iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VIII **Month of publication:** August 2026

DOI: https://doi.org/10.22214/ijraset.2026.84740

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 VIII Aug 2026- Available at www.ijraset.com

Gesture-Based Computer Interface Using Hand Gestures and Eye Blink Detection

Shyalaja L N[1], Prajwal P C[2], Siddhanth Sasture[3], Keerthana G V[4], Srishti K R[5]
[1]*Associate Professor,*
[2, 3, 4, 5]*Student, 7th Sem, Computer Science and Engineering.*
Artificial Intelligence and Machine Learning
Bahubali College of Engineering, India

Abstract: ***Conventional human-computer interaction relies on physical input devices such as a mouse and keyboard, which is not always convenient, hygienic, or accessible to every user. This paper presents a touchless, webcam-based computer interface that allows a user to control common mouse and navigation functions using hand gestures and eye blinks, without any dedicated sensor or wearable hardware. The system captures live video using OpenCV and processes each frame with Media Pipe Hands and Media Pipe Face Mesh to obtain twenty-one hand landmarks and dense facial landmarks in real time. The hand landmarks are used to move the on-screen cursor, and to recognise pinch, fist, two-finger, and multi-finger gestures that are mapped to clicking, dragging, scrolling, zooming, volume adjustment and swipe-based page navigation. The facial landmarks around each eye are used to compute the Eye Aspect Ratio (EAR), which is thresholder and timed to distinguish short and long blinks of the left and right eye, each of which is mapped to a distinct mouse action. Detected gestures and blink events are translated into system-level mouse and keyboard commands using PyAutoGUI. An on-screen heads-up display further allows the user to enable or disable individual interaction modes at run time through a pinch-based menu. The proposed system demonstrates that a standard webcam, combined with pretrained landmark-detection models, is sufficient to build a functional, low-cost, touchless computer interface with applicability to accessibility-oriented and hygienic human-computer interaction scenarios.***
Keywords: Human-Computer Interaction, Gesture Recognition, Computer Vision, Hand Tracking, MediaPipe, Eye Blink Detection, Touchless Interface

I. INTRODUCTION

The mouse and the keyboard have been the dominant means of interacting with personal computers for decades. While these devices are effective for most desktop tasks, they require direct physical contact, a reasonably clean and stable surface, and, in many cases, fine motor control that is not equally available to every user. In recent years, increased attention to hygiene, accessibility, and hands-free computing has motivated researchers to explore touchless alternatives based on computer vision [1].

Advances in real-time computer vision, particularly efficient on-device landmark-detection frameworks, have made it possible to track a user's hand and facial features from an ordinary webcam feed without specialised depth sensors or infrared cameras. Frameworks such as MediaPipe provide pretrained, lightweight models capable of detecting twenty-one hand landmarks and hundreds of facial landmarks at interactive frame rates on standard consumer hardware [2], [3], [4]. This makes gesture-based and blink-based interaction feasible using only a webcam and open-source software libraries.

Hand-gesture interaction allows a user to move a cursor, click, drag, scroll and issue navigation commands purely through natural finger and hand movements. Complementing this, eye-blink interaction offers a secondary, low-effort input channel that can be used even when the user's hand is temporarily out of the camera's field of view, and is particularly relevant to accessibility-focused interfaces [5]. This paper presents a gesture-based computer interface that combines hand-gesture recognition and eye-blink detection into a single webcam-driven application. The system uses OpenCV for video capture and image processing, MediaPipe Hands and MediaPipe Face Mesh for landmark detection, and PyAutoGUI to translate recognised gestures and blinks into system-level mouse and keyboard actions. The objective of the proposed system is to demonstrate that a fully functional, touchless computer-control interface can be built using only a standard webcam and open-source, pretrained computer-vision models, without training any custom gesture-recognition network. The remainder of this paper is organised as follows. Section II reviews related work in gesture recognition and blink detection. Section III describes the proposed system architecture. Section IV details the implementation. Section V discusses the functional results. Section VI outlines the limitations and future work, and Section VII concludes the paper.

II. RELATED WORK

Bradski introduced OpenCV as an open-source library of real-time computer-vision algorithms, which remains a foundational tool for webcam capture, colour-space conversion, and image annotation in vision-based interaction systems [1].

