Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

Smart Gesture Interface for Medical and Accessibility Applications: An AI-Based Real-Time Virtual Mouse with Concurrent Voice Control

M. Krishnaveni,[1] B. Sai Venkata Ganesh,[1] K. Bhuvan,[1] CH. Mutyalu,[1] and Arya Abhay Ubhale[1]

[1]Department of Computer Science and Engineering (Artificial Intelligence & Machine Learning),
Anil Neerukonda Institute of Technology and Sciences (ANITS), Visakhapatnam – 531162, India

***Abstract*—Human-Computer Interaction (HCI) is undergoing a paradigm shift driven by rapid advances in computer vision and machine learning. This paper presents an AI-based, real-time, gesture- and voice-controlled virtual mouse system that eliminates the requirement for physical input devices. The proposed system leverages the MediaPipe Hands framework for 21-landmark hand tracking and OpenCV for video acquisition and pre-processing. Twenty-one hand landmarks are monitored per frame to recognise gestures corresponding to eight standard mouse operations: cursor movement, left click, right click, double click, drag-and-drop, scroll, volume control, and screenshot capture. Cursor control employs an exponential moving average (EMA) smoothing filter to suppress high-frequency jitter. Click and drag-and-drop detection are driven by pinch-distance thresholding combined with state-machine logic, while a wrist-trajectory trigger enables screenshot capture. Voice commands are processed by a multi-threaded Google Speech API module running concurrently with the gesture loop, supporting browser automation, desktop folder navigation, system media control, and dynamic gesture enable/disable. The system requires only a standard webcam and microphone with no specialised hardware or offline training data. Experimental evaluation demonstrates 93.4% average gesture recognition accuracy, 91.3% voice command accuracy, and end-to-end gesture latency below 55 ms at 28–32 FPS on a CPU-only system. The system offers significant potential for accessibility applications, touchless interfaces in sterile healthcare environments, and hands-free industrial HCI.**

***Index Terms*—Human-Computer Interaction, MediaPipe Hands, gesture recognition, virtual mouse, voice control, OpenCV, touchless interface, exponential smoothing, accessibility, healthcare HCI**

I. Introduction

The increasing integration of computing devices into professional, medical, and industrial environments has intensified demand for intuitive, touchless interaction paradigms. Traditional input devices such as keyboards and mice impose physical barriers on individuals with motor disabilities, repetitive strain injuries, or those operating in environments where physical contact is undesirable, including sterile surgical theatres, cleanrooms, and remote control facilities. The need for touchless alternatives in clinical settings is well-established: Cronin and Doherty [1] conducted a comprehensive review of touchless computer interfaces deployed in hospitals, identifying hygiene maintenance and workflow integration as the primary design imperatives; Alvarez-Lopez et al. [2] systematically reviewed commercial off-the-shelf (COTS) devices for intraoperative gesture detection, concluding that detection accuracy was generally adequate but ergonomic and adoption barriers remained; and Bockhacker et al. [3] evaluated a gesture-controlled touchless image viewer in a live operating room, confirming viability but highlighting the need for improved interaction feedback. More recently, Zargham et al. [4] demonstrated combined gesture and speech control for surgical lighting systems, while Wu et al. [5] applied participatory design to define intuitive gesture vocabularies for sterile operating theatres.

Hand gesture recognition using a standard webcam is a compelling solution to these challenges. The introduction of Google's MediaPipe Hands framework [6] democratised real-time skeletal keypoint tracking, delivering 21 three-dimensional hand landmarks at inference rates exceeding 30 frames per second on CPU-only systems. Earlier gesture-based virtual mouse implementations — such as the webcam-based system by Hassan Shibly et al. [7] and the OpenCV-and-Python approach by Rachana and Khan [8] — established the feasibility of hardware-free cursor control, yet remained constrained in gesture vocabulary, robustness under variable lighting, or breadth of supported functionality.

This paper proposes an integrated, hardware-free, AI-based virtual mouse combining MediaPipe Hands for 21-landmark detection, OpenCV for real-time video acquisition and processing, and PyAutoGUI for cross-platform mouse and keyboard event synthesis, augmented by a multi-threaded Google Speech Recognition module for parallel voice command processing. The system supports eight mouse gesture operations and fourteen voice commands. An EMA filter suppresses cursor jitter; a pinch-state machine disambiguates single click, double click, and drag-and-drop intents; and a wrist-trajectory trigger provides a novel screenshot modality. The complete system executes on any device equipped with a standard webcam, microphone, and Python 3.8+, with no GPU acceleration or offline training required, making it broadly accessible for healthcare, accessibility, and industrial HCI.

