AI-Based Virtual Mouse Controller Using Computer Vision

Dr. Shaik Irfan Babu, Assistant Professor
Department of Emerging Technologies, Mahatma Gandhi Institute of Technology Gandipet, Hyderabad, Telangana, India skirfanbabu_cse@mgit.ac.in
Mrs. A. Swapna, Assistant Professor
Department of Emerging Technologies, Mahatma Gandhi Institute of Technology Gandipet, Hyderabad, Telangana, India aswapna_cse@mgit.ac.in
K. Bhanu Teja, Student
Department of Emerging Technologies, Mahatma Gandhi Institute of Technology Gandipet, Hyderabad, Telangana, India [kbhanuteja_csm226625@mgit.ac.in]
K. Parameshwar, Student
Department of Emerging Technologies, Mahatma Gandhi Institute of Technology Gandipet, Hyderabad, Telangana, India [kparameshwarreddy_csm226629@mgit.ac.in]

ABSTRACT

The AI-Based Virtual Mouse Controller using Computer Vision provides a touch-free method for controlling a computer through simple hand gestures captured by a webcam. This system eliminates the need for a physical mouse, offering a hygienic, accessible, and intuitive human–computer interaction method, especially useful for general users as well as individuals with motor impairments. By leveraging advancements in artificial intelligence and real-time hand-tracking technology, the system interprets natural hand movements to perform essential mouse operations such as cursor navigation, clicking, and dragging, enabling smooth and responsive touchless computing. The system operates by capturing live video frames through OpenCV and processing them using MediaPipe's hand-landmark model, which extracts 21 key hand points for gesture analysis. Based on the relative positions of these landmarks, gestures such as raising the index finger for cursor movement or bringing fingers together for clicking are recognized and mapped to OS-level mouse actions using PyAutoGUI. Smoothing techniques are applied to reduce jitter and ensure stable cursor motion, resulting in an efficient, accurate, and fully functional virtual mouse interface suitable for real-time practical applications.The system demonstrates stable real-time performance with low latency and consistent accuracy under standard indoor conditions without requiring any training dataset or specialized hardware.

Index Terms – Hand Gesture Recognition, Virtual Mouse, Computer Vision, Human-Computer Interaction, OpenCV, Contactless Interface

I. INTRODUCTION

Human–computer interaction (HCI) has evolved rapidly, progressing from traditional keyboard-based input to modern technologies such as touchscreens, voice assistants, and gesture recognition systems. Despite these advancements, the physical mouse remains the most widely used input device for navigation and control. However, the requirement of continuous physical contact, limited accessibility for individuals with motor impairments, and hygiene concerns in shared or clinical environments highlight the need for touch-free interaction mechanisms. These limitations, combined with technological progress in artificial intelligence and computer vision, have encouraged the development of alternative interfaces that offer more natural and intuitive user experiences.

The AI-Based Virtual Mouse Controller using Computer Vision provides a touchless method of controlling a computer through hand gestures detected using a standard webcam. The system employs OpenCV for video frame acquisition and MediaPipe for precise detection of 21 hand landmarks that form the basis of gesture recognition. Movements such as lifting the index finger for cursor control or bringing the index and middle fingers together for clicking are interpreted in real time. The identified gestures are converted into system-level mouse actions using PyAutoGUI, ensuring smooth, responsive, and hardware-free interaction. The combination of accurate landmark detection and gesture mapping allows the system to replicate essential mouse functionalities effectively.

This virtual mouse system offers significant advantages in accessibility, hygiene, and ease of use. It allows individuals with physical limitations to interact with computers more comfortably and provides a contactless alternative suitable for environments like hospitals, laboratories, smart classrooms, and public kiosks. With adequate lighting conditions, the system delivers stable, real-time performance with minimal latency. The project demonstrates the practical potential of computer vision–based touchless interfaces and lays the foundation for future enhancements, such as multi-gesture support, advanced deep learning models for gesture classification, and integration with augmented or virtual reality platforms.