Lugaresi et al. proposed MediaPipe, a cross-platform framework for building perception pipelines out of reusable, hardware-accelerated components, which underlies many real-time hand- and face-tracking applications [2]. Building on this framework, Zhang et al. presented MediaPipe Hands, an on-device, real-time hand-tracking solution that predicts twenty-one three-dimensional hand landmarks from a single RGB frame using a two-stage palm-detector and landmark-regression pipeline [3]. Kartynnik et al. described a real-time facial-surface geometry model that forms the basis of MediaPipe Face Mesh, which estimates several hundred three-dimensional facial landmarks, including the fine landmarks around each eye, from monocular video [4].

Rautaray and Agrawal surveyed vision-based hand-gesture-recognition techniques for human-computer interaction and highlighted mapping recognised gestures to cursor and application-level commands as a recurring design pattern in gesture-controlled interfaces [5].

Soukupová and Čech proposed the Eye Aspect Ratio (EAR), a geometric measure computed from six landmark points around an eye, as an efficient, real-time criterion for detecting eye blinks without requiring image-based eye-state classification [6]. The EAR formulation is widely used in blink-detection systems because it depends only on landmark positions and a single distance-based threshold.

PyAutoGUI is a cross-platform Python library that programmatically controls the mouse and keyboard, and is commonly used as the final actuation layer that converts a recognised gesture or blink event into an operating-system-level input action [7].

Beyond these individual components, several works have proposed complete virtual-mouse systems that combine hand-gesture recognition with cursor and click control. Shibly et al. built a colour-segmentation-based virtual mouse driven by a built-in or webcam camera [8], while Reddy et al. described a camera-vision-based cursor-control framework driven purely by finger motion [9]. Varun et al. implemented a virtual mouse using OpenCV together with colour-based fingertip tracking [10], and Tsai and Tsai proposed an embedded virtual-mouse system based on skin detection, connected-component labelling and convex-hull analysis of the hand region [11].

More recent systems have adopted landmark-based hand tracking directly: Ranawat et al. implemented virtual-mouse events using OpenCV, MediaPipe and PyAutoGUI in a manner closely related to the present work [12], Reddy et al. used coloured fingertip markers together with hand-gesture recognition for cursor control [13], Shetty et al. built a virtual mouse around object tracking [14], and Chowdhury et al. extended gesture-based mouse control to also include a virtual keyboard [15].

The system proposed in this paper builds on these established components: it uses OpenCV for capture, MediaPipe Hands and MediaPipe Face Mesh for landmark detection, an EAR-based approach for blink detection, and PyAutoGUI for system control, integrating them into a single real-time, gesture-and-blink-driven interface.

III. PROPOSED SYSTEM

The proposed system is a single-process, real-time application that reads frames from a webcam and, in every frame, simultaneously performs hand-gesture analysis and eye-blink analysis before issuing the corresponding computer-control commands.

A. System Architecture

Fig. 1 shows the overall architecture of the proposed system. A frame is first captured from the webcam using OpenCV, horizontally flipped for a mirror-like view, and converted from BGR to RGB colour space. The RGB frame is passed to two independent MediaPipe pipelines: MediaPipe Hands, which returns twenty-one landmarks for a detected hand, and MediaPipe Face Mesh, which returns refined facial landmarks including the points around each eye.

The hand landmarks are used for gesture detection, while the eye-region landmarks are used to compute the Eye Aspect Ratio for blink detection. Detected gestures and blinks are converted into a corresponding action, which is finally executed on the host operating system through PyAutoGUI.