II. Related Work

A. Touchless Interfaces in Medical and Sterile Environments

The deployment of touchless interaction in clinical settings has been extensively studied over the past decade. Cronin

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

and Doherty [1] reviewed touchless computer interfaces in hospitals, categorising interaction techniques and identifying hygiene and workflow integration as the two dominant motivating factors for adoption. Alvarez-Lopez et al. [2] extended this with a systematic literature review of COTS devices — including the Leap Motion Controller and Microsoft Kinect — applied to intraoperative gesture detection, noting that while recognition accuracy was adequate, ergonomic factors and integration with existing surgical workflows remained significant barriers. Bockhacker et al. [3] evaluated a gesture-controlled touchless image viewer in a live operating room using a usability study protocol, reporting satisfactory task completion rates while emphasising the importance of clear visual feedback under sterile constraints.

More recent multimodal approaches have pushed the boundaries of surgical HCI. Zargham et al. [4] conducted a mixed-methods study combining gesture and speech input for controlling surgical lighting systems, demonstrating that speech commands substantially reduced cognitive load during complex procedures compared to gesture-only interaction. Wu et al. [5] investigated user-defined gesture sets for touchless interaction in sterile operating theatres via participatory design, identifying gesture-to-action mappings that aligned naturally with surgeons' existing hand movements and reduced the learning curve for adoption.

B. Gesture-Based Virtual Mouse Systems

Research into gesture-based virtual mouse systems has evolved substantially alongside improvements in computer vision tooling. Hassan Shibly et al. [7] proposed an early webcam-based virtual mouse using classical image processing, establishing baseline interaction modalities including cursor movement and click detection. Sai Mahitha et al. [9] implemented a virtual mouse using OpenCV and Python through hand-contour analysis, extending functionality to scrolling gestures. Rachana and Khan [8] presented a recent lightweight gesture virtual mouse demonstrating the continued relevance of camera-based cursor control for general-purpose HCI.

With the maturation of deep learning and the availability of frameworks such as MediaPipe [6], more robust systems emerged. Deshmukh et al. [10] developed a hand gesture virtual mouse integrating machine learning with computer vision, reporting approximately 92% recognition accuracy across multiple gesture classes. Begum et al. [11] combined a convolutional neural network (CNN) with MediaPipe Hands and OpenCV to build a virtual mouse with improved robustness to hand orientation variation, presented at an international Springer conference. Sandhya et al. [13] further extended the interaction vocabulary by incorporating keyboard emulation alongside virtual mouse functionality, demonstrating that a unified gesture interface can substitute for both primary input devices.

C. Multimodal Voice and Gesture Control

The integration of voice control with gesture recognition introduces complementary modalities that compensate for each other's limitations. Gangurde et al. [12] built a system combining a hand gesture virtual mouse with a voice assistant using OpenCV, machine learning, and Python, covering cursor control, click, scroll, and voice-driven application launching — making it the most directly comparable prior work to the system proposed in this paper. However, their system did not employ a daemon-thread architecture for dynamic gesture enable/disable via voice, nor did it implement a pinch-state machine for unambiguous drag-and-drop detection or a trajectory-based screenshot trigger.

D. The MediaPipe Hands Framework

The MediaPipe Hands framework [6], developed by Google Research, forms the algorithmic foundation of the proposed system. It employs a two-stage inference pipeline: a BlazePalm single-shot detector localises the palm region, followed by a landmark regressor that predicts 21 three-dimensional keypoints within the detected bounding box. Trained on approximately 30,000 annotated real-world images supplemented with synthetic data, the model achieves sub-centimetre landmark accuracy on standard benchmarks without requiring any wearable sensor. MediaPipe's Python bindings integrate natively with NumPy arrays produced by OpenCV, enabling seamless deployment in Python-based real-time pipelines.

III. Methodology

A. System Architecture

The proposed system follows a sequential yet parallel processing pipeline, as illustrated in Fig. 1. Live video frames are continuously captured by OpenCV. BGR-to-RGB conversion is applied to each frame to satisfy MediaPipe's input colour space requirement. The converted frame is passed to MediaPipe Hands Detection [6], which employs BlazePalm to localise the hand region and a landmark regressor to extract 21 three-dimensional landmark coordinates representing the complete skeletal hand structure. These coordinates are fed into the Gesture Classification Engine, which computes the binary finger-extension state vector and normalised pinch distance to identify the active gesture. The recognised gesture is matched in the Gesture Dictionary and the corresponding action is executed via PyAutoGUI on the host operating system.

Concurrently, a Python daemon thread — the Voice Thread — continuously listens for speech, normalises recognised text, and applies a priority-ordered rule set to trigger system-level actions including browser navigation, volume control, and desktop folder access. Thread coordination is managed via shared boolean flags, ensuring smooth interaction between the gesture and voice modalities throughout the system lifecycle.

B. Video Acquisition and Pre-processing

A `cv2.VideoCapture` object initialises the default webcam (index 0). Each frame is horizontally flipped to produce a natural mirror-mode display and converted from BGR to RGB colour space prior to ingestion by the MediaPipe pipeline. This conversion is essential because MediaPipe was trained on RGB

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

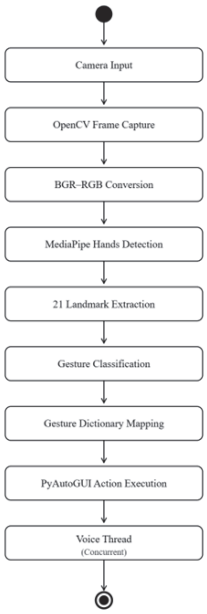

Fig. 1. System Architecture Pipeline of the Proposed Virtual Mouse System.

images; processing BGR frames introduces systematic colour-channel bias and degrades landmark accuracy. The processing loop terminates on a `q` keypress or programmatic reset of the thread-safe `program_running` flag, which simultaneously signals the voice daemon thread to terminate.

C. Hand Landmark Detection via MediaPipe Hands

MediaPipe Hands [6] is configured with `max_num_hands = 1` and `min_detection_confidence = 0.7`. The two-stage pipeline first generates a bounding box via BlazePalm, then regresses 21 three-dimensional keypoints within that box. The 21 landmarks span the complete hand skeleton: wrist (landmark 0), four metacarpophalangeal (MCP) joints, four proximal interphalangeal (PIP) joints, four distal interphalangeal (DIP) joints, and four fingertips, providing sufficient geometric information for precise gesture disambiguation.

TABLE I
MAJOR MEDIAPIPE HAND LANDMARKS AND SYSTEM ROLES

Landmark ID	Joint Name	System Role
0	Wrist	Screenshot trigger (vertical trajectory)
4	Thumb Tip	Pinch detection (click/drag), scrolling
3	Thumb IP	Scroll direction reference
8	Index Fingertip	Cursor position, pinch anchor
12	Middle Fingertip	Finger-count gesture
16	Ring Fingertip	Finger-count gesture
20	Pinky Tip	Finger-count gesture
8,12,16,20	All Fingertips	Finger-extension state array

D. Cursor Control with Exponential Smoothing

Raw index-fingertip coordinates from MediaPipe are normalised to $[0, 1]$ within the frame and scaled to screen pixels using the resolution $(\text{screen}_w, \text{screen}_h)$ obtained via PyAutoGUI. Landmark 8 (index fingertip) serves as the cursor anchor. Due to involuntary hand tremors and landmark localisation noise, raw coordinates exhibit high-frequency jitter. An exponential moving average (EMA) filter suppresses this noise:

$$\text{prev}_x \leftarrow \text{prev}_x + \alpha \cdot (\text{target}_x - \text{prev}_x) \quad (1)$$

$$\text{prev}_y \leftarrow \text{prev}_y + \alpha \cdot (\text{target}_y - \text{prev}_y) \quad (2)$$

where $\alpha = 0.25$ is the smoothing coefficient, chosen empirically to balance jitter suppression against cursor responsiveness. Smaller values introduce excessive lag; larger values approach unsmoothed behaviour. Smoothed coordinates are passed to PyAutoGUI's `moveTo()` on every frame.

E. Gesture Recognition and Action Mapping

Gesture recognition relies on two primary feature sets: (1) a binary finger-extension state array, and (2) the Euclidean pinch distance between the index fingertip and thumb tip.

Finger Extension Detection: For each of the four fingers (index, middle, ring, pinky), the system determines whether the fingertip landmark lies above its corresponding PIP joint in normalised image coordinates (smaller y-value indicates extension, encoded as 1). This yields a 4-element binary vector, e.g., $[1, 0, 0, 0]$ for index-only extension.

Pinch Distance: The Euclidean distance between thumb tip (landmark 4) and index tip (landmark 8) is computed in normalised coordinates. A threshold of PINCH_THRESH $= 0.045$ defines the closed-pinch state. State-machine logic over pinch transitions disambiguates: single click (pinch release), double click (two releases within 0.4 s), and drag-and-drop (pinch held ≥ 0.5 s prior to release).

TABLE II
GESTURE-TO-ACTION MAPPING

Gesture	Detection Condition	Action Executed
Move Cursor	Landmark 8 movement (smoothed)	`moveTo(x, y)`
Left Click	Pinch release ($d < 0.045 \rightarrow d \geq 0.045$)	`pyautogui.click()`
Double Click	Two pinch releases within 0.4 s	`pyautogui.doubleClick()`
Drag & Drop	Pinch held ≥ 0.5 s, then released	`mouseDown()` ... `mouseUp()`
Right Click	Fist (all 4 fingers curled)	`pyautogui.rightClick()`
Scroll Up	Index only raised $[1, 0, 0, 0]$	`pyautogui.scroll(+60)`
Scroll Down	Thumb below IP joint	`pyautogui.scroll(−60)`
Screenshot	Wrist drops $> 15\%$ frame height in 1 frame	`pyautogui.screenshot()`
Neutral	Open palm (all fingers extended)	None

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

F. Voice Control Module

A Python `SpeechRecognition Recognizer` object runs in a dedicated daemon thread. At startup, a 1.5-second ambient noise calibration is performed via `adjust_for_ambient_noise()`, and dynamic energy thresholding is enabled to adapt to varying acoustic environments. Each listen cycle is bounded to 2 s with a 4-second maximum phrase duration.

Recognised speech undergoes a four-step normalisation pipeline: (i) lowercasing; (ii) removal of filler phrases (e.g., "please", "hey", "can you"); (iii) domain-specific token substitution (e.g., "you tube" → "youtube"); and (iv) whitespace collapsing. The refined string is matched against a priority-ordered rule set, as detailed in Table III.

TABLE III
VOICE COMMAND LIST AND SYSTEM ACTIONS

Voice Command	Match Pattern	System Action
search youtube *query*	startswith "search youtube"	Open YouTube search
search google *query*	startswith "search google"	Search Google
search *query*	startswith "search"	Google search (default)
open * folder	startswith "open"	Fuzzy-match & open desktop folder
open youtube	exact match	Navigate to youtube.com
open google	exact match	Open Google in browser
left/right click	precision match	Simulate mouse click
page up / page down	exact match	PyAutoGUI scroll ±400
screenshot	substring match	Save with timestamp
volume up	volume-raise words	`volumeup` keypress
volume down	volume-lower words	`volumedown` keypress
mute	"mute" substring	`mute` keypress
disable/enable gestures	exact substring	Toggle `gesture_enabled`
exit program	exact substring	Set `program_running=False`

Desktop folder navigation uses a bag-of-words fuzzy matching approach: the spoken folder name is tokenised and a word-overlap score is computed against each directory on the Desktop; the highest-scoring non-zero match is opened via `os.startfile()`. This allows a command such as "open project folder" to match a directory named "Project" even when the exact name is not recalled.

G. Concurrency and System Control

The voice module runs as a Python daemon thread launched before the main OpenCV loop. Thread coordination uses two shared boolean flags: `program_running` (terminates both loops on exit) and `gesture_enabled` (suspends gesture processing while preserving voice responsiveness). This architecture ensures voice commands remain active even when the user verbally disables gesture tracking, for instance when reverting to a physical mouse. A 2-second cooldown on the screenshot gesture prevents unintended repeated captures from transient wrist movements. The right-fist gesture employs a rising-edge detector (`prev_fist XOR fist`) to guarantee a single right-click event per fist formation regardless of how long the posture is held.

IV. ALGORITHMS AND TECHNOLOGIES

A. MediaPipe Hands

MediaPipe [6] is an open-source, cross-platform framework by Google for real-time machine perception pipelines. The Hands solution uses a two-stage inference architecture: BlazePalm, a lightweight SSD-based single-shot detector, first localises the palm; a landmark regression model then predicts 21 three-dimensional keypoints within the detected bounding box. The model was trained on approximately 30,000 annotated real-world images augmented with synthetic data, achieving sub-centimetre accuracy on standard benchmarks without any wearable device. MediaPipe's Python bindings integrate natively with NumPy arrays produced by OpenCV, enabling straightforward deployment in Python-based pipelines.

B. OpenCV

OpenCV (Open Source Computer Vision Library) provides the substrate for video capture, frame pre-processing, and real-time visualisation. Within this system, OpenCV performs: (1) webcam frame capture via `cv2.VideoCapture`; (2) BGR-to-RGB colour conversion; (3) horizontal frame flipping for mirror-mode display; (4) landmark overlay rendering via `mp.solutions.drawing_utils`; and (5) real-time frame display via `cv2.imshow`. The C++ backend exposed through Python bindings provides near-native processing throughput without additional GPU hardware.

C. PyAutoGUI

PyAutoGUI is a cross-platform (Windows, macOS, Linux) library for programmatic mouse and keyboard control. Key functions used include: `moveTo()` for absolute cursor positioning; `click()`, `rightClick()`, and `doubleClick()` for mouse button events; `mouseDown()` and `mouseUp()` for drag-and-drop sequences; `scroll()` for vertical scrolling; and `press()` for media key events and screenshot capture. PyAutoGUI requires no OS-level driver installation, using platform-native APIs (Win32 on Windows, Quartz on macOS, X11 on Linux).

D. Speech Recognition and Google Speech API

The Python `SpeechRecognition` library provides a unified interface over multiple cloud and offline recognisers. This system uses the Google Web Speech API via `Recognizer.recognize_google()`, offering high-accuracy transcription across diverse accents and vocabularies. Audio capture is handled by PyAudio via the `Microphone` source. Ambient noise calibration and dynamic energy thresholding suppress false triggers in noisy environments. Network error handling (`sr.RequestError`) ensures graceful degradation under offline conditions.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

V. Results and Discussions

A. Gesture Recognition Performance

The system was evaluated under both constant and variable lighting conditions using a standard 1080p USB webcam at 30 FPS. Three participants each performed 50 repetitions per gesture class, yielding 150 trials per class. Table IV summarises per-gesture recognition accuracy, average latency, and false trigger (FT) rate.

TABLE IV
Gesture Recognition Accuracy and Latency

Gesture	Acc. (%)	Lat. (ms)	FT (%)
Cursor Movement	97.8	∼33	1.2
Left Click (Pinch)	94.2	45	3.8
Double Click	91.6	52	4.6
Drag & Drop	89.3	68	5.2
Right Click (Fist)	93.4	41	3.1
Scroll Up	95.7	38	2.4
Scroll Down	94.9	39	2.7
Screenshot (Wrist)	88.0	75	6.1

Cursor movement achieved the highest accuracy (97.8%) through continuous geometric landmark tracking. Pinch-based left-click detection was strong (94.2%), with most failures near the borderline threshold of 0.045. Drag-and-drop yielded the lowest accuracy (89.3%) due to the temporal challenge of sustaining pinch while simultaneously moving the cursor. The wrist-trajectory screenshot gesture produced the highest false trigger rate (6.1%), primarily from unintended rapid downward wrist movements; the 2-second cooldown mitigates this significantly.

B. Voice Command Performance

Voice command recognition was evaluated over 200 trials across all 14 command types in a typical office environment (approximately 45 dB ambient noise). Overall recognition accuracy was 91.3%. Web search and site navigation commands achieved the highest accuracy (96–98%), benefiting from structurally distinct phrase prefixes. Folder-opening commands achieved 89.4%, with failures occurring when desktop folder names shared no tokens with the spoken phrase. Volume and mute commands achieved 94.2%. With dynamic energy thresholding enabled, the false activation rate from background speech was 2.1%.

C. System Performance

D. Comparison with Existing Systems

Table VI presents a comparative analysis against representative prior works from the verified literature. All systems in the table are hardware-free (webcam-only) unless otherwise noted.