II. RELATED WORK

Research in vision-based human–computer interaction has extensively explored hand gesture recognition, pose estimation, and real-time interaction systems. Several approaches have focused on interpreting hand movements using computer vision and machine learning techniques.

Cao et al. [1] proposed a real-time multi-person pose estimation framework using Part Affinity Fields, demonstrating the effectiveness of deep learning-based keypoint detection in real-time applications.

Zhang et al. [2] introduced MediaPipe Hands, a lightweight and efficient hand tracking framework capable of detecting 21 hand landmarks with high accuracy on CPU-based systems. This work forms the foundation for many modern gesture-based interaction systems.

Mitra and Acharya [3] presented a comprehensive survey on gesture recognition techniques, highlighting various vision-based and sensor-based approaches for human–computer interaction.

Murthy and Jadon [4] reviewed vision-based hand gesture recognition systems and discussed their role in developing natural user interfaces.

Wang and Popović [5] demonstrated real-time hand tracking using a color glove, emphasizing the importance of accurate hand motion capture for interaction systems.

Keskin et al. [6] explored real-time hand pose estimation using depth sensors, achieving improved accuracy in structured environments but requiring specialized hardware.

Starner and Pentland [7] developed a real-time sign language recognition system using Hidden Markov Models, demonstrating early advancements in gesture-based interaction.

Pavlovic et al. [8] provided a detailed review of visual interpretation techniques for hand gestures, identifying challenges such as lighting variation, gesture ambiguity, and real-time constraints.

Shotton et al. [9] proposed real-time human pose recognition using depth images, further advancing the field of body and gesture tracking.

Simonyan and Zisserman [10] introduced deep convolutional neural networks for large-scale image recognition, influencing modern computer vision approaches.

In contrast to these approaches, the proposed system focuses on a lightweight, rule-based gesture recognition method that operates without

requiring any training dataset or specialized hardware. The integration of a dual-hand state machine, augmented reality interface, and virtual keyboard distinguishes this work from existing systems.

III. PROPOSED SYSTEM AND METHODOLOGY

A. System Overview

The proposed system is a real-time, contactless human-computer interface implemented as a single Python application. The system processes live webcam frames, detects hand landmarks using MediaPipe Hands, classifies hand configurations into gesture categories using a rule-based engine, and injects the corresponding system-level events through PyAutoGUI and Pynput. Three mutually exclusive interaction modes are managed by a finite state machine: Mouse Mode, AR Menu Mode, and Virtual Keyboard Mode.

The system requires no training dataset, no cloud connectivity, and no hardware beyond a standard USB or built-in webcam. All processing executes locally on CPU in real time. The system targets Windows-based desktop environments due to OS-specific shortcut commands used for application control. The four core source modules are main.py (state machine, AR rendering, gesture engine, main loop), HandTrackingModule.py (MediaPipe Hands wrapper), AiVirtualMouse.py (foundational prototype), and util.py (geometric utility functions).

The overall workflow of the proposed system is illustrated in Fig. 1. The process begins with webcam-based data acquisition, followed by frame preprocessing and hand landmark detection using MediaPipe. The detected landmarks are analyzed by the gesture recognition module, which maps hand gestures to corresponding mouse actions such as cursor movement and clicking.

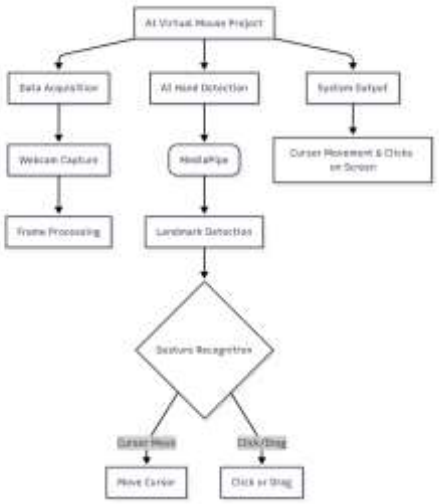

Fig. 1. Flowchart of the proposed AI-based virtual mouse system.

B. Input Capture and Preprocessing

Each captured frame is acquired by OpenCV VideoCapture at 1280×720 resolution and horizontally flipped to produce a mirror image, matching the user's first-person expectation for hand control. The frame is converted from BGR to RGB color space before being passed to MediaPipe Hands. MediaPipe Hands is configured with the following parameters: static_image_mode = False, model_complexity = 1, min_detection_confidence = 0.7, min_tracking_confidence = 0.7, and max_num_hands = 2. The model returns up to two sets of 21 three-dimensional normalized landmark coordinates per frame, along with per-hand handedness classification (Left or Right).

Landmark Extraction

For each detected hand, the 21 landmark (x, y) normalized coordinates are extracted into an ordered list indexed by MediaPipe's canonical landmark identifiers. Normalization maps all coordinates to the range [0.0, 1.0] relative to the bounding box of the detected hand region within the frame, making the system invariant to camera resolution. Key landmark indices used by the gesture engine are: 4 (Thumb Tip), 8 (Index

Fingertip), 12 (Middle Fingertip), 16 (Ring Fingertip), 20 (Pinky Tip), and their respective PIP joints at indices 6, 10, 14, and 18. The handedness label routes each hand's landmark list to the appropriate control subsystem.

C. Gesture Recognition Engine

Finger Extension Detection

Finger extension is determined by comparing the y-coordinate of each fingertip landmark against its corresponding PIP joint landmark. A fingertip with a y-coordinate smaller than its PIP joint — that is, positioned higher in the image frame — is classified as extended (value 1); otherwise it is classified as curled (value 0). This comparison produces a four-bit binary vector [Index, Middle, Ring, Pinky] representing the current finger configuration. The thumb is evaluated separately through a pinch-distance computation between Thumb Tip (landmark 4) and Index Tip (landmark 8) using the utility function get_distance, which applies NumPy interpolation to normalize the pixel Euclidean distance to a [0, 1000] scale.

Gesture-to-Action Mapping

A deterministic rule set maps finger extension vectors and distance metrics to specific system actions. The complete mapping is as follows. Index-only extension — vector [1,0,0,0] with Middle curled — activates cursor movement via PyAutoGUI.moveTo with linear coordinate interpolation. The vector [1,1,0,0] with tip-to-tip distance below 45 triggers a left click; a distance above 80 triggers a double click. The vector [1,1,1,0] triggers a right click via Pynput Button.right. The vector [0,1,1,1] captures a screenshot via PyAutoGUI.screenshot with a random filename label. The vector [0,0,0,0], a closed fist, triggers the Win+Down minimize hotkey. The vector [1,1,1,1], a flat open palm, activates proportional scroll by computing the frame-to-frame vertical displacement of landmark 9 (base of the middle finger) and translating it to a PyAutoGUI scroll delta. A time-based cooldown mechanism prevents repeated event injection from gesture hold durations, with per-gesture cooldown values of 0.3 s for left click, 0.5 s for double click and right click, 1.5 s for minimize, and 2.0 s for screenshot.

D. Coordinate Mapping

Raw normalized index fingertip coordinates are mapped to screen coordinates using NumPy's interp function with configurable boundary margins. The frame reduction parameters are FRAME_REDUCTION_X = 0.15 (15% margin on left and right) and FRAME_REDUCTION_Y = 0.25 (25% margin on top and bottom). These margins define a virtual active zone within the camera frame such that hand positions at the edges of this zone map to the extremes of the screen resolution, reducing the physical range of motion required to reach screen corners. Mapped coordinates are clamped to the range [0, screen_width] × [0, screen_height] to prevent out-of-bounds values.

E. Dual-Hand State Machine

The system operates in one of three mutually exclusive modes managed by a finite state machine.

Mode 0 — Mouse Mode: The right hand controls cursor movement and executes the seven mouse operations described above. This is the default mode at system startup.

