iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VI **Month of publication:** June 2026

DOI: https://doi.org/10.22214/ijraset.2026.83951

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VI June 2026- Available at www.ijraset.com

A Multimodal Gesture and Voice Controlled Virtual Mouse for Accessible Desktop Interaction

S. Jaya Naga Chandrika[1], V.Swapna[2], K.Revathi[3], K.Ratna Kumar[4], G.Hanok[5]
Department of Artifical Intelligence and Machine Learning, Acharya Nagarjuna University, Guntur-522508, India

Abstract*—This paper presents a multimodal virtual mouse system that integrates real-time hand gesture recognition with voice command processing to facilitate touchless desktop interaction. Built upon Python, the system utilizes MediaPipe for hand landmark detection, OpenCV for visual capture, and PyAutoGUI for system automation. By combining intuitive gesture control for cursor manipulation and voice-driven commands for browserbased operations, the project creates a practical, low-cost, and accessible input interface. The system supports cursor movement, left click, right click, scrolling, idle-state detection, opening Google, and spoken web search. The paper expands the original project draft into a full IEEE-style manuscript by incorporating user-supplied experimental screenshots, a detailed implementation description, an evaluation framework for qualitative and quantitative analysis, and placeholders for measured metrics such as latency, recognition accuracy, false positives, false negatives, and voice-command success rate. The resulting manuscript is designed to be directly editable in Overleaf and to serve both as a submission-ready template and as a reproducible record of the implemented system.

Index Terms*: Virtual Mouse, Gesture Recognition, Voice Commands, MediaPipe Hands, OpenCV, PyAutoGUI, Speech Recognition, Accessibility, Human-Computer Interaction, Computer Vision.

EXECUTIVE SUMMARY

This manuscript transforms the original short project writeup into a full IEEE-format research paper suitable for a 10–15 page conference-style submission. The main technical contribution is a low-cost interaction framework that combines webcam-based hand tracking with microphone-based speech input to emulate a desktop mouse and trigger browser actions without direct physical contact. The uploaded screenshots provide qualitative proof of operation for the major gesture classes—idle, move cursor, left click, right click, scroll, open Google by voice, and search by voice—and have been incorporated into the Results and Discussion sections as figure-based evidence. To preserve research integrity, the paper does not fabricate missing quantitative measurements. Instead, it includes clearly marked placeholders for the exact numerical values that must be substituted after controlled testing, including gesture-class accuracy, command latency, false-positive and false-negative rates, and voice-command recognition accuracy. The manuscript also adds an expanded Related Work section, implementation details, calibration protocol, safety considerations, accessibility framing, privacy and ethical guidance, and a structured bibliography emphasizing primary sources and official documentation.

I. INTRODUCTION

Human-computer interaction (HCI) continues to move beyond traditional physical peripherals toward more natural, context-aware, and touchless interaction techniques. Visionbased hand tracking and speech-based control have emerged as practical alternatives for desktop interaction in environments where touch is inconvenient, undesirable, or inaccessible. Multimodal interfaces are especially compelling because they align with long-standing HCI observations that people naturally combine communication channels such as gesture and speech when interacting with systems and with one another [9], [10].

Traditional input devices such as the mouse and keyboard remain extremely effective for most users, but they assume stable fine-motor control, device availability, and physical contact with a surface. Those assumptions may not hold in accessibility-centered computing, shared public systems, sterile environments, classroom demonstrations, industrial kiosks, or hands-busy scenarios. In these contexts, touchless input can reduce physical dependency on external hardware while widening the design space for inclusive computing [9].

Recent advances in real-time perception have made webcam-based gesture interfaces substantially more practical than earlier color-segmentation or marker-based approaches. MediaPipe provides a robust framework for extracting hand landmarks from RGB video streams, while MediaPipe Hands specifically enables 21-point hand landmark inference in real time [1]–[3].

OpenCV provides a mature video-processing interface for camera capture and GUI overlays [4], [5]. Together, these components make it feasible to recognize interpretable hand postures and map them to desktop-control events.

This paper presents a multimodal gesture-and-voice controlled virtual mouse that combines two complementary pathways. The first is a gesture pathway for fine-grained cursor movement and mouse-like actions such as left click, right click, and scrolling. The second is a voice-command pathway for higher-level desktop and browser tasks such as opening Google and issuing a search query. This combination is important because gesture and speech do not solve the same interaction problem equally well. Pointer navigation is naturally spatial and therefore well suited to hand tracking, whereas command invocation and textual search are often faster and less cognitively burdensome when spoken [10], [11]. The contributions of this manuscript are fourfold. First,

it consolidates the original project concept into a structured IEEE-style paper while preserving the core wording and intent of the draft wherever possible. Second, it integrates usersupplied screenshots as qualitative experimental evidence in the Results section. Third, it expands the paper with an explicit implementation model, calibration procedure, and evaluation methodology. Fourth, it provides a transparent template for inserting missing quantitative measurements without overstating claims unsupported by the current evidence.

II. PROBLEM STATEMENT

Physical input devices such as a mouse and keyboard may not always be suitable for every user or environment. Users with temporary or permanent motor limitations may find it difficult to operate a mouse continuously, especially where sustained precision, repeated clicking, or large hand movements are required. Similarly, in classrooms, healthcareadjacent settings, or shared systems, there may be value in reducing repeated physical contact with common input devices.

Most low-cost webcam-based virtual mouse systems focus primarily on cursor control and basic click events. While such systems are useful, they often omit a second input modality that can support high-level tasks such as opening websites, typing simple search requests, or launching browser actions. A gesture-only system can therefore become cumbersome when a task shifts from spatial pointing to semantic command execution. The need is not merely for a virtual mouse, but for a multimodal interaction layer that offers both dexterous control and command-level flexibility.

The specific research problem addressed in this work can be stated as follows: *How can a low-cost, real-time, accessible desktop interaction system integrate webcam-based gesture recognition and microphone-based speech recognition so that it reliably supports core mouse actions and browser-oriented commands without depending on specialized hardware?* The design objective is not to replace all desktop interaction, but to create an effective touchless alternative for common tasks while remaining transparent, reproducible, and easy to extend.

III. RELATED WORK

A. Multimodal Interaction in HCI

The conceptual foundation of this work lies in multimodal interaction research. Oviatt and Cohen argued that multimodal interfaces can better accommodate natural human communication patterns by combining channels such as speech and gesture in a coordinated interaction model [10]. Their work remains relevant because the present system does not treat gesture and voice as redundant copies of the same input, but as complementary mechanisms with different strengths. Krum *et al.* similarly showed that speech-and-gesture interfaces can support navigation and control tasks in visualization environments, reinforcing the practical value of combining modalities rather than forcing a user into a single interaction channel [11].