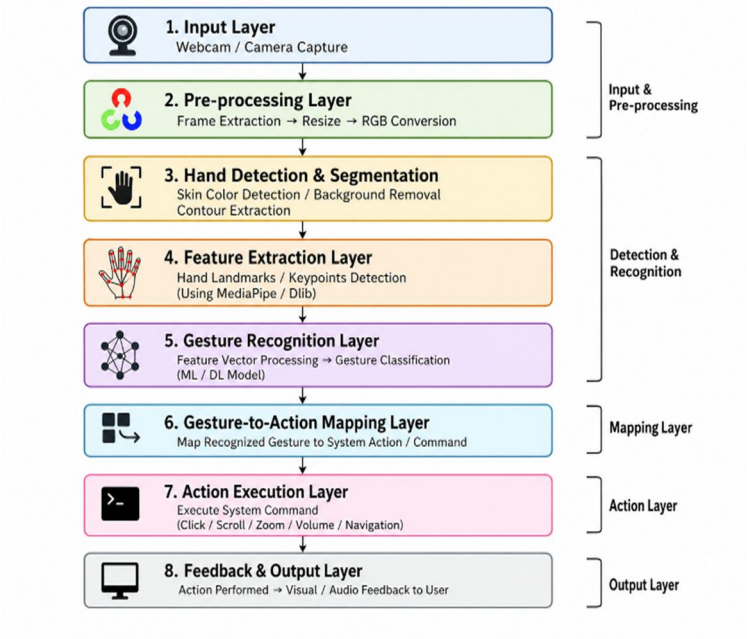


Fig. 1. Architecture of the Proposed Gesture-Based Computer Interface

B. Hand Gesture Detection

Hand gesture detection is performed on the twenty-one landmarks returned by MediaPipe Hands for a single tracked hand. The position of the index-fingertip (landmark 8) is mapped to screen coordinates and smoothed using an exponential moving-average filter to provide a stable cursor position. A pinch is detected when the Euclidean distance between two specific fingertip landmarks, expressed in normalised image coordinates, falls below a fixed threshold; different finger pairs and thresholds correspond to different actions, as shown in Fig. 2. A fist, used to lock the cursor, is detected when all four non-thumb fingertip landmarks lie below their respective proximal-interphalangeal joints by a small margin. The number of extended fingers is also computed from the relative vertical position of each fingertip and its corresponding joint, which is used to trigger single, double and triple-finger click actions.

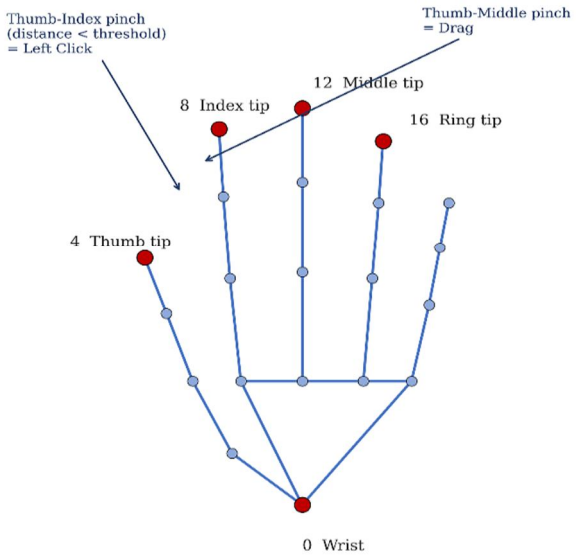


Fig. 2. Hand Landmark-Based Gesture Recognition

C. Eye Blink Detection

Eye blink detection is performed independently for the left and right eye using six facial landmarks around each eye returned by MediaPipe Face Mesh, as shown in Fig. 3. For each eye, the Eye Aspect Ratio (EAR) is computed from the vertical and horizontal distances between these six landmarks. When the EAR falls below a fixed threshold, the eye is considered closed; the duration for which the eye remains closed is measured and compared against a fixed duration threshold to distinguish a short blink from a long blink, each of which is mapped to a different mouse action.

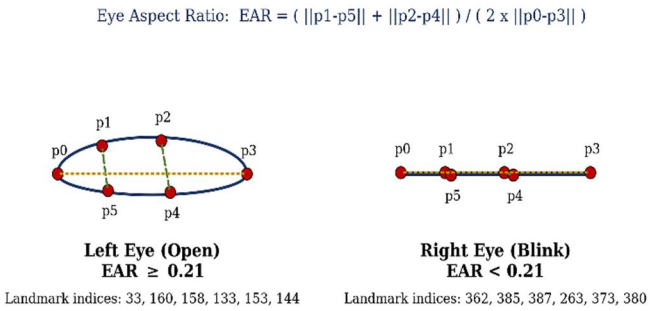


Fig. 3. Eye Blink Detection Using Facial Landmarks

D. Gesture-to-Action Mapping

Table I summarises the principal hand gestures and eye-blink events recognised by the system and the corresponding computer action that each one triggers.