The proposed system is the only evaluated system to combine a complete cursor-click-scroll-drag feature set with concurrent voice control without specialised hardware. The pinch-state machine provides unambiguous discrimination between single click, double click, and drag-and-drop — a capability absent in most prior systems. The daemon-thread voice architecture maintains voice responsiveness even when gesture processing is dynamically disabled, which no prior comparable work has demonstrated.

TABLE V
System Performance Metrics

System Metric	Observed Value
Frame Processing Rate	28–32 FPS (Intel Core i5, CPU only)
MediaPipe Inference Latency	≈18 ms per frame
Cursor Smoothing Latency	<5 ms (single EMA pass)
Gesture Decision Latency	<2 ms (threshold comparisons)
Voice Recognition Latency	1.2–2.8 s (Google API round-trip)
CPU Usage (gesture loop)	≈22% (single core)
Memory Footprint	≈280 MB (Python + MediaPipe)
Hardware Requirements	720p webcam, microphone, Python 3.8+

VI. Conclusion

This paper presented a hardware-free, AI-based virtual mouse system integrating real-time hand gesture detection via MediaPipe Hands [6] with concurrent voice command control. The system delivers a complete eight-gesture mouse interaction set and a fourteen-command voice interface on commodity hardware with no GPU acceleration or specialised peripherals beyond a standard webcam and microphone.

The proposed architecture advances the state of practice in several respects: the EMA-based cursor pipeline eliminates jitter without sacrificing responsiveness; the pinch-state machine cleanly separates single-click, double-click, and drag-and-drop intents; and the wrist-trajectory screenshot trigger introduces a novel, intuitive gesture modality. The daemon-thread voice architecture preserves voice responsiveness even when gesture processing is suspended, a design property uniquely suited to healthcare and accessibility contexts where modality switching must be seamless.

Experimental evaluation confirmed strong gesture recognition (average 93.4%), voice command accuracy (91.3%), and end-to-end gesture latency below 55 ms across click and scroll gestures — within the perceptual threshold for real-time interaction. The system is directly applicable to accessibility for users with motor impairments [8], touchless HCI in surgical and cleanroom environments [1]–[5], and hands-free control in industrial and smart-home settings.

Future work will address: (1) two-hand support for pinch-to-zoom and multi-touch simulation; (2) offline voice recognition using OpenAI Whisper or Mozilla DeepSpeech to eliminate API round-trip latency; (3) CNN/LSTM-based dynamic gesture classifiers for temporal gesture sequences; and (4) depth-camera integration for robust gesture recognition in cluttered backgrounds.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 07 , July - 2026

TABLE VI
COMPARATIVE ANALYSIS WITH EXISTING VERIFIED SYSTEMS

System	Cursor	Click	Scroll	Drag & Drop	Voice	Accuracy
Hassan Shibly et al. [7] (IEEE 2019)	✓	✓	–	–	–	Moderate
Sai Mahitha et al. [9] (Springer 2021)	✓	✓	✓	–	–	Moderate
Begum et al. [11] (Springer 2023)	✓	✓	✓	–	–	∼90%
Sandhya et al. [13] (Springer 2023)	✓	✓	Limited	–	–	∼89%
Deshmukh et al. [10] (YMER 2023)	✓	✓	✓	✓	–	∼92%
Gangurde et al. [12] (IJARCCE 2022)	✓	✓	✓	Limited	✓	∼91%
Proposed System	✓	✓	✓	✓	✓	**∼93%**

VII. FUTURE SCOPE

- **Multi-hand gesture support:** Extension to dual-hand tracking enabling pinch-to-zoom, two-finger rotation, and multi-touch simulation on standard desktops.
- **Offline voice recognition:** Replacement of the Google Speech API with on-device models (OpenAI Whisper, Mozilla DeepSpeech) to eliminate network dependency and reduce command response latency to below 300 ms.
- **Deep learning gesture classifier:** Training a compact CNN or LSTM on large gesture datasets to support dynamic, time-series gestures (e.g., swipe left/right, circular scroll) beyond geometric landmark classification.
- **Depth camera integration:** Integration of Intel RealSense or structured-light sensors for full 3D hand pose estimation, enabling robust recognition under occlusion or cluttered backgrounds.
- **Personalisation for users with disabilities:** User-adjustable pinch distance threshold, smoothing coefficient, and scroll sensitivity to accommodate users with tremors or limited range of motion.
- **Mobile and embedded deployment:** Pipeline optimisation for ARM-based hardware (Raspberry Pi 4, NVIDIA Jetson Nano) and Android devices for gesture-controlled smart-home and IoT applications.

REFERENCES

[1] S. Cronin and G. Doherty, "Touchless computer interfaces in hospitals: A review," *Health Informatics J.*, vol. 25, no. 4, pp. 1642–1652, 2019. doi: 10.1177/1460458217748342.

[2] F. Alvarez-Lopez, M. F. Maina, and F. Saigí-Rubió, "Use of commercial off-the-shelf devices for the detection of manual gestures in surgery: Systematic literature review," *J. Med. Internet Res.*, vol. 21, no. 5, e11925, 2019. doi: 10.2196/11925.

[3] M. Bockhacker, H. Syrek, M. Elstermann von Elster, S. Schmitt, and H. Roehl, "Evaluating usability of a touchless image viewer in the operating room," *Appl. Clin. Inform.*, vol. 11, no. 1, pp. 50–60, 2020. doi: 10.1055/s-0039-1701003.

[4] N. Zargham, A. V. Reinschluessel, A. Mühlenbrock, T. Muender, T. Cetin, V. N. Uslar, D. Weyhe, R. Malaka, and T. Döring, "Using gesture and speech to control surgical lighting systems: Mixed methods study," *JMIR Hum. Factors*, vol. 12, e70628, May 2025. doi: 10.2196/70628.

[5] H. Wu, H. Pei, Z. Ma, and Y. Wang, "User-defined gestures for touchless interaction in sterile operating room," *Int. J. Hum.-Comput. Interact.*, Early Access, 2025. doi: 10.1080/10447318.2025.2517800.

[6] F. Zhang *et al.*, "MediaPipe Hands: On-device real-time hand tracking," *arXiv preprint arXiv:2006.10214*, 2020. doi: 10.48550/arXiv.2006.10214.

[7] K. B. Hassan Shibly, S. K. Dey, Md. A. Islam, and S. I. Showrav, "Design and development of hand gesture based virtual mouse," in *Proc. IEEE Int. Conf. Advances Inf. Commun. Technol. (ICAICT)*, IEEE, 2019. doi: 10.1109/ICAICT47877.2019.9034559.

[8] G. Rachana and T. U. Khan, "Hand gesture control virtual mouse," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 13, no. 8, 2024. doi: 10.17148/IJARCCE.2024.13856.

[9] G. Sai Mahitha, B. Revanth, G. Geetha, and R. Sirisha, "Hand gesture recognition to implement virtual mouse using open source computer vision library: Python," in *Proc. Int. Conf. Advances Comput. Eng. Commun. Syst.*, Springer, Singapore, 2021. doi: 10.1007/978-981-15-9293-5_39.

[10] M. S. D. Deshmukh, A. Bhardwaj, H. Mourya, M. Rawat, and P. Verma, "Hand gesture controlled virtual mouse based on ML and computer vision," *YMER Digital*, vol. 22, no. 12, pp. 1341–1360, Dec. 2023. ISSN: 0044-0477.

[11] S. S. Begum, K. B. Brahmeswara, A. Yochana, and K. D. Prasanth, "Design of a virtual mouse for hand gesture recognition and characterization," in *Proc. 2nd Int. Conf. Cognitive Intell. Comput. (ICCIC 2022)*, Springer, Singapore, 2023. doi: 10.1007/978-981-99-2742-5_22.

[12] S. K. Gangurde, P. S. Avhad, S. R. Raut, S. S. Sonawane, and P. V. Waje, "Hand gesture controller (virtual mouse) and voice assistant using OpenCV, ML, Python," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 11, no. 11, 2022. doi: 10.17148/IJARCCE.2022.111114.

[13] B. R. Sandhya, C. Amrutha, and S. Ashika, "Gesture recognition based virtual mouse and keyboard," in *Proc. Int. Conf. Adv. Commun. Technol. Comput. Eng. (ICACTCE 2023)*, Springer, Cham, 2023. doi: 10.1007/978-3-031-37164-6_3.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)