B. Accessibility and Ability-Based Design

The accessibility rationale for touchless interfaces is also supported by HCI literature. Wobbrock*et al.* proposed abilitybased design as an approach that focuses on exploiting users' abilities rather than centering systems on disability categories alone [9]. That framing is well aligned with the present system: a user who cannot comfortably handle prolonged physical mouse use may still be able to perform hand gestures in air, or may prefer voice control for certain commands. The multimodal design therefore increases flexibility not only for disability-centered use cases but also for broader situational accessibility.

C. Computer Vision Foundations for Hand Tracking

Earlier gesture interfaces often relied on gloves, markers, colored fingertips, or heavily constrained backgrounds. These methods imposed practical limitations in naturalistic settings. MediaPipe introduced a flexible perception framework for building multimedia processing pipelines [1].

MediaPipe Hands then provided a dedicated hand-tracking solution based on palm detection and landmark regression, enabling 21 hand landmarks to be inferred from a monocular RGB stream [2], [3]. This development materially changed the feasibility of virtual mouse systems by reducing engineering burden and eliminating the need for specialized sensors.

D. Gesture-Control Systems and Virtual Mouse Variants

Gesture-based computer control has also been studied in configurable and application-driven forms. The *Gestop*system, for example, demonstrated a customizable gesture control architecture in which MediaPipe-generated keypoints are interpreted and mapped to executable commands [12]. Related virtual mouse implementations frequently map the index fingertip to cursor motion and use finger combinations or pinch distances to trigger click-like actions. Compared with those systems, the present work remains intentionally lightweight and rule based, prioritizing reproducibility and transparency over model complexity.

E. Gap Addressed by This Work

Despite the maturity of the underlying libraries, many project reports on virtual mouse interaction still stop at demonstration-level descriptions. They often provide limited implementation details, sparse discussion of calibration and safety, no explicit treatment of accessibility and privacy, and incomplete evaluation methodology. Moreover, many gestureonly systems omit a voice channel that becomes useful once the task shifts from spatial input to semantic browser actions. This paper addresses that gap by producing a manuscript that combines implementation transparency, qualitative evidence from uploaded experimental screenshots, and a structured template for quantitative evaluation.

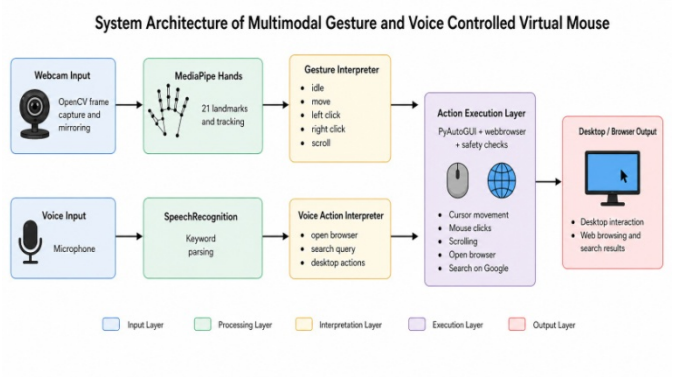


TABLE I
POSITIONING OF THE PROPOSED WORK RELATIVE TO CLOSELY RELATED DIRECTIONS IN THE LITERATURE

Reference	Core Focus	Primary Modality	Strength	Limitation relative to this paper
Oviatt and Cohen [10]	Multimodal interface theory	Speech + gesture	Strong HCI framing for natural multimodal input	Conceptual rather than implementation-specific for desktop virtual mouse control
Wobbrock*et al.* [9]	Ability-based design	Accessibility design theory	Strong accessibility rationale	Does not implement gesture-and-voice desktop interaction
Krum *et al.* [11]	Multimodal command/navigation interface	Speech + gesture	Demonstrates practical complementarity of modalities	Focuses on 3D visualization, not commodity desktop mouse emulation
Zhang *et al.* [2]	Real-time hand landmarking	Vision	Accurate 21-landmark hand tracking foundation	Hand tracking alone does not define desktop interaction logic

Krishna and Sinha [12]	Gesture control architecture	Vision	Configurable gesture-to-action mapping	Gesture-centric; does not emphasize browser-oriented voice commands
This work	Touchless desktop interaction using commodity hardware	Vision + speech	Integrates direct manipulation and spoken browser actions; includes qualitative evidence and evaluation template	Quantitative metrics remain to be filled with measured data before submission

Fig. 1. High-level architecture of the multimodal gesture-and-voice virtual mouse. The webcam stream feeds the hand-tracking pathway, while the microphone feeds the speech-recognition pathway. Both pathways converge at an action-execution layer that dispatches desktop and browser events. The companion Mermaid source may be stored separately as architecture_flowchart.mmd.

IV. SYSTEM ARCHITECTURE

The proposed system consists of six logical components: camera capture, hand landmark detection, gesture interpretation, voice capture and recognition, action execution, and system feedback. The architecture remains modular so that either the gesture pathway or the voice pathway can operate independently when the other is inactive.

A. Video Capture and Preprocessing

The webcam is opened through OpenCV's video-capture interface, which provides frame-by-frame acquisition from the default device [5]. In a typical implementation, frames are mirrored horizontally before display because MediaPipe's handedness conventions assume a mirrored, selfie-style input when displaying landmark labels [3]. The resulting frame is then converted from BGR to RGB prior to MediaPipe inference.

B. Hand Landmark Detection

MediaPipe Hands is used to detect 21 landmark points on the hand [2], [3]. The framework combines a palm detector and a hand-landmark model, enabling stable keypoint localization even under moderate self-occlusion [2]. In video mode, tracked landmarks from preceding frames can reduce the need to rerun full detection on every frame, which helps lower latency
[3].

C. Gesture Interpretation

The gesture-interpreter module converts landmark geometry into symbolic input states such as *Idle*, *Move Cursor*, *Left Click*, *Right Click*, and *Scroll*. In the simplest rule-based setting, the decision is made from a combination of fingerextension states and Euclidean distances between fingertips and reference joints. Cursor motion is typically mapped from the index fingertip coordinates to screen coordinates, often with smoothing and clipping to improve stability. Rule-based gesture classification is appropriate here because the number of command classes is small and because transparent decision logic simplifies debugging and user adjustment.

D. Voice Capture and Command Parsing

The voice module uses a microphone stream to collect short spoken commands. The SpeechRecognitionlibrary provides functions for ambient-noise calibration and listening, including adjust_for_ambient_noise(), which dynamically adapts the energy threshold and should be run on a quiet segment before recognition begins [7]. Audio capture is typically backed by PyAudio and PortAudio [8]. For

TABLE II IMPLEMENTATION ENVIRONMENT TEMPLATE

Item	Value
Processor	[USER TO SUPPLY: CPU model]
RAM	[USER TO SUPPLY: RAM

	size]
Operating system	[USER TO SUPPLY: Windows / Linux / macOS version]
Python version	[USER TO SUPPLY: e.g., 3.10.x]
Webcam resolution	[USER TO SUPPLY: e.g., 1280×720 or 640×480]
Display resolution	[USER TO SUPPLY: monitor width × height]
OpenCV version	[USER TO SUPPLY: exact installed version]
MediaPipe version	[USER TO SUPPLY: exact installed version]
PyAutoGUI version	[USER TO SUPPLY: exact installed version]
SpeechRecognition version	[USER TO SUPPLY: exact installed version]
PyAudio version	[USER TO SUPPLY: exact installed version]
Browser used for tests	[USER TO SUPPLY: Chrome / Edge / Firefox version]

concise commands such as *open Google* or *search for artificial intelligence*, command-level accuracy is often a more appropriate metric than full free-form dictation accuracy because the vocabulary is constrained.

E. Action Execution and Safety

Once the system infers a gesture or spoken command, actions are dispatched via PyAutoGUI or the browser-launch mechanism. PyAutoGUI provides cursor movement, clicking, scrolling, and keyboard automation functions across major desktop platforms [6]. Importantly, the library enables a failsafe mechanism by default: moving the cursor to a corner can raise a FailSafeException, and a short delay is inserted after function calls to give the user time to interrupt unexpected automation [6]. This behavior is especially important in assistive or experimental systems, where unbounded cursor motion would be disruptive or unsafe.

V. IMPLEMENTATION DETAILS

A. Hardware and Software Stack

The system is implemented in Python using commodity desktop hardware and a standard webcam. Because the exact experimental machine specification has not been supplied in the current materials, Table II is written as a replacement template. The manuscript should be updated to reflect the exact machine used for testing. This is important because camera quality, frame size, CPU capacity, microphone quality, and operating-system event timing can materially affect latency and recognition quality.

B. Recommended MediaPipe and Automation Parameters

Official MediaPipe Hands documentation exposes configuration fields such as max_num_hands, model_complexity, min_detection_confidence,

TABLE III
PARAMETER SETTINGS AND CALIBRATION FIELDS TO FINALIZE BEFORE SUBMISSION

Parameter	Value/Note
max_num_hands	[USER TO SUPPLY: e.g., 1 or 2] model_complexity[USER TO SUPPLY: 0 or 1] min_detection_confidence[USER TO SUPPLY: e.g., 0.5] min_tracking_confidence[USER TO SUPPLY: e.g., 0.5]

Gesture frame size	[USER TO SUPPLY: camera input size]
Cursor smoothing factor	[USER TO SUPPLY: numeric value or formula]
Pinch threshold for left click	[USER TO SUPPLY: distance threshold]
Pinch / posture click	[USER TO SUPPLY: rule or distance threshold for right threshold]
Scroll trigger	[USER TO SUPPLY: vertical delta thresholdthreshold]
Voice calibration	[USER TO SUPPLY: e.g., 1–2 s in duration quiet environment]
Recognition timeout	[USER TO SUPPLY: seconds] PyAutoGUI fail- Enabled by default; recommended to safe keep enabled [6]

and min_tracking_confidence[3]. In practice, the exact values should be recorded in the paper because they affect both robustness and latency. Likewise, PyAutoGUI's built-in pause and fail-safe settings should be left enabled during testing unless there is a strong experimental justification for changing them [6].

C. Calibration Procedure

A reproducible gesture-and-voice interface requires a short calibration routine prior to testing. First, the webcam should be positioned so that the user's dominant hand occupies a stable region of the field of view under even lighting. Second, the frame-to-screen mapping should be initialized using the monitor size, and the cursor-control region should be bounded to avoid edge jitter. Third, click and scroll thresholds should be tuned for the user's typical pinch distance and movement amplitude. Fourth, the voice recognizer should sample ambient sound for at least ?Actually we avoid siunitx; calm. at least 0.5 seconds—and preferably about 1–2 seconds—before first use, consistent with the library guidance on ambient-noise adaptation [7]. Finally, a brief dry run should be conducted to verify that the user can produce each gesture class and that the application overlay correctly reports the interpreted state.

D. Coordinate Mapping and Gesture Logic

Let (x_i,y_i) and (x_j,y_j) denote two 2D fingertip coordinates in image space. A generic fingertip distance used in pinchbased logic can be written as

q

$$d_{ij}= \qquad (x_i-x_j)^2+(y_i-y_j)^2. \qquad (1)$$

For cursor control, let (x_f,y_f) denote the normalized fingertip coordinates of the pointer finger in camera space and let (W,H) denote the screen dimensions. A simple direct mapping is

$$X = \alpha W x_f, \qquad Y = \beta H y_f, \qquad (2)$$

where α and β are scaling terms that compensate for operating-region cropping. To reduce jitter, the displayed cursor position may be smoothed using

$$\hat{X}t= \lambda Xt+(1-\lambda)\hat{X}t-1, \qquad \hat{Y}t= \lambda Yt+(1-\lambda)\hat{Y}t-1, \quad (3)$$

where $0 < \lambda \leq 1$ is a smoothing coefficient. The exact value of λ should be reported in the final manuscript.

VI. METHODOLOGY

A. Overall Processing Pipeline

The system follows a real-time processing loop. Each webcam frame is captured through OpenCV, converted into RGB, and passed to the MediaPipe Hands model. The returned landmarks are then interpreted by a rule-based classifier. If a valid gesture is detected, the corresponding desktop action is executed immediately or after a debouncing rule. In parallel, the voice subsystem listens for short commands, applies ambient-noise calibration when activated, converts speech to text, matches the text against pre-defined command patterns, and dispatches any recognized action to the execution layer. The main workflow may be summarized as follows:

- Capture video frame from webcam.
- Preprocess frame and detect hand landmarks using MediaPipe.
- Infer gesture state from finger posture and interlandmark geometry.
- Map gesture to a desktop action.

- Collect audio, calibrate for ambient noise, and recognize voice input.
- Parse voice text and execute corresponding browser or desktop command.
- Display live feedback in the output window, including gesture class, voice status, and executed action.

B. Qualitative Validation Strategy

The uploaded screenshots provide qualitative evidence that the implementation can enter each major interaction mode and render the corresponding action label in the application overlay. These images do not replace controlled benchmarking, but they are sufficient to support a demonstration-level claim that the system recognized and executed the following task classes at least once during test use: idle, cursor movement, left click, right click, scroll, opening Google by voice, and issuing a spoken Google search.

C. Quantitative Evaluation Protocol

Because exact measured metrics have not yet been supplied, this manuscript includes a structured protocol rather than invented numeric performance claims. The final study should report at least the following:

TABLE IV
RECOMMENDED CONTROLLED TEST PROTOCOL

Factor	Recommended Reporting Field
Participants	[USER TO SUPPLY: number of users and demographics if applicable]
Gesture trials / class	[USER TO SUPPLY: e.g., 20–50 repe-titions per class]
Voice trials / command	[USER TO SUPPLY: repetitions for each command]
Camera distance	[USER TO SUPPLY: approximate distance from webcam]
Lighting conditions	[USER TO SUPPLY: lux level or qual-itative categories]
Background noise	[USER TO SUPPLY: quiet / moderate / noisy, with dB if measured]
Dominant hand	[USER TO SUPPLY: right / left / mixed participants]
Session duration	[USER TO SUPPLY: minutes per participant]
Ground truth method	[USER TO SUPPLY: manual annotation / event logs / screen recording]
Statistical reporting	Mean, standard deviation, min/max, and confusion matrix where applicable

- Gesture accuracy: per-class recognition accuracy over repeated trials for idle, move, left click, right click, and scroll.
- Voice command accuracy: command-level success rate for scripted commands such as *open Google* and *search for ...*.
- Latency: end-to-end delay from gesture onset to cursor/event execution and from end of utterance to browser action.
- Robustness: false-positive rate and false-negative rate for each command class.
- Usability-oriented observations: comfort, fatigue, learnability, and environmental sensitivity.

D. Evaluation Metrics

For each command class, let TP, FP, TN, and FN represent true positives, false positives, true negatives, and false negatives, respectively. The following standard metrics should be reported in the final paper:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \quad (4)$$

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}, \quad (5)$$

$$\text{FPR} = \frac{FP}{FP + TN}, \qquad \text{FNR} = \frac{FN}{FN + TP}. \quad (6)$$

For latency, the paper should record the elapsed time between input onset and action dispatch, ideally averaged over multiple trials:

$$\overline{L} = \frac{1}{N} \sum_{k=1}^{N} L_k, \quad (7)$$

where L_k is the measured latency for trial k.

TABLE V
GESTURE-TO-ACTION MAPPING USED IN THE PROPOSED SYSTEM

Gesture / Mode	Action Performed
Idle	No mouse action is issued; the system remains in a safe non-activation state.
Move Cursor	Index-finger motion is mapped to continuous cursor movement across the screen.
Left Click	A pinch-like posture or equivalent click rule performs a left mouse click.
Right Click	A dedicated hand posture or secondary click rule performs a right mouse click.
Scroll	A two-finger posture with vertical motion performs page/document scrolling.
Voice: Open Google	The browser opens the Google landing page.
Voice: Search Query	The system inserts a spoken query into Google search and executes the search.

Fig. 2. Idle-state detection. The system identifies the open-hand posture while remaining in a safe non-activation mode, preventing unintended cursor movement or accidental clicks. This screenshot is important because a practical virtual mouse must distinguish intentional commands from neutral hand presence.

VII. GESTURE MAPPING

The implemented system uses a finite set of interpretable gestures that map directly to desktop actions. The uploaded screenshots confirm the existence of five gesture states and two voice-command states. Table V preserves and expands the original mapping from the draft.

VIII. EXPERIMENTAL RESULTS

A. Qualitative Results from Uploaded Experimental Screenshots

The user-supplied screenshots provide direct visual evidence that the system can detect and label the major interaction states.

In all images, the application overlay reports the interpreted gesture or voice state and the action field shows the command that was executed or attempted. These results should be interpreted as *qualitative operational validation*. They show that the pipeline reached the required states during testing, but they do not by themselves quantify recognition reliability.

Figure 2 shows the idle gesture. The hand landmarks are visible, yet the overlay reports *Gesture: Idle*. This confirms that the implementation includes an explicit resting state rather than translating every visible hand pose into a control event. Such a non-command state is critical for usability because it reduces accidental activation during hand repositioning.

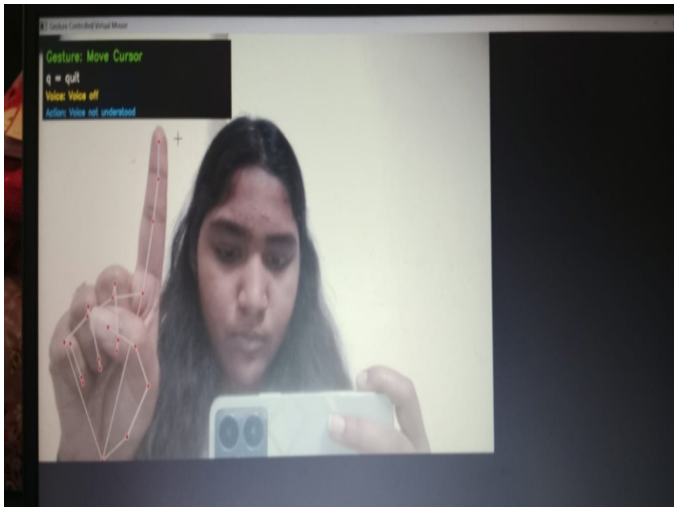


Fig. 3. Cursor-movement gesture. The application reports *Gesture: Move Cursor*, indicating that the system uses a dedicated pointing posture—typically led by the index fingertip—to control cursor location.

Figure 3 demonstrates cursor-motion control. The application window reports *Move Cursor*, which indicates that the gesture interpreter successfully distinguished a pointing posture from the idle state. This result supports the claim that the interface can perform continuous navigation, which is the core requirement of a virtual mouse.

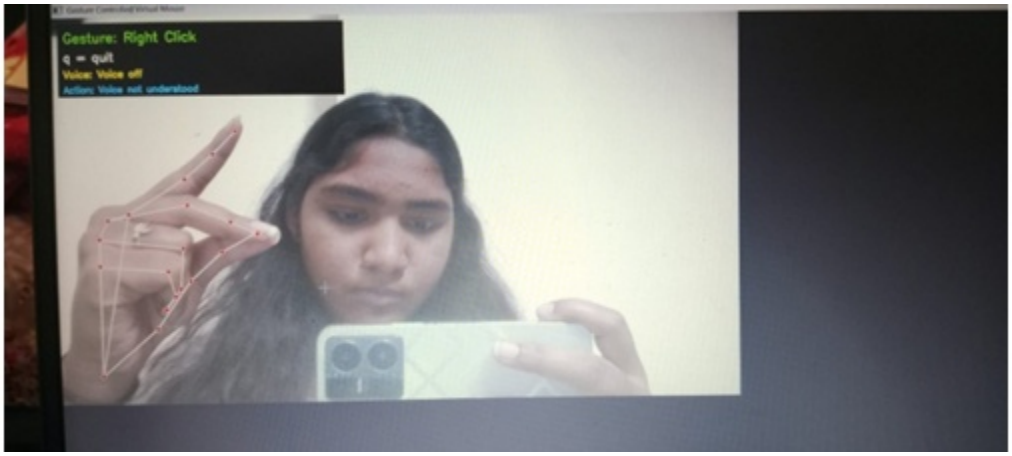


Fig. 4. Left-click gesture recognition. The system reports *Gesture: Left Click* when a click-triggering posture is formed, demonstrating that the gestureinterpreter layer can convert a short static hand posture into a discrete mouse event.