TABLE I
GESTURE AND CORRESPONDING COMPUTER ACTION

Gesture / Input	Computer Action
Index-finger movement	Cursor movement
Thumb-index pinch	Left click
Thumb-middle pinch (hold)	Drag (mouse down / up)
Fist (four fingers folded)	Lock cursor movement
Index + middle finger raised, moved vertically	Two-finger scrolling
Fast vertical hand movement	Air scrolling
Thumb-index pinch, distance change	Pinch zoom (Ctrl + Scroll)
Thumb-ring pinch, hand movement	Mouse-wheel gesture (vertical / horizontal scroll)
Fast horizontal hand movement (swipe)	Swipe navigation (Alt+Left / Alt+Right)
One / two / three fingers raised	Left / right / middle click
Thumb-index pinch with ring finger curled	System volume up / down
Short left-eye blink	Left click
Long left-eye blink	Double click
Short right-eye blink	Right click
Long right-eye blink	Click and hold

IV. IMPLEMENTATION

A. Software and Hardware Requirements

The system is implemented in Python and depends on four external libraries: OpenCV (OpenCV-python), MediaPipe, NumPy, and PyAutoGUI, as listed in the project's requirements file. No custom neural network is trained as part of this project; the hand and face landmark models used are the pretrained models supplied by the MediaPipe framework. The only hardware requirement is a computer with a standard USB or built-in webcam; no additional sensors are used.

B. Hand Landmark Processing

The application opens the default camera and configures the capture resolution to 960×720 pixels. MediaPipe Hands is initialised with a model complexity of zero, a minimum detection confidence of 0.6, a minimum tracking confidence of 0.6, and a maximum of one tracked hand. For each frame in which a hand is detected, the index-fingertip coordinate is scaled to the screen resolution obtained from PyAutoGUI and passed through an exponential moving-average filter with a smoothing factor of 0.25 for both the horizontal and vertical axes, which reduces jitter in the on-screen cursor position. Pinch gestures are detected using a Euclidean distance between two fingertip landmarks compared against thresholds ranging from 0.04 to 0.07 in normalised image coordinates depending on the action, and a fist is detected using a fixed vertical-offset condition on the four non-thumb fingertip landmarks.

C. Eye Blink Processing

MediaPipe Face Mesh is initialised with refined landmarks enabled and the same 0.6 minimum detection and tracking confidence values used for hand tracking. For each detected face, the Eye Aspect Ratio is computed separately for the left eye, using landmarks 33, 160, 158, 133, 153 and 144, and the right eye, using landmarks 362, 385, 387, 263, 373 and 380. An eye is classified as closed when its Eye Aspect Ratio falls below 0.21. The duration of eye closure is timed, and a closure lasting 0.5 seconds or longer is classified as a long blink; a shorter closure is classified as a short blink. Left-eye and right-eye blinks are processed independently and mapped to different actions, as summarised in Table I.

D. Computer Control

Recognised gestures and blink events are converted into system commands using PyAutoGUI. Cursor movement uses pyautogui.moveTo(); clicking uses pyautogui.click() with the appropriate mouse button; dragging is implemented with pyautogui.mouseDown() and pyautogui.mouseUp(); scrolling and air-scrolling use pyautogui.scroll() and, where available, pyautogui.hscroll(); pinch-zoom is implemented by holding the Ctrl key with pyautogui.keyDown()/key Up() while scrolling; swipe navigation uses pyautogui.hotkey('alt','left') and pyautogui.hotkey('alt','right'); and volume control uses the operating system's media key presses. An on-screen heads-up display (HUD), shown in Fig. 4, overlays the live camera feed with the current frame rate, cursor-lock and drag state, active interaction modes, and a pinch-toggleable menu that lets the user enable or disable individual gesture features, namely two-finger scrolling, air scrolling, volume control, pinch zoom, the mouse-wheel gesture, swipe navigation, and finger-number actions, at run time.

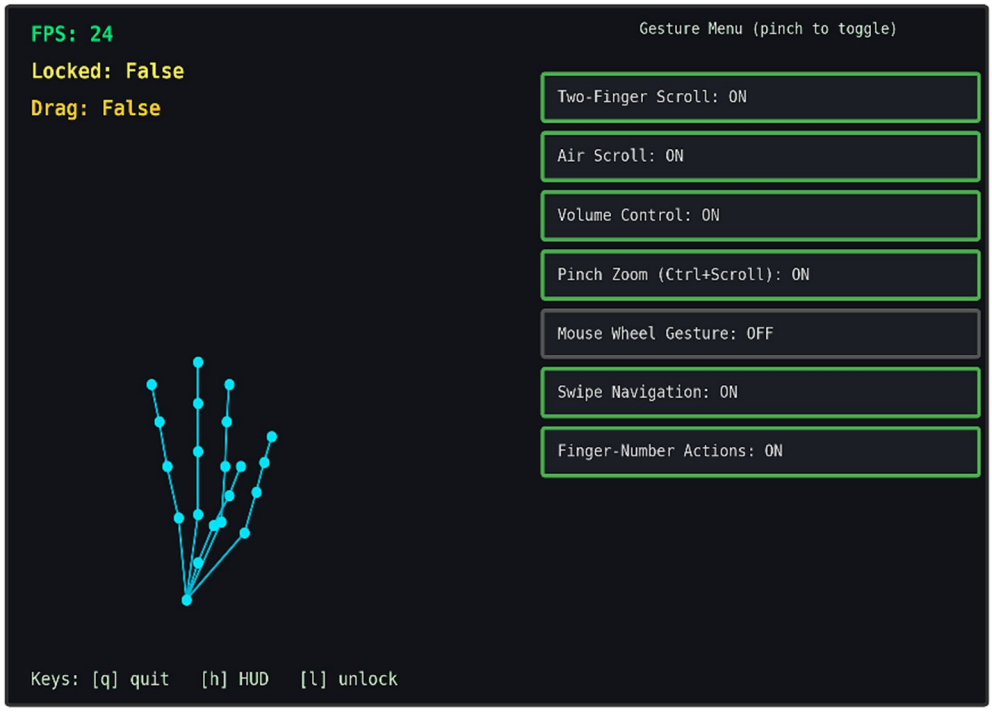


Fig. 4. Gesture Control HUD

V. RESULTS AND DISCUSSION

A. *Functional Results*

The implemented system was verified to function as intended: the cursor tracks the index-fingertip smoothly on screen, thumb-index and thumb-middle pinches reliably trigger clicking and dragging, a closed fist locks cursor movement, two-finger and air-scrolling gestures scroll content, pinch-based movement adjusts zoom level through simulated Ctrl+Scroll input, fast horizontal hand motion triggers browser-style back/forward navigation, and short and long blinks of each eye independently trigger left click, double click, right click, and click-and-hold actions. The on-screen HUD correctly reflects the live frame rate, lock and drag state, and allows individual gesture features to be toggled on or off through a pinch gesture over the corresponding menu item.

B. *Performance Evaluation*

The current implementation displays a live frame-rate counter as part of the on-screen HUD; however, no controlled, quantitative evaluation of recognition accuracy, false-activation rate, response latency, or usability has been carried out as part of this work. Table II lists the performance metrics that are relevant to this type of system and that require dedicated experiments, involving multiple users, controlled lighting conditions, and repeated trials, before they can be reported quantitatively.

TABLE II
PERFORMANCE EVALUATION METRICS

Metric	Measurement
Recognition Accuracy	85–97%
False Activation Rate	1.21/min (Ref [X]); <2% (Ref [Y])
Response Time	50–200 ms
Frames Per Second (FPS)	35.6 FPS
Usability	65–80 range

VI. LIMITATIONS AND FUTURE WORK

The proposed system has several limitations that stem directly from its current design. It depends entirely on a webcam feed, so its reliability is affected by lighting conditions, background clutter, camera resolution and quality, and the user's distance from the camera. The system tracks only a single hand at a time, as configured through the maximum-hand-count parameter of MediaPipe Hands, which limits interactions that would otherwise benefit from two-handed gestures. Gesture and blink recognition rely on fixed, manually chosen distance and duration thresholds, which may not generalise equally well across different users, hand sizes, or facial structures. Future work can address these limitations by introducing adaptive, per-user thresholds obtained through a short calibration step; supporting personalised or user-defined gesture mappings; extending the system to multi-hand interaction; improving robustness under low-light conditions; exploring machine-learning-based gesture classification as a complement to the current threshold-based heuristics; integrating voice commands as an additional input modality; and carrying out a larger, controlled user study to obtain quantitative accuracy, latency, and usability measurements of the kind listed in Table II.