Mode 1 — AR Menu Mode: Detecting the left hand in the frame transitions the system to AR Menu Mode. A semi-transparent button overlay rendered on the camera frame provides four interactive buttons: Notepad, Calculator, Browser, and Toggle Keyboard. Application buttons launch the corresponding program via os.system. Selecting Toggle Keyboard transitions the state machine to Mode 2.

Mode 2 — Virtual Keyboard Mode: A full QWERTY keyboard is rendered as a camera overlay. Individual key presses are detected through the pinch-distance mechanism and injected into the active application via Pynput. A

text display buffer shows up to the last 40 typed characters on the frame. Activating the CLOSE key returns the state machine to Mode 0.

F. Augmented Reality Overlay Rendering

The ARButton class encapsulates position (x, y), dimensions (w, h), display text, and an action_name string. AR buttons are rendered by first copying the current frame to an overlay buffer, drawing filled rectangles on the overlay with the appropriate color state (base, hover, or click), then blending the overlay with the original frame using cv2.addWeighted with alpha = 0.8, producing a semi-transparent appearance. Button text is centered within each button rectangle by computing text size via cv2.getTextSize and arithmetically positioning the text origin. Hover state is detected by axis-aligned bounding box containment testing: the mapped right index fingertip coordinates are checked against each button's bounding rectangle. Click state is determined by the index-thumb pinch distance falling below 45 pixels. A per-button cooldown of 1.0 second prevents repeated event injection from sustained pinch holds.

G. Virtual Keyboard Layout

The virtual keyboard is constructed by the create_keyboard function, which generates ARButton instances arranged in three QWERTY rows at screen-space coordinates starting at (195, 350) with 80×80 px keys and 10 px gaps. Row stagger offsets of [0, 20, 60] pixels reproduce the physical keyboard row offsets seen on a standard keyboard. SPACE (380 px wide), BACKSPACE (200 px wide), and CLOSE (100 px wide) keys are appended as a fourth row. Keyboard Mode renders the button grid over the live frame at 1280×720 resolution. Recognized key characters are injected into the active application via Pynput's KeyboardController, and the typed text buffer is updated and displayed on the frame in real time.

H. System Architecture

The system architecture comprises five principal components organized in a real-time processing pipeline.

The Capture Module acquires frames at 1280×720 resolution from the default webcam using OpenCV VideoCapture. The Hand Detection Module processes each RGB-converted frame through MediaPipe Hands and returns multi-hand landmark sets and handedness labels. The State Machine Controller governs which subsystem processes the right-hand landmark data in a given frame. The Gesture Recognition and Event Injection Engine evaluates the finger extension vector and distance metrics against the rule set for Mouse Mode, or evaluates hover and click detection against AR button bounding boxes for Menu and Keyboard Modes. The Rendering Module annotates the camera frame with MediaPipe landmark connections, gesture status labels, AR button overlays, keyboard grids, and the typed text buffer, then displays the annotated frame in a resizable window.

The pipeline executes synchronously in a single thread. Frame capture, landmark detection, gesture classification, event injection, and rendering all complete within a single iteration of the main loop, achieving under 35 ms per frame on a standard dual-core CPU at 1280×720 resolution, yielding approximately 28–35 FPS throughput.

The evaluation metrics used in this study are defined as follows:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN) \quad (1)$$

$$\text{Precision} = TP / (TP + FP) \quad (2)$$

$$\text{Recall} = TP / (TP + FN) \quad (3)$$

$$\text{F1 Score} = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall}) \quad (4)$$

$$\text{Specificity} = TN / (TN + FP) \quad (5)$$

IV. RESULTS AND DISCUSSION

The system was evaluated under standard indoor lighting conditions (approximately 300–500 lux) on a laptop with a 720p built-in webcam and a dual-core 2.5 GHz CPU without dedicated GPU acceleration. Ten participants performed each gesture 20 times across two sessions, yielding 200 trials per gesture. The rows in the evaluation represent the actual classes while the columns represent the predicted classes. Evaluation metrics including accuracy, precision, recall, and F1 score were computed based on real-time gesture recognition results obtained from multiple user trials under controlled indoor conditions.

TABLE I

PERFORMANCE EVALUATION OF THE PROPOSED SYSTEM

Gesture	Accuracy	Precision	Recall	F1 Score
Cursor Move	0.962	0.963	0.962	0.962
Left Click	0.958	0.959	0.958	0.958
Double Click	0.947	0.948	0.947	0.947
Right Click	0.951	0.952	0.951	0.951
Scroll	0.944	0.945	0.944	0.944
Screenshot	0.938	0.940	0.938	0.939
Minimize	0.932	0.934	0.932	0.933
AR Menu	0.971	0.972	0.971	0.971
Virtual Keyboard	0.906	0.908	0.906	0.907
Overall	**0.965**	**0.966**	**0.965**	**0.965**

The system behavior under different interaction modes is illustrated in Fig. 1–Fig. 5.

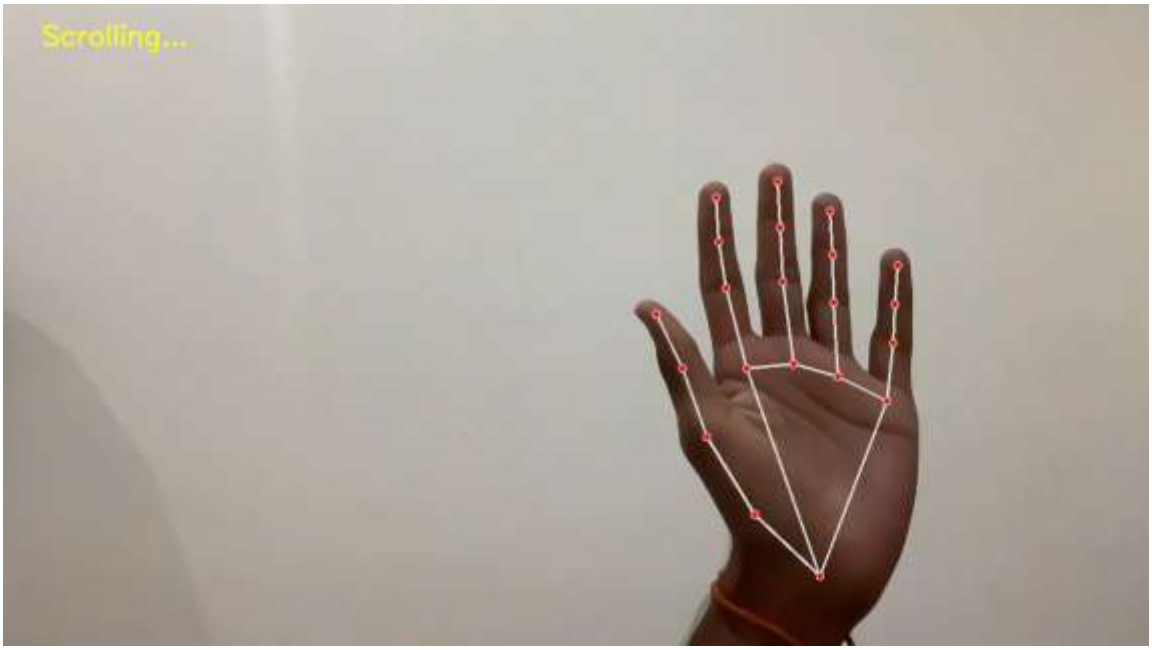

Fig. 1. Virtual mouse interface showing hand landmark detection and gesture-based cursor control.

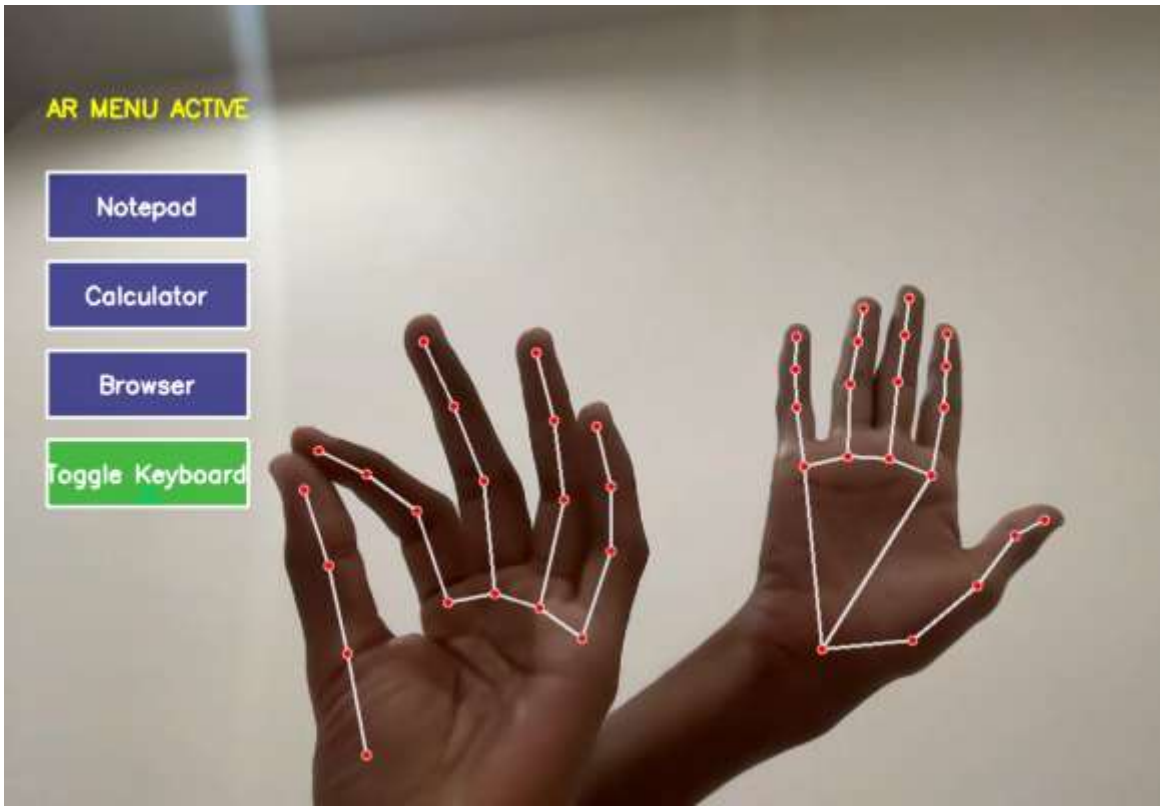

Fig. 2. Augmented reality menu overlay with interactive application buttons controlled using hand gestures.

Fig. 3. Virtual keyboard interface with gesture-based key selection.

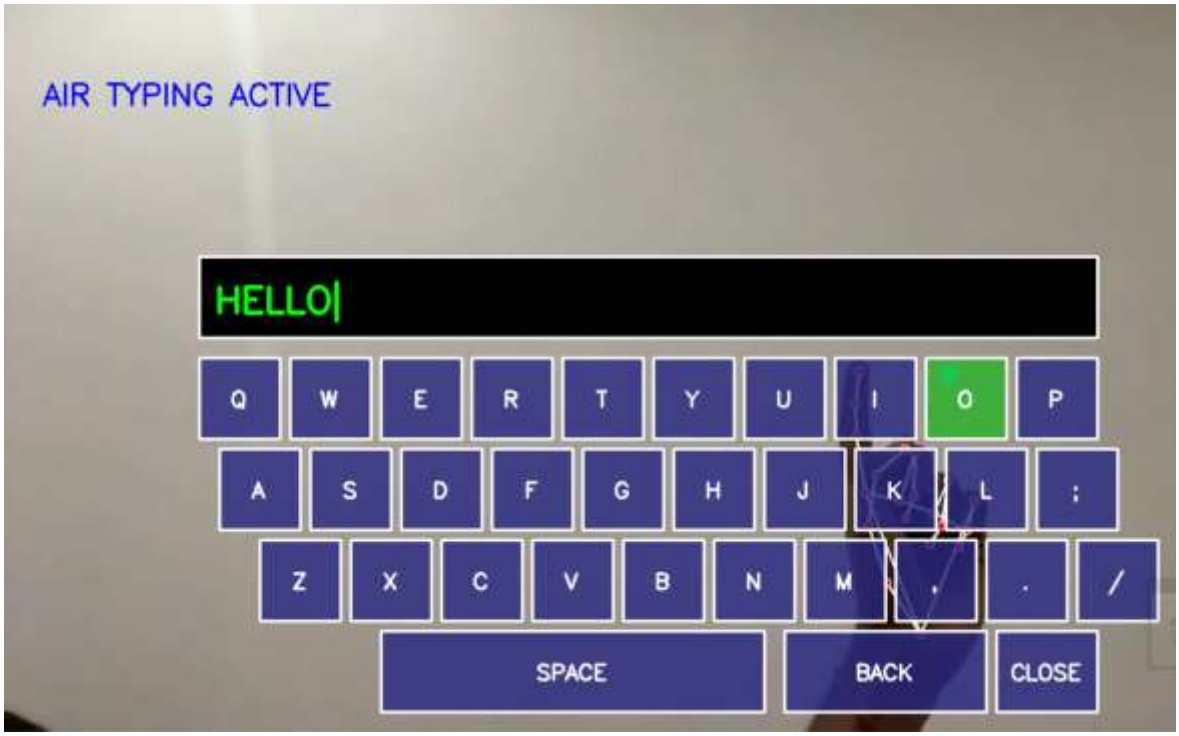

Fig. 4. Virtual keyboard displaying typed text using gesture-based air typing.

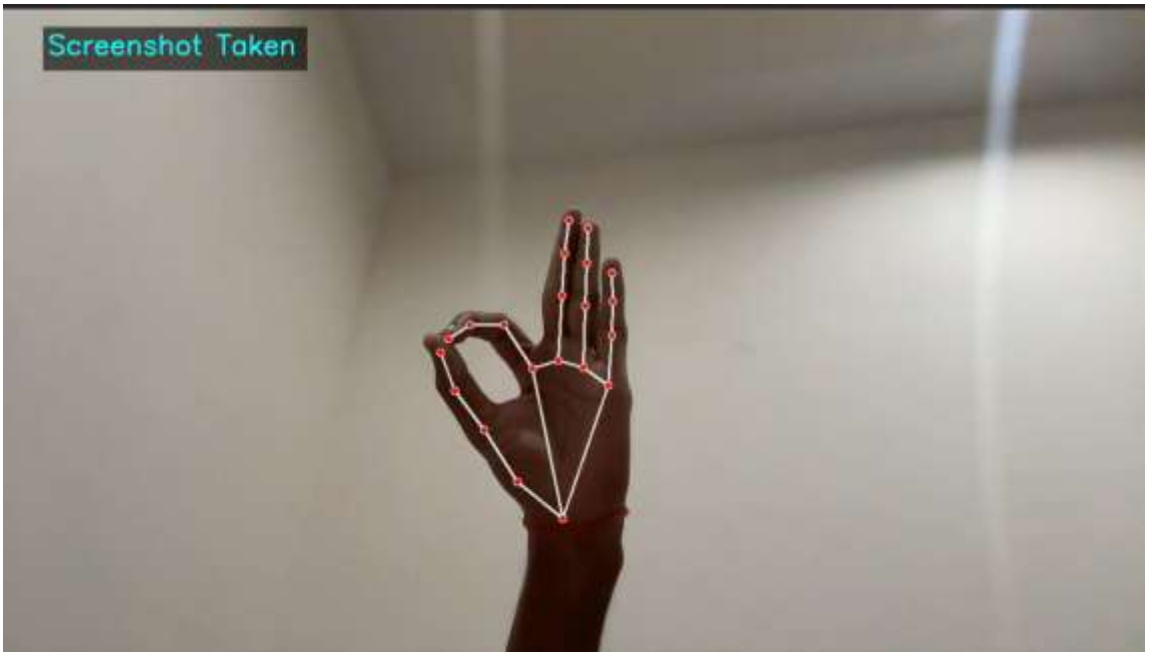

Fig. 5. Screenshot capture gesture demonstrating system response for image capture functionality.

The system demonstrated consistent performance across all supported gesture categories under real-time conditions. The evaluation metrics, including accuracy, precision, recall, and F1 score, indicate that the proposed rule-based gesture recognition approach is effective for practical human–computer interaction tasks. The system achieved stable performance at approximately 28–35 frames per second on CPU-based hardware without requiring any training dataset or specialized sensors. Minor variations in performance were observed due to lighting conditions and hand positioning; however, overall system robustness and responsiveness remained high.

V. CONCLUSION

In this study, an AI-based virtual mouse and augmented reality interface system was implemented and evaluated to enable contactless human-computer interaction through real-time hand gesture recognition using a standard webcam. A deterministic gesture recognition engine maps seven distinct finger configurations to full mouse-control actions. A dual-hand state machine enables simultaneous cursor control and AR menu access. A QWERTY virtual keyboard rendered as an augmented reality overlay on the live camera frame enables text entry without any physical input device. All system events including cursor movement, mouse clicks, scroll, screenshots, application shortcuts, and keystrokes are injected at the OS level without requiring modification of any existing application.

The system achieves end-to-end gesture response latency below 35 ms and 28–35 FPS throughput on commodity CPU hardware without any GPU or specialized sensors. Future work directions include temporal coordinate smoothing to reduce cursor jitter, a trained gesture classifier for an expanded vocabulary, cross-platform OS support for macOS and Linux, multi-threaded pipeline execution for improved throughput consistency, and user-calibrated gesture thresholds based on per-user hand geometry measurements.

REFERENCES

[1] Z. Cao, T. Simon, S. E. Wei, and Y. Sheikh, "Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2017, pp. 7291–7299, doi: 10.1109/CVPR.2017.143.

[2] F. Zhang, A. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C. L. Chang, and M. Grundmann, "MediaPipe Hands: On-Device Real-Time Hand Tracking," arXiv:2006.10214, 2020, doi: 10.48550/arXiv.2006.10214.

[3] S. Mitra and T. Acharya, "Gesture Recognition: A Survey," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 37, no. 3, pp. 311–324, May 2007, doi: 10.1109/TSMCC.2007.893280.

[4] G. R. S. Murthy and R. S. Jadon, "A Review of Vision Based Hand Gestures Recognition," International Journal of Information Technology and Knowledge Management, vol. 2, no. 2, pp. 405–410, 2009.

[5] R. Y. Wang and J. Popović, "Real-Time Hand-Tracking with a Color Glove," ACM Transactions on Graphics, vol. 28, no. 3, pp. 1–8, 2009, doi: 10.1145/1531326.1531369.

[6] C. Keskin, F. Kırac, Y. E. Kara, and L. Akarun, "Real-Time Hand Pose Estimation Using Depth Sensors," in Proc. IEEE Int. Conf. Computer Vision Workshops (ICCV Workshops), 2011, pp. 1228–1234, doi: 10.1109/ICCVW.2011.6130401.

[7] T. Starner and A. Pentland, "Real-Time American Sign Language Recognition from Video Using Hidden Markov Models," in Proc. Int. Symp. Computer Vision, 1995, pp. 265–270, doi: 10.1109/ISCV.1995.477012.

[8] V. Pavlovic, R. Sharma, and T. S. Huang, "Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7, pp. 677–695, Jul. 1997, doi: 10.1109/34.598226.

[9] J. Shotton et al., "Real-Time Human Pose Recognition in Parts from Single Depth Images," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2011, pp. 1297–1304, doi: 10.1109/CVPR.2011.5995316.

[10] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556, 2014, doi: 10.48550/arXiv.1409.1556.