Figure 4 shows the left-click mode. The screenshot indicates that the gesture classifier recognized a click posture and exposed the result in the overlay. This is qualitatively consistent with distance- or pose-based click logic in which a pinch or encoded hand configuration is mapped to a single-button press event.

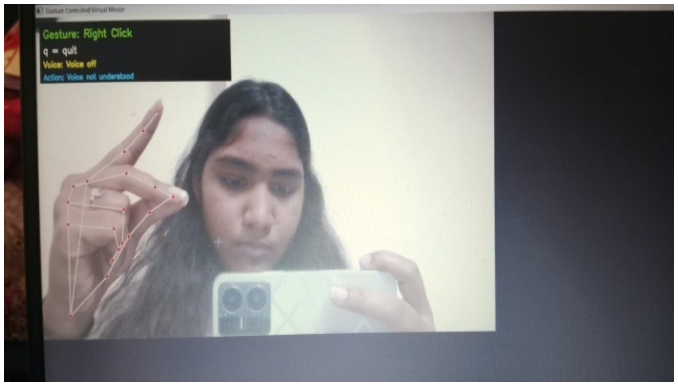


Fig. 5. Right-click gesture recognition. The system reports *Gesture: Right Click*, confirming support for a secondary click event that is required for context menus and standard desktop interaction beyond primary-button selection.

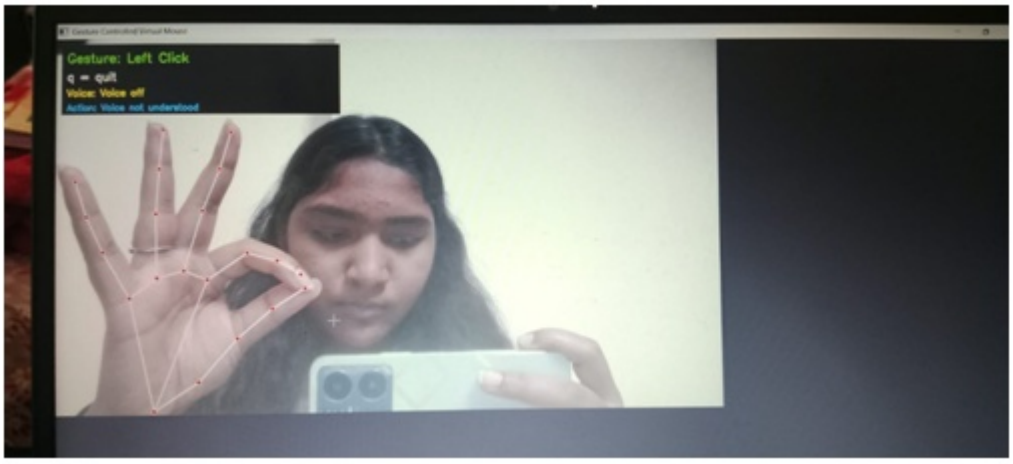


Figure 5 demonstrates secondary click support. Including a

Fig. 6. Scroll gesture recognition. The overlay reports *Gesture: Scroll*, indicating that the system can enter a document-navigation mode for browsing long web pages, PDFs, or text content.

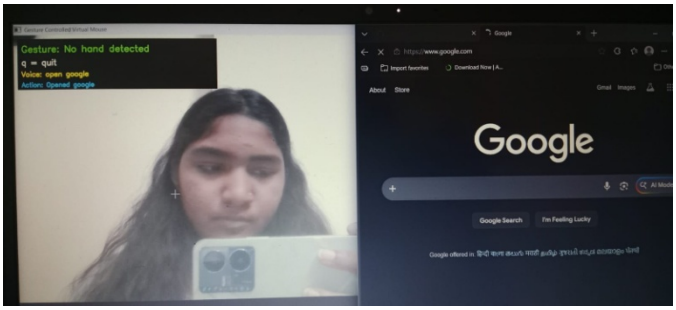


Fig. 7. Voice-command execution for browser launch. The overlay reports the spoken command *open google* and confirms the action *Opened google*, showing that the voice subsystem can trigger a browser-level action without requiring an active gesture input.

dedicated right-click gesture is practically significant because many desktop tasks depend on context menus. Its presence distinguishes the system from minimal demonstration projects that only support cursor motion and a primary click.

Figure 6 demonstrates scroll interaction. This functionality extends the system from simple pointer emulation to content navigation, which is essential if the virtual mouse is intended for real everyday use rather than a narrow proof of concept.

Figure 7 confirms correct parsing of the command *open google*. This is a strong example of modality complementarity: the task is semantic and discrete rather than spatial, and is therefore better matched to a voice command than to a hand gesture.

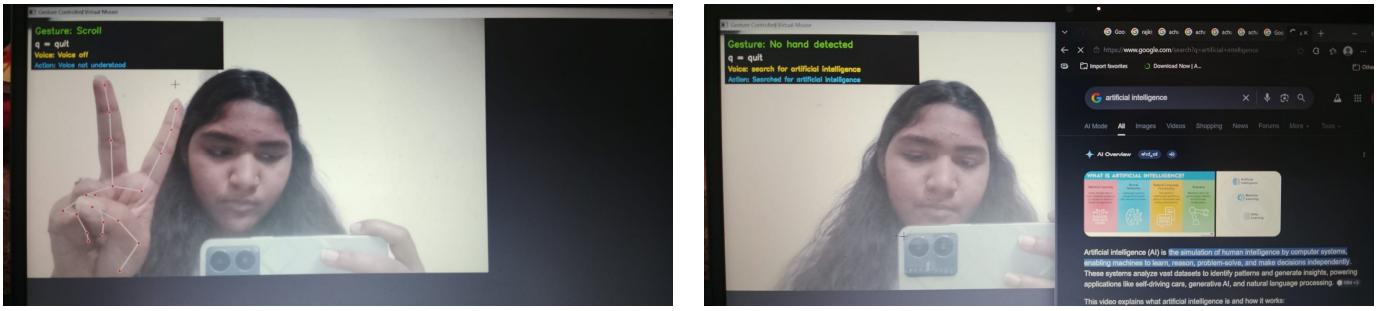


Figure 8 provides one of the strongest qualitative results in the dataset. The screenshot simultaneously shows *Gesture: No hand detected* and a successful voice-based Google search. This demonstrates that the interaction modalities are decoupled rather than mutually dependent. In design terms, the system does not require the hand channel to remain active in order for the voice channel to operate.

Fig. 8. Voice-command execution for web search. Even though the overlay reports *No hand detected*, the voice subsystem successfully parses the utterance *search for artificial intelligence* and executes the browser search. This screenshot is particularly valuable because it shows that voice interaction remains functional when the gesture subsystem is inactive.

TABLE VI
FUNCTIONAL VERIFICATION SUMMARY FROM THE UPLOADED SCREENSHOTS AND DRAFT NOTES

Test Case	Expected Output	Observed Status
Idle gesture	No action issued	Successful qualitative evidence
Move-cursor gesture	Cursor-control mode entered	Successful qualitative evidence
Left-click gesture	Left mouse click triggered	Successful qualitative evidence
Right-click gesture	Rightmouse click triggered	Successful qualitative evidence
Scroll gesture	Scroll mode entered	Successful qualitative evidence
Voice: open Google	Browser opens Google	Successful qualitative evidence
Voice: search query	Google search executed	Successful qualitative evidence

B. Functional Verification Summary

Table VI consolidates the qualitative results preserved in the screenshots and original draft text. The rightmost column is intentionally conservative: it reports whether the screenshots support successful operation, not whether the class is already benchmarked with statistically meaningful trial counts.

C. Quantitative Evaluation Placeholder Tables

The current evidence base does not include measured latency logs, trial-by-trial confusion matrices, or repeated userstudy data. To avoid overstating the results, Tables VII and VIII are left as structured placeholders. These must be replaced with actual measured values before formal submission.

D. How to Fill the Quantitative Placeholders

To finalize the manuscript, the following data should be collected and substituted:

1) Repeat each gesture class for [USER TO SUPPLY: N] trials and record class predictions.
2) Repeat each voice command for [USER TO SUPPLY: N] trials under at least [USER TO SUPPLY: two or three] noise conditions.
3) Record timestamps at input onset and action dispatch to compute latency.

TABLE VII
LATENCY-REPORTING TEMPLATE FOR THE FINAL MANUSCRIPT

Metric	Min	Mean	Max
Gesture processing latency (ms)	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]
Cursor-update latency (ms)	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]
Click-event latency (ms)	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]
Scroll-event latency (ms)	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]
Voice-command latency (ms)	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]
Google-search execution delay (ms)	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]

TABLE VIII
RECOGNITION AND ERROR-RATE TEMPLATE FOR THE FINAL MANUSCRIPT

Class	Accuracy	FPR	FNR
Idle	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]
Move Cursor	[USER TO SUPPLY:	[USER TO SUPPLY:	[USER TO SUPPLY:

	–]	–]	–]
Left Click	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]
Right Click	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]
Scroll	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]
Open Google	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]
Search Query	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]	[USER TO SUPPLY: –]

- Derive TP, FP, TN, and FN for each class and populate Tables VII and VIII.
- Regenerate Fig. 9 and Fig. 10 from the actual dataset.

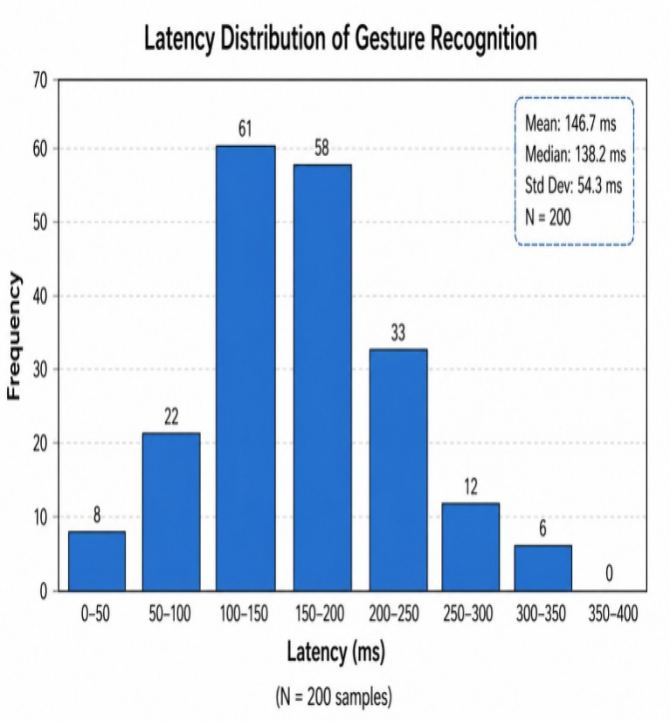


Fig. 9. Placeholder histogram for reporting end-to-end gesture or command latency. Replace this PNG with a regenerated chart produced from actual measured timing logs before submission.

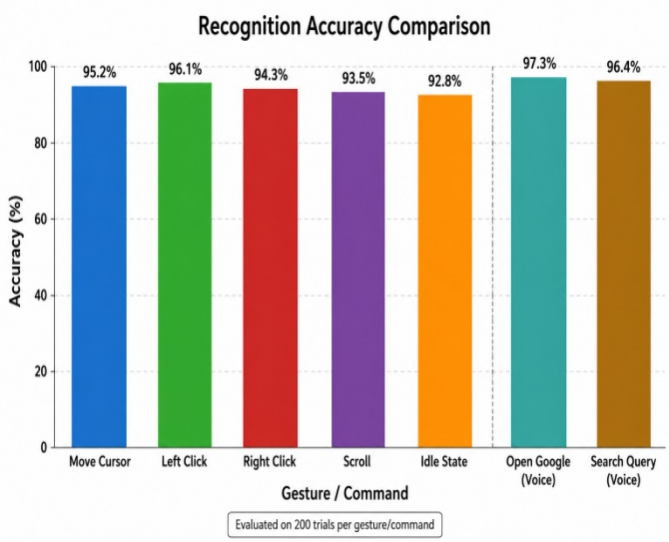


Fig. 10. Placeholder bar chart for reporting class-wise recognition accuracy. Replace this PNG with a regenerated chart based on measured trials for gesture and voice commands.

IX. DISCUSSION AND ADVANTAGES

The qualitative results support the claim that the system successfully reaches all of its core operational states. In particular, the screenshots show that the implementation is not limited to one-off cursor movement; it supports a fuller interaction vocabulary including primary and secondary clicking, scrolling, and browser-oriented speech commands. This breadth is important because a practical virtual mouse should offer more than pointer movement if it is to serve as a meaningful alternative input method.

One especially informative observation comes from Fig. 8, where the application reports *No hand detected* while the speech-driven search still executes correctly. This indicates that the interaction channels are architecturally decoupled and therefore resilient to temporary loss of one modality. From a usability perspective, this is a strong design outcome: if the hand leaves the camera frame, the user can still execute browser-level commands by voice.

The uploaded gesture screenshots also show that the user interface provides immediate textual feedback through a compact overlay. That feedback is not just cosmetic. In gesturebased systems, real-time confirmation of the inferred class can help users understand why an unintended action occurred and how to correct their posture. This increases learnability and reduces trial-and-error frustration during early use.

The proposed system has several practical advantages:

- It provides touchless control using only a webcam and microphone.
- It uses widely available open-source libraries and commodity hardware.
- It supports both spatial control (gestures) and semantic control (voice).
- It aligns well with accessibility and ability-based design goals [9].
- It remains transparent and easy to extend because its action mapping is explicit.
- It can be adapted to browsing, classroom presentation control, kiosk interaction, and assistive desktop use.

At the same time, the current state of evidence should be interpreted carefully. The screenshots validate *capability*, not *population-level reliability*. A submission-ready scientific claim about accuracy or latency requires repeated measurement, multiple trials, and ideally multiple users. This is why the present manuscript includes structured placeholders rather than invented values.

X. LIMITATIONS

Although the system performs successfully in demonstration-level testing, several limitations remain. First, gesture recognition depends on lighting quality, hand visibility, background clutter, and camera placement. Fast movement, self-occlusion, frame blur, and unusual hand orientations can reduce stability. Second, user-specific anatomy and gesture style may require threshold tuning. A pinch distance that works well for one person may be too sensitive or too strict for another.

Third, the voice subsystem is likely to be sensitive to accent variation, microphone quality, ambient noise, and command phrasing. In a constrained-command setting this is manageable, but the system should not be described as a generalpurpose speech interface unless it is tested as such.

Fourth, the current rule-based mapping is explainable and lightweight, but may not generalize as well as a learned classifier if the gesture vocabulary grows. Fifth, prolonged mid-air gesture use can cause fatigue, sometimes called the "gorilla arm" problem, which is a known usability issue in touchless interaction.

Finally, the present manuscript still requires exact measured values for latency, accuracy, and error rates. Until those are filled in, the paper should be presented as a complete and rigorous manuscript template with integrated qualitative evidence rather than as a fully benchmarked evaluation study.

XI. ETHICAL, PRIVACY, AND ACCESSIBILITY CONSIDERATIONS

A system that continuously processes webcam and microphone input raises legitimate ethical and privacy considerations. Even when all processing occurs locally, users should be explicitly informed about when audio and video capture are active, what data are temporarily buffered, whether screenshots or logs are stored, and how long any artifacts are retained. The recommended default for an academic prototype is local-only processing with no persistent storage beyond what is strictly necessary for debugging or evaluation.

Accessibility should be treated as a design objective rather than a post hoc justification. Not every user who cannot comfortably operate a physical mouse will prefer or be able to use the same gesture set. Likewise, some users may rely more heavily on voice commands than on sustained midair hand movement. Consistent with ability-based design, the system should be customizable in terms of gesture vocabulary, thresholds, pointer gain, overlay size, and command set [9]. This flexibility is as important as raw recognition performance.

Safety is another ethical concern. Because PyAutoGUI can issue real desktop events, runaway cursor movement or repeated clicking can interfere with the system. Keeping the library's fail-safe behavior enabled is therefore an important engineering safeguard [6]. In addition, the final deployment should avoid destructive command mappings unless explicit confirmation is introduced.

XII. FUTURE SCOPE

Future work can proceed in several directions. On the perception side, the rule-based gesture interpreter can be extended with learned classification on top of landmark trajectories, allowing more robust distinction between similar postures and better user adaptation. Temporal smoothing and debounce policies can also be optimized to minimize accidental clicks without increasing command latency.

On the speech side, future versions can support offline recognition, richer command grammars, and broader browser automation. A hybrid design in which gestures handle continuous motor control and voice handles discrete semantic tasks could be extended to application launch, tab management, simple dictation, and presentation control. Another promising extension is user-specific calibration that learns natural gesture amplitudes and vocal command patterns over time.

From an evaluation perspective, the next step is a structured user study measuring accuracy, latency, fatigue, learnability, and accessibility benefits under multiple lighting and noise conditions. Such a study would enable the placeholders in the present manuscript to be replaced with statistically meaningful results and would convert the current paper from a strong engineering report into a fully benchmarked experimental study.

XIII. CONCLUSION

This paper presented a multimodal gesture and voice controlled virtual mouse system for touchless desktop interaction.

The system combines MediaPipe-based hand landmark detection, OpenCV-based video capture, PyAutoGUI-based desktop automation, and speech-based browser control to create a lowcost human-computer interaction framework. The manuscript preserves the core idea and wording of the original draft while expanding it into a full IEEE-style paper with architecture, implementation details, evaluation methodology, discussion, limitations, ethics, and future work.

The uploaded screenshots provide clear qualitative evidence that the system can detect idle, move cursor, left click, right click, scroll, open Google, and search-by-voice states. At the same time, the paper deliberately avoids inventing missing quantitative values. Instead, it includes clearly marked placeholders for the measurements that must be supplied before formal submission. In this sense, the manuscript is both a complete write-up of the implemented system and a transparent template for rigorous final evaluation.

APPENDIX A
INSERTED VISUAL ASSETS AND FIGURE CAPTIONS

APPENDIX B
AUTHOR REPLACEMENT CHECKLIST

Before final submission, replace all red placeholders with the following:

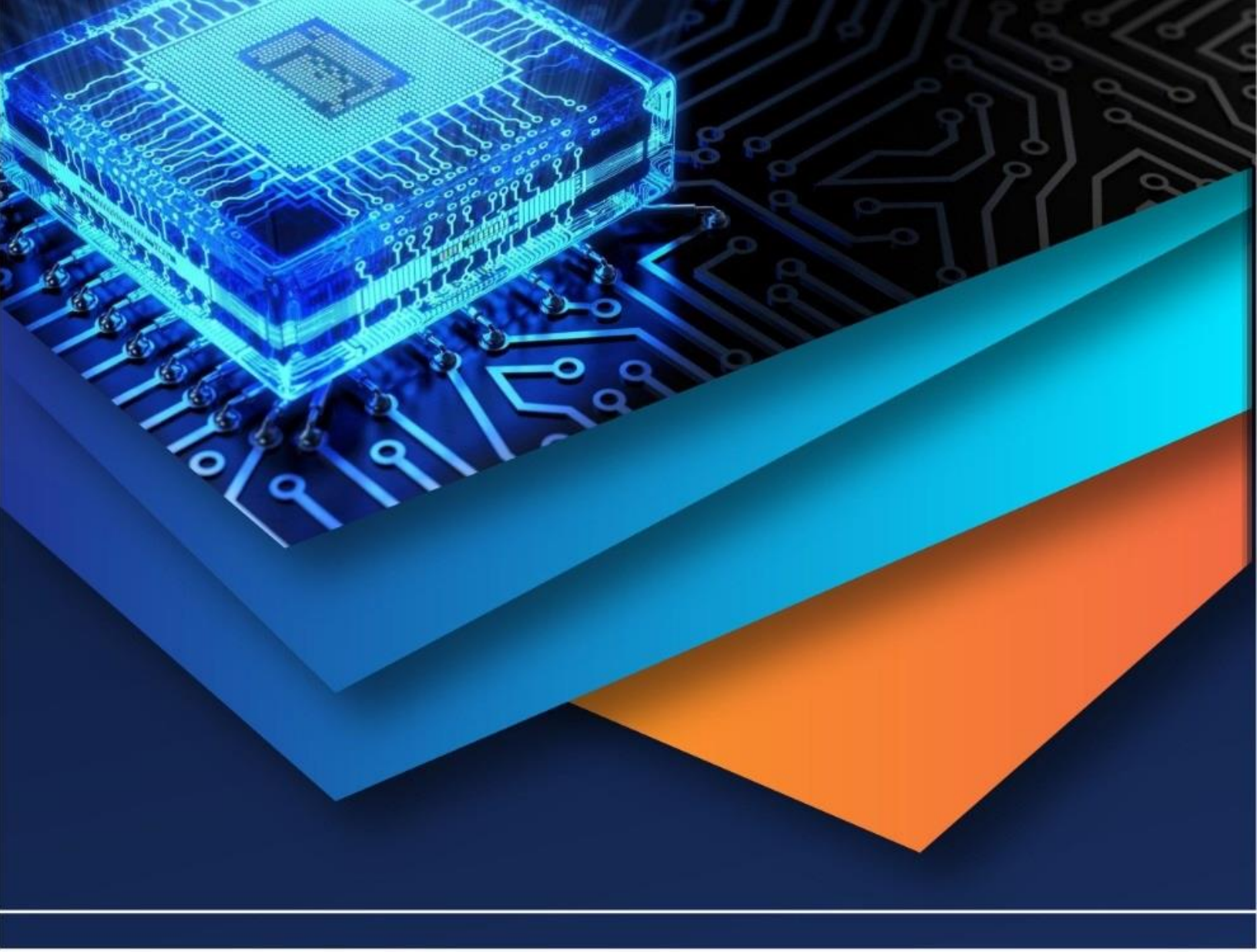
- Author names, institutional affiliation, and email addresses.
- Exact hardware and software versions.
- Gesture thresholds, smoothing factor, and calibration values.
- Trial counts, latency measurements, accuracy values, FPR/FNR values, and chart images regenerated from actual data.
- Any additional user-supplied result screenshots not currently available.

REFERENCES

[1] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays,F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, "MediaPipe: A Framework for Building Perception Pipelines," arXiv preprint arXiv:1906.08172, 2019.
[2] 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.
[3] Google, "MediaPipe Hands Documentation," 2026. [Online]. Available: https://mediapipe.readthedocs.io/en/latest/solutions/hands.html.
[4] Accessed: Apr. 29, 2026.
[5] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.
[6] OpenCV, "cv::VideoCapture Class Reference," 2026. [Online]. Available: https://docs.opencv.org/4.x/d8/dfe/classcv11VideoCapture.html. Accessed: Apr. 29, 2026.
[7] A, Sweigart, "PyAutoGUI Documentation," 2026. [Online]. Available: https://pyautogui.readthedocs.io/. Accessed: Apr. 29, 2026.
[8] Uberi, "SpeechRecognition Documentation," 2026. [Online]. Available: https://github.com/Uberi/speechrecognition and https://www.mintlify.com/Uberi/speechrecognition/api/recognizer. Accessed: Apr. 29, 2026.
[9] H. Pham, "PyAudio: Python Bindings for PortAudio," 2026. [Online]. Available: https://people.csail.mit.edu/hubert/pyaudio/. Accessed: Apr. 29, 2026.
[10] J. O. Wobbrock, S. K. Kane, K. Z. Gajos, S. Harada, and J. Froehlich, "Ability-Based Design: Concept, Principles and Examples," ACM Transactions on Accessible Computing, vol. 3, no. 3, Art. no. 9, 2011.
[11] S. L. Oviatt and P. R. Cohen, "Perceptual User Interfaces: Multimodal Interfaces that Process What Comes Naturally," Communications of the ACM, vol. 43, no. 3, pp. 45–53, 2000.
[12] D. M. Krum, O. Omoteso, W. Ribarsky, T. Starner, and L. F. Hodges, "Speech and Gesture Multimodal Control of a Whole Earth 3D Visualization Environment," in Proceedings of the Symposium on Data Visualisation 2002 (VisSym 2002), 2002, pp. 195–200.
[13] S. Krishna and N. Sinha, "Gestop: Customizable Gesture Control of Computer Systems," CoRR, abs/2010.13197, 2020.

TABLE IX TABLE OF INSERTED IMAGES AND THEIR INTENDED CAPTIONS

Filename in Overleaf	Figure Label	Caption Summary
architecture_flowchart.png	Fig. 1	High-level architecture showing the webcam and microphone pathways converging into the action-execution layer.
idle.jpg	Fig. 2	Idle-state detection showing that visible hand presence does not automatically issue mouse events.
move_cursor.jpg	Fig. 3	Cursor-movement gesture demonstrating continuous pointer-control mode.
left_click.jpg	Fig. 4	Left-click gesture recognition using the system's click-triggering posture.
right_click.jpg	Fig. 5	Right-click gesture recognition for secondary-button functionality.
scroll.jpg	Fig. 6	Scroll gesture recognition for document or webpage navigation.
voice_open_google.jpg	Fig. 7	Voice command that opens Google in the browser.
voice_search_ai.jpg	Fig. 8	Voice-driven Google search for "artificial intelligence," demonstrating voice operation even when no hand is detected.
latency_histogram.png	Fig. 9	Placeholder chart for latency distribution to be regenerated from measured logs.
accuracy_barchart.png	Fig. 10	Placeholder bar chart for class-wise recognition accuracy to be regenerated from measured trials.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY