Displays real-time feedback, FPS, gesture status, and volume history.

C. *MediaPipe Configuration*

The MediaPipe hand tracking configuration is selected to support real-time execution. The prototype tracks one hand at a time to reduce ambiguity and computational overhead. A confidence threshold is used to ensure that only stable hand detections are considered for gesture interpretation.

D. *Gesture-to-Command Mapping*

The implemented prototype supports three main gesture cat- egories. The L-shaped gesture is used for continuous volume

TABLE VI
MEDIAPIPE AND GESTURE PROCESSING PARAMETERS

Parameter	Value
Maximum hands tracked	1
Detection confidence	0.7
Minimum gesture distance	25 pixels
Maximum gesture distance	200 pixels
Volume range	0–100%
Cooldown interval	0.05 s
Volume change threshold	2 units

TABLE VII
GESTURE-TO-COMMAND MAPPING

Gesture	Detection Condition	Command
L gesture	Thumb and index finger active	Volume adjustment
Open palm	Multiple fingers extended	Mute/unmute toggle
Two-finger gesture	Index and middle fingers active	Boosted audio re-sponse
No valid gesture	Condition not satisfied	No action

adjustment. The open-palm gesture is used to toggle mute and unmute states. The two-finger gesture is used for boosted audio response. If no valid gesture is detected, the system does not perform any audio

operation.

E. *Dashboard Interface*

A dashboard-oriented interface was implemented to make the interaction transparent to the user. The dashboard displays the current gesture state, volume percentage, hand detection status, frame rate, and volume history. This design improves usability because the user can instantly verify whether the system has detected the hand and interpreted the gesture correctly.

The dashboard also helps during testing and debugging. If the system fails to detect a gesture, the user can observe whether the issue is caused by hand position, lighting, distance from the webcam, or an invalid gesture posture.

F. *Audio Control Logic*

The volume-control logic is based on distance mapping. When the L-shaped gesture is active, the system measures the thumb-index distance and converts it into a volume percentage. The mapped value is then sent to the audio-control module. To avoid sudden fluctuations, the system applies a threshold-based update strategy so that minor variations in finger position do not repeatedly trigger unstable volume changes.

Mute/unmute and boosted audio response are handled as discrete commands. Since discrete gestures can be triggered repeatedly if the hand remains in the same pose, a cooldown interval is applied to prevent unwanted repeated execution.

Fig. 4. Conceptual dashboard interface for real-time gesture and audio feedback.

Second, it uses rule-based gesture interpretation, which re- duces training requirements and keeps the system lightweight. Third, the modular design makes it possible to extend the same prototype toward browser media control, custom gesture learning, or accessibility-oriented command mapping in future versions.

EXPERIMENTAL EVALUATION

The experimental evaluation was designed to observe the behavior of the proposed prototype under practical desktop usage conditions. Since the system depends on webcam-based hand detection, three major factors were considered: lighting condition, distance between the hand and the camera, and real-time frame-rate behavior. The objective of the evaluation was not only to measure whether a gesture was

detected, but also to understand how stable the system remains in different visual environments.

A. *Evaluation Setup*

The prototype was evaluated using the built-in webcam of a Lenovo IdeaPad 3 laptop. The webcam captured real-time frames at a resolution of 640 × 480 pixels. The user performed predefined gestures in front of the webcam, and each trial was counted as successful when the system correctly detected the gesture and triggered the intended audio-control response.

For the pilot evaluation, six testing conditions were con- sidered: normal lighting, low lighting, bright lighting, 30 cm camera distance, 60 cm camera distance, and 100 cm camera distance. Each condition was tested using 30 gesture trials. The accuracy was calculated using:

Successful Detections

G. *Implementation-Level Advantages*

The implementation has three practical advantages. First, it uses a standard webcam and does not require additional hardware such as gloves, infrared sensors, or depth cameras.

$$Accuracy = \frac{Successful\ \ Detections}{Total\ \ Trials} \times 100 \qquad (3)$$

The values shown in Table VIII summarize the pilot obser- vations recorded under different lighting and camera-distance conditions.

TABLE VIII

PILOT EVALUATION RESULTS UNDER DIFFERENT CONDITIONS

Condition	Trials	Success	Accuracy
Normal lighting	30	28	93.33%
Low lighting	30	23	76.67%
Bright lighting	30	26	86.67%
Distance 30 cm	30	24	80.00%
Distance 60 cm	30	28	93.33%
Distance 100 cm	30	21	70.00%

TABLE IX

LIGHTING-BASED OBSERVATION SUMMARY

Condition	Observation
Normal lighting	Stable landmark detection and smooth volume response.
Low lighting	Reduced hand visibility caused missed or unstable detections.
Bright lighting	Detection remained usable, but glare af- fected consistency in some cases.

TABLE X

DISTANCE-BASED OBSERVATION SUMMARY

Distance	Observation

30 cm	Hand sometimes appeared too close, reducing complete landmark visibility.
60 cm	Balanced distance with stable landmark detection and gesture response.
100 cm	Hand appeared smaller, reducing detection accuracy and gesture stability.

B. *Lighting Based Performance*

The system achieved the best performance under normal lighting conditions. Under normal lighting, hand landmarks were detected more consistently because the hand region was clearly visible to the camera. In low lighting, the success rate decreased because the webcam captured less detailed hand features, which affected landmark stability. Bright lighting pro- duced better results than low lighting, but excessive brightness or glare sometimes affected the clarity of the hand boundary.

C. *Distance Based Performance*

The camera distance also affected gesture recognition. At 60 cm, the system produced the most stable result because the hand was fully visible and landmark positions remained clear. At 30 cm, the hand sometimes occupied too much of the frame, causing partial landmark visibility. At 100 cm, the hand appeared smaller, reducing landmark precision and making gesture classification less reliable.

D. *Frame Rate Analysis*

The average frame rate observed during pilot execution was approximately 26.4 FPS. This frame rate was sufficient for real time audio interaction because the delay between gesture movement and volume response remained acceptable for normal desktop usage. Frame rate behavior was affected by camera quality, background complexity, dashboard rendering, and system load.

TABLE XI

REAL-TIME PERFORMANCE PARAMETERS

Parameter	**Observed Value**
Average FPS	26.4 FPS
Input frame size	640 × 480 pixels
Dashboard size	1000 × 600 pixels
Gesture update type	Real-time frame-based update
Command cooldown	0.05 s

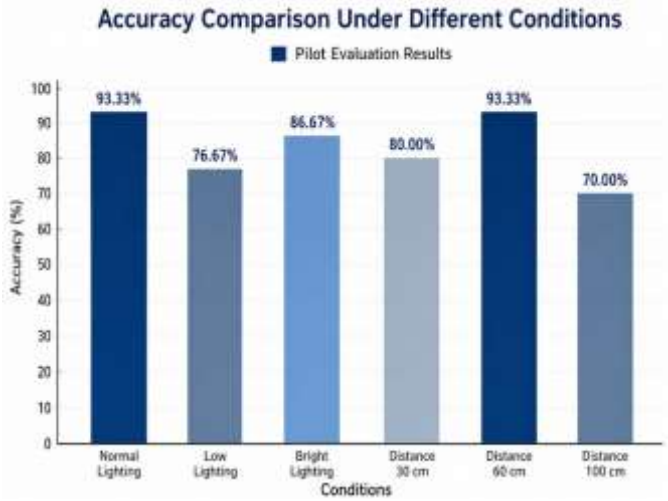

Fig. 5. Accuracy comparison under lighting and distance conditions.

TABLE XII
OBSERVED FAILURE CASES AND POSSIBLE CAUSES

Failure Case	Possible Cause
Gesture not detected	Poor lighting or hand outside camera frame.
Wrong gesture detected	Similar landmark pattern or unusual hand orientation.
Volume fluctuation	Minor finger movement during distance-based mapping.
Delayed response	Low FPS or temporary landmark instability.
Mute repeated quickly	Discrete gesture held for too long without cooldown tuning.

E. *Graphical Result Representation*

To visually compare the pilot evaluation results, the accu- racy values under different lighting and distance conditions can be plotted as a bar graph. This helps identify the most reliable operating condition for the proposed system.

F. *Failure Case Analysis*

The pilot testing also showed that the system may fail or behave inconsistently in certain conditions. Most failures were caused by weak hand visibility, fast hand movement, incomplete finger visibility, or incorrect hand orientation. Since the current prototype uses rule-based gesture logic, it depends strongly on the relative positions of landmarks.

G. *Evaluation Summary*

The evaluation shows that the proposed prototype performs effectively under normal lighting and medium camera distance.

The best pilot result was observed at 60 cm distance and normal lighting, where landmark visibility and gesture stability were high. Performance decreased under low lighting and longer distance because the hand region became less clear to the webcam. These observations indicate that camera place- ment and

lighting quality are important for reliable touchless audio interaction.

DISCUSSION

The proposed prototype demonstrates that a standard web- cam can be used as a low-cost input device for touchless audio interaction. The use of MediaPipe landmarks makes the system more structured than traditional image-processing approaches because gesture decisions are based on landmark relationships rather than raw pixel segmentation. This improves the inter- pretability of the system, as each gesture can be explained through finger positions, landmark distances, and command mapping.

The system is intentionally designed as a lightweight rule- based prototype. This design avoids the need for dataset collection, training, and model deployment during the initial implementation stage. For basic desktop audio-control opera- tions, rule-based landmark mapping is sufficient because the required gestures are limited and easy to define. The L-shaped gesture provides continuous volume adjustment, the open- palm gesture provides mute/unmute control, and the two-finger gesture provides boosted audio response.

However, rule-based gesture interpretation also creates lim- itations. The system depends on stable landmark positions, proper hand orientation, and clear camera visibility. If the hand is rotated, partially outside the frame, or affected by poor lighting, the relative landmark conditions may not match correctly. This means that the current prototype is suitable for controlled desktop environments but requires further improve- ment for highly dynamic or uncontrolled environments.

A major advantage of the proposed design is its modularity. The system separates video acquisition, landmark extraction, gesture interpretation, command execution, and dashboard feedback into logical modules. This makes the prototype easier to extend. For example, the rule-based gesture layer can later be replaced with a trained classifier, while the audio-control layer can be extended toward browser media control, smart- device control, or accessibility-oriented command mapping.

Compared with single-purpose volume-control systems, the proposed framework gives more attention to feedback visual- ization and future integration. The dashboard interface allows the user to observe the current gesture, volume level, frame rate, and detection state in real time. This makes the system more usable during both interaction and testing. The browser- extension-ready architecture further increases the application scope beyond desktop audio control.

BROWSER EXTENSION READINESS

The proposed desktop prototype can be extended into a Chrome extension-based system for browser-level media inter- action. In the current implementation, gesture recognition and

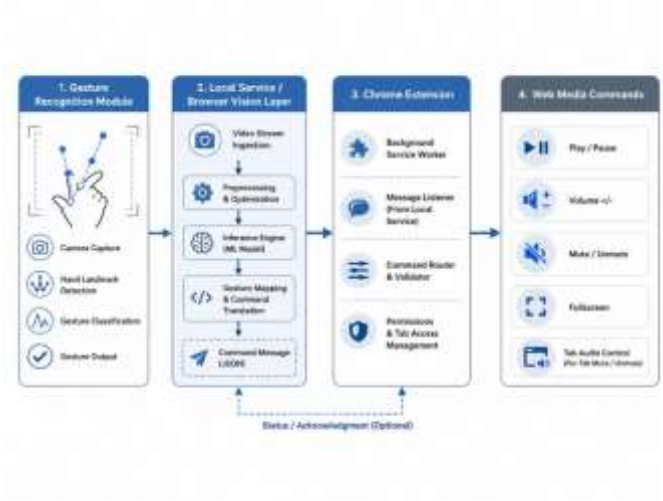

Fig. 6. Proposed browser extension layer for web-based media control.

audio control operate at the desktop level. In a future browser- integrated version, recognized gesture states can be mapped to browser media actions such as play, pause, mute, volume increase, volume decrease, full-screen control, tab-level audio control, and online meeting interaction.

The extension-ready design can follow two possible archi- tectures. In the first architecture, the Python-based gesture- recognition module runs as a local background service. The Chrome extension communicates with this local service and receives recognized gesture commands. These commands are then converted into browser actions using extension APIs or content scripts. This approach allows reuse of the existing Python, OpenCV, and MediaPipe implementation.

In the second architecture, the gesture-recognition pipeline can be moved directly into the browser using JavaScript- based vision libraries. In this case, the browser captures webcam input, performs gesture recognition, and triggers media commands without requiring a separate Python process. This approach may improve portability but would require reimplementation of the current recognition pipeline for the browser environment.

The browser extension concept makes the system more practical for modern multimedia usage. Many users consume audio and video through web platforms such as online lectures, streaming services, video players, and meeting applications. Gesture based browser control can reduce dependency on key- board shortcuts and mouse interaction during such activities. The proposed extension layer is not claimed as part of the current implemented prototype. Instead, it is presented as a future development direction. This distinction is important be- cause the current work focuses on the desktop based touchless audio control prototype, while the extension layer represents
the next stage of scalability.

A. *Possible Browser Level Gesture Commands*

In a browser integrated version, the gesture mapping can be expanded beyond system volume control. Table XIII shows

TABLE XIII
POSSIBLE BROWSER-LEVEL GESTURE COMMANDS

Gesture	Possible Browser Action
Open palm	Play or pause media playback

L gesture	Increase or decrease media volume
Two-finger gesture	Mute or unmute active browser tab
Swipe gesture	Move to next or previous media item
Hold gesture	Activate focus mode or full-screen mode

possible mappings for future browser-level media interaction.

B. *Custom Gesture Expansion*

The current prototype uses predefined rule-based gestures. In future work, custom gesture recognition can be added so that users can define gestures according to their comfort and application needs. This would be useful for accessibility- focused users who may not be able to perform standard gestures consistently. A custom gesture module can collect sample gestures from the webcam, extract landmark features, and train a lightweight classifier for personalized gesture commands.

Dynamic gesture recognition can also be added using temporal models such as LSTM or GRU. Such models can analyze frame sequences instead of single-frame landmark conditions. This would allow the system to recognize motion- based gestures such as swipes, circular movement, double-tap gestures, and hold-based commands. These additions can make the framework more flexible and suitable for advanced human- computer interaction applications.

LIMITATIONS

Although the proposed prototype demonstrates the feasibil- ity of touchless audio interaction, it has certain limitations. The current implementation uses rule-based gesture interpre- tation, which depends on predefined landmark relationships. Therefore, the system may not perform consistently when the hand is rotated, partially visible, too close to the camera, or placed at an unusual angle.

Lighting condition is another important limitation. In low- light environments, the webcam may capture less detailed hand features, which can reduce landmark stability. Similarly, excessive brightness or glare may affect hand visibility and cause inconsistent gesture detection. The system performs best when the hand is clearly visible under normal lighting conditions.

The current prototype tracks one hand at a time. This reduces computational complexity and avoids command am- biguity, but it also limits multi-hand interaction. Multi-hand gestures may be useful for advanced media control, gaming interfaces, or accessibility applications, but they are not sup- ported in the current version.

The system also does not include a trained custom gesture classifier. The gesture logic is based on fixed landmark condi- tions rather than user-specific training. As a result, users with different hand shapes, movement styles, or physical limitations may experience reduced gesture consistency. A personalized gesture learning module would be required to make the system more adaptive.

Browser extension integration is discussed as a future extension-ready architecture, but it is not part of the current implemented prototype. The current work focuses on desktop- level audio control, while browser-level interaction requires additional development using Chrome extension APIs, local service communication, or browser-based computer vision modules.

FUTURE SCOPE

The proposed system can be extended in several directions. The first major improvement is custom gesture learning. Instead of relying only on predefined rule-based gestures, the system can allow users to

record their own gestures and map them to commands. This would make the framework more flexible and useful for accessibility-focused users.

A second improvement is dynamic gesture recognition. The current prototype mainly interprets static landmark relation- ships. Future versions can use temporal models such as Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), or transformer-based sequence models to recognize motion-based gestures. This would allow gestures such as swipes, circular motions, hold gestures, and double-tap actions.

A third direction is browser-level integration through a Chrome extension. The gesture recognition module can be connected to browser media controls so that users can control online lectures, video streaming platforms, browser tabs, and web conferencing tools. Possible browser actions include play, pause, mute, unmute, volume increase, volume decrease, full- screen activation, and next/previous media navigation.

The system can also be extended toward smart home and Internet of Things (IoT) applications. Gesture commands can be mapped to smart speakers, displays, lighting systems, or other connected devices. This would transform the current desktop audio-control prototype into a broader touchless con- trol interface.

Another important future direction is cross-platform deploy- ment. The present implementation is focused on a Windows- based desktop environment. Future versions can support Linux, macOS, Android, or browser-based deployment. A web-based version would improve portability because it would not require users to install Python dependencies manually.

Finally, the dashboard can be improved with calibration support, gesture confidence scores, and user-specific settings. Calibration can help the system adjust to different hand sizes, camera positions, and lighting environments. Confidence- based feedback can also help users understand when a gesture is stable enough to trigger a command.

CONCLUSION

This paper presented a landmark-driven touchless audio interaction framework using OpenCV and MediaPipe. The implemented prototype uses a standard webcam to detect hand landmarks in real time and maps selected gesture states to desktop audio-control operations. The system supports volume adjustment, mute/unmute control, boosted audio response, dashboard-based feedback, and frame-rate monitoring.

The proposed approach demonstrates that a lightweight, low-cost, and camera-based system can be used for touchless audio interaction without requiring external sensors, gloves, or depth cameras. The pilot evaluation indicates that the system performs effectively under normal lighting and medium camera distance, while performance decreases under low-light conditions and longer camera distance due to reduced hand visibility and landmark stability.

Unlike conventional single-purpose volume-control demon- strations, the proposed work frames the system as a modular touchless interaction framework. The architecture separates video acquisition, landmark extraction, gesture interpretation, command execution, and dashboard feedback, making it easier to extend in future versions.

The paper also discussed browser extension readiness as a future development direction. With additional integration, the system can be extended toward browser media control, online meeting interaction, custom gesture learning, dynamic gesture recognition, accessibility support, and smart interface appli- cations. Overall, the proposed prototype provides a practical foundation for developing more advanced

touchless human- computer interaction systems.

CODE AVAILABILITY

The implementation repository for the proposed touchless audio-control prototype is available on GitHub: *Touchless Audio Control Dashboard Repository*.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to Prof. B. D. Patil, Department of Computer Science, Shri Shivaji Vidya Prasarak Sanstha's Bapusaheb Shivajirao Deore College of Engineering, Dhule, for her valuable guidance, support, and encouragement throughout the development of this project.

REFERENCES

1. F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann, "MediaPipe Hands: On-device real-time hand tracking," *arXiv preprint arXiv:2006.10214*, 2020.
2. E. Uboweja, D. Tian, Q. Wang, Y.-C. Kuo, J. Zou, L. Wang, G. Sung, and M. Grundmann, "On-device real-time custom hand gesture recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2023, pp. 4273–4282.
3. S. Biswas, A. Nandy, A. K. Naskar, and R. Saw, "MediaPipe with LSTM architecture for real-time hand gesture recognition," in *Computer Vision and Image Processing*, 2023.
4. N. Arun, Namratha V, A. Dutta, and S. B, "Hand gesture recognition and volume control," *International Journal of Creative Research Thoughts*, vol. 10, no. 6, pp. a430–a434, 2022.
5. R. Tamilkodi, N. Madhuri, G. Dhanushkumar, G. Dileepkumar, G. Rajkumar, and Y. Sandeep, "Hand gesture recognition and volume con- trol," in *Proceedings of the International Conference on Computational Innovations and Emerging Trends*, 2024, pp. 1002–1010.
6. G. Bradski, "The OpenCV library," *Dr. Dobb's Journal of Software Tools*, 2000.
7. A. Kumar and S. Sharma, "A review of hand gesture recognition techniques for human-computer interaction," *International Journal of Computer Applications*, vol. 176, no. 32, pp. 1–6, 2020.
8. M. Oudah, A. Al-Naji, and J. Chahl, "Hand gesture recognition based on computer vision: A review of techniques," *Journal of Imaging*, vol. 6, no. 8, pp. 1–28, 2020.