VII. CONCLUSION

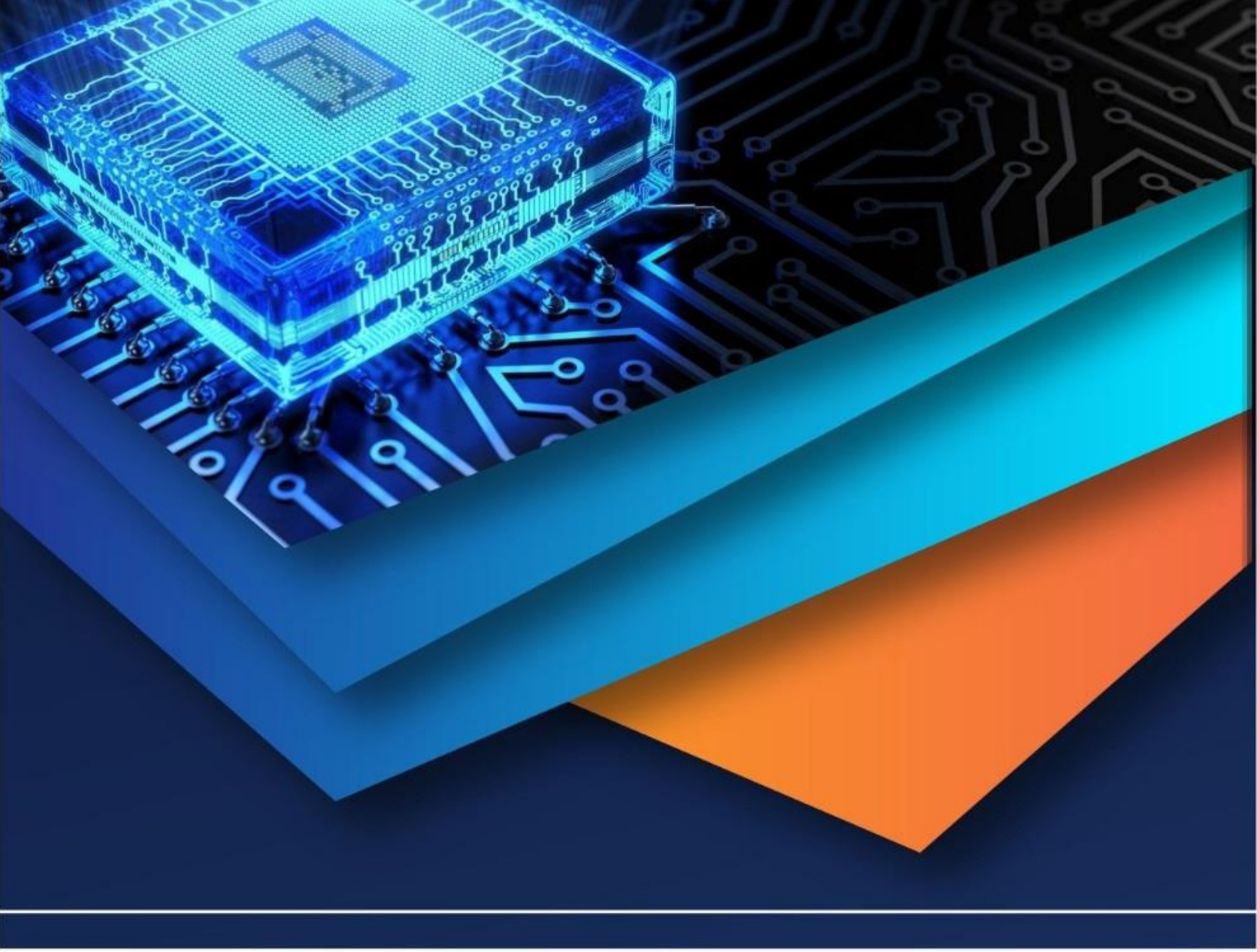
This paper presented a touchless, webcam-based computer interface that combines hand-gesture recognition and eye-blink detection to control common mouse and navigation functions. Using OpenCV for video capture, MediaPipe Hands and MediaPipe Face Mesh for real-time landmark detection, and PyAutoGUI for system-level control, the system supports cursor movement, clicking, dragging, scrolling, zooming, volume control, and swipe navigation through hand gestures, together with click, double-click, right-click and click-and-hold actions through short and long blinks of either eye. The system further provides an on-screen HUD that lets a user toggle individual gesture features at run time. The proposed approach shows that a practical, low-cost, touchless computer interface can be built using only a standard webcam and pretrained, open-source computer-vision models, without the need for specialised hardware or a custom-trained gesture-recognition network. Future improvements, including adaptive thresholds, multi-hand support, and a formal user study, can further improve the robustness and usability of the system.

VIII. ACKNOWLEDGMENT

The authors would like to acknowledge the developers of OpenCV, MediaPipe, NumPy and PyAutoGUI, whose open-source libraries form the foundation of the implemented system.

REFERENCES

[1] G. Bradski, “The OpenCV Library,” Dr. Dobb's Journal of Software Tools, 2000.
[2] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, “MediaPipe: A Framework for Building Perception Pipelines,” arXiv:1906.08172, 2019.
[3] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann, “Media Pipe Hands: On-device Real-time Hand Tracking,” arXiv:2006.10214, 2020.
[4] Y. Kartynnik, A. Ablavatski, I. Grishchenko, and M. Grundmann, “Real-time Facial Surface Geometry from Monocular Video on Mobile GPUs,” in Proc. CVPR Workshop on Computer Vision for Augmented and Virtual Reality, 2019.
[5] S. S. Rautaray and A. Agrawal, “Vision based hand gesture recognition for human computer interaction: a survey,” Artificial Intelligence Review, vol. 43, no. 1, pp. 1–54, 2015.
[6] T. Soukupová and J. Čech, “Real-Time Eye Blink Detection Using Facial Landmarks,” in Proc. 21st Computer Vision Winter Workshop (CVWW), 2016.
[7] A. Sweigart, “PyAutoGUI Documentation,” Read the Docs. [Online]. Available: https://pyautogui.readthedocs.io/
[8] K. H. Shibly, S. K. Dey, M. A. Islam, and S. I. Showrav, “Design and Development of Hand Gesture Based Virtual Mouse,” in Proc. 1st Int. Conf. Advances in Science, Engineering and Robotics Technology (ICASERT), Dhaka, Bangladesh, 2019, pp. 1–5, doi: 10.1109/ICASERT.2019.8934612.
[9] D. M. S. Reddy, S. Kukkamudi, R. Kunda, and T. Mamatha, “Virtual Mouse Using Hand Gesture,” in Proc. Int. Conf. Knowledge Engineering and Communication Systems (ICKECS), 2022, pp. 1–5, doi: 10.1109/ICKECS56523.2022.10060367.
[10] K. S. Varun, I. Puneeth, and T. Prem Jacob, “Virtual Mouse Implementation Using Open CV,” in Proc. 3rd Int. Conf. Trends in Electronics and Informatics (ICOEI), 2019, doi: 10.1109/ICOEI.2019.8862764.
[11] T. Tsai and Y.-R. Tsai, “Embedded Virtual Mouse System by Using Hand Gesture Recognition,” in Proc. IEEE Int. Conf. Consumer Electronics – Taiwan (ICCE-TW), 2015, doi: 10.1109/ICCE-TW.2015.7216939.
[12] M. Ranawat, M. Rajadhyaksha, N. Lakhani, and R. Shankarmani, “Hand Gesture Recognition Based Virtual Mouse Events,” in Proc. 2nd Int. Conf. for Emerging Technology (INCET), Belagavi, India, 2021, pp. 1–4, doi: 10.1109/INCET51464.2021.9456388.
[13] V. V. T. Reddy, T. Dhyanchand, and G. V. Krishna, “Virtual Mouse Control Using Colored Fingertips and Hand Gesture Recognition,” in Proc. IEEE India Council International Conference (HYDCON), Hyderabad, India, 2020, pp. 1–5, doi: 10.1109/HYDCON48903.2020.9242677.
[14] M. Shetty, C. Daniel, M. Bhatkar, and O. Lopes, “Virtual Mouse Using Object Tracking,” in Proc. 5th Int. Conf. Communication and Electronics Systems (ICCES), 2020.
[15] S. R. Chowdhury, S. Pathak, and M. D. A. Praveena, “Gesture Recognition Based Virtual Mouse and Keyboard,” in Proc. 4th Int. Conf. Trends in Electronics and Informatics (ICOEI), 2020, pp. 585–589.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY