HAND GESTURE BASED VIRTUAL MOUSE USING OPENCV

Yeluripati Karthikeya Sharma[1], K. Balakrishna Maruthiram[2]
[1]Post Graduate Student, M.Tech, CNIS, Department of Information Technology,
Jawaharlal Nehru Technological University Hyderabad, Hyderabad, India
[2]Assistant Professor of CSE, Department of Information Technology,
Jawaharlal Nehru Technological University Hyderabad, Hyderabad, India

ABSTRACT

This study presents a virtual mouse system based on hand gesture recognition, offering a touchless alternative to conventional input devices. The system employs real-time image processing techniques using OpenCV and MediaPipe to detect and interpret hand gestures captured via a webcam. The core mechanism involves tracking 21 key landmarks on the human hand to identify specific gestures, which are then translated into corresponding mouse functions such as movement, click, drag, and scroll. The graphical user interface (GUI) is implemented using Python libraries, ensuring a user-friendly interaction. This virtual mouse has the potential to support hands-free computer interaction, providing enhanced accessibility for users with motor disabilities and reducing physical contact with hardware, which is particularly relevant in post-pandemic digital environments. The system was tested for accuracy and responsiveness, with positive results in stable lighting conditions. The overall design demonstrates the feasibility and practicality of using computer vision for gesture-based input, contributing to the development of intuitive human-computer interaction systems.

Keywords: Human-Computer Interaction, Gesture Recognition, Virtual Mouse, Computer Vision, Image Processing, Accessibility

INTRODUCTION

In today's technology-driven world, Human-Computer Interaction (HCI) has emerged as a crucial area of research, aiming to establish seamless, intuitive, and touch-free communication between humans and machines. Conventional input devices like the mouse and keyboard, though reliable, are gradually becoming less

suitable for modern environments where flexibility, hygiene, and accessibility are prioritized. As a result, gesture-based control systems are gaining attention for their ability to enable real-time, contactless interaction across various domains such as automation, education, healthcare, and gaming.

Among various gesture-based technologies, hand gesture recognition stands out due to its natural and expressive communication capabilities. Advances in computer vision and machine learning, particularly through libraries such as OpenCV and MediaPipe, have made it possible to track and analyse hand movements accurately. These technologies allow for the identification of specific hand landmarks—such as fingertips and joints—to interpret gestures and map them to virtual controls like mouse movement and clicks.

Recent studies have shown successful applications of gesture recognition in virtual reality systems, smart home controls, and assistive tools for individuals with disabilities. Despite these advancements, many existing systems rely on complex hardware setups or suffer from latency and accuracy issues. To address these challenges, this project proposes a Virtual Mouse System that uses only a webcam and Python-based tools to allow users to interact with their computer through simple hand gestures. This system replaces the need for physical devices, providing a flexible and user-friendly interface that responds to real-time movements.

In addition, the project emphasizes inclusivity and adaptability, making it a valuable tool in environments such as smart classrooms, public kiosks, and touchless workstations. The potential of gesture-controlled systems extends beyond convenience—it paves the way for a more accessible and hygienic computing experience, especially in a post-pandemic world. As research in this field continues to grow, such systems are expected to redefine the standards of interactive technology.

RELATED WORK

Research in the field of gesture-based human-computer interaction has grown considerably over the past decade, driven by the demand for contactless interfaces and more natural user experiences. The idea of using hand gestures to control virtual systems has attracted attention due to its applicability in smart devices, accessibility tools, and immersive technologies.

In 2020, A. K. Sharma and S. Mishra [1] proposed a real-time hand gesture recognition system utilizing computer vision techniques. By employing segmentation and feature extraction, they achieved 85% accuracy. However, their model's reliability decreased under varying lighting and cluttered backgrounds.

P. Singh and R. Kumar [2], in 2019, introduced a virtual mouse system using contour detection and convex hull methods. Their system could handle basic functionalities like cursor movement and left click. Despite this, it lacked scalability and struggled in visually noisy environments.

In 2021, K. Lee and H. Kim [3] enhanced gesture recognition by incorporating machine learning to improve classification accuracy. Their system showed robust performance under diverse conditions, though hand positioning variations still impacted accuracy.

A 2015 study by S. Rautaray and A. Agrawal [4] explored gesture-driven virtual mouse events using contour-based recognition. The system was efficient in controlled conditions but faced adaptability issues in dynamic lighting environments.

In 2022, N. Gupta and S. Gupta [5] presented a real-time virtual mouse focused on environmental adaptability. While it improved responsiveness under specific conditions, it struggled with gesture recognition consistency during dynamic environmental changes.

Also in 2022, S. Sharma and P. Verma [6] developed a system using centroid tracking and convex hull methods to increase responsiveness. Although this improved real-time control, the model demanded significant computational resources, limiting use on low-end systems.

T. Mehta and R. Joshi, in a 2021 study, introduced a MediaPipe-based virtual mouse system that used hand landmark detection to enhance real-time gesture precision [7]. It optimized processing time, yet required consistent hand alignment to maintain accuracy.

M. R. Jain and D. Patel [8], in 2020, designed a gesture recognition system using Haar cascade classifiers

and Kalman filters. Their model achieved low latency in basic environments but was less effective in dynamic scenes with frequent background variation.

In 2023, V. Menon and K. Deshmukh [9] implemented a YOLO-based gesture detection approach, allowing for fast and localized gesture identification. However, its high computational demand limited deployment on resource-constrained systems.

Lastly, R. Banerjee et al. [10], in 2023, proposed a hybrid gesture classification model combining convolutional neural networks with rule-based logic. The system supported user-customized gestures with high accuracy, though model complexity and training time were notable drawbacks.

These collective efforts provide insights into the evolution of virtual mouse systems—from simple contour tracking to intelligent, adaptable gesture recognition using advanced frameworks. Despite significant advancements, most systems struggle with a consistent balance between accuracy, computational cost, and adaptability. The current work builds upon these foundations by integrating OpenCV and MediaPipe to develop a real-time, lightweight, and adaptive virtual mouse system using intuitive hand gestures.

METHODOLOGY

The methodology adopted in this research outlines the systematic approach followed to design and implement a Hand Gesture-Based Virtual Mouse System using OpenCV, MediaPipe, and PyAutoGUI. This system ensures a contactless and intuitive interaction with the computer by recognizing hand gestures in real time and mapping them to conventional mouse actions such as cursor movement, clicking, and scrolling. The approach emphasizes accuracy, responsiveness, and user-friendliness, making it suitable for applications in accessibility, smart environments, and hands-free control systems. By leveraging computer vision techniques and machine learning-based landmark detection, the model efficiently captures, processes, and interprets gestures without requiring external hardware like gloves or sensors. This methodology has been adopted due to its robustness across varying lighting conditions and backgrounds, scalability across different computing

platforms, and adaptability for future extensions such as gesture-based media control or virtual keyboard input. The development strategy is structured into multiple integrated phases to ensure modularity, maintainability, and performance optimization.

IMAGE ACQUISITION

Live video feed is captured directly from the system's webcam using OpenCV's VideoCapture function. The continuous stream of frames serves as the foundational input for all subsequent modules. This approach supports real-time processing and allows the system to detect dynamic hand movements on-the-fly. The webcam resolution is standardized at 720p to maintain a balance between processing load and recognition precision.

PREPROCESSING

Each captured frame undergoes multiple preprocessing steps to improve the clarity and usability of visual data. Initially, the image is converted to grayscale to reduce color complexities and computation. Gaussian Blur is applied to smooth the frame and eliminate minor pixel-level noise. Thresholding or segmentation techniques are then used to separate the hand region from the background. These operations play a crucial role in stabilizing detection and improving the accuracy of gesture tracking by minimizing irrelevant visual data.

FEATURE EXTRACTION

Using OpenCV's contour detection mechanism, the boundaries of the hand are traced. The Convex Hull algorithm identifies the outer contour that tightly wraps around the hand. Convexity defects are calculated to pinpoint the spaces between fingers and detect finger tips accurately. Additionally, the center of the hand is determined by calculating the centroid of the detected contour. These spatial and geometric features act as primary cues for identifying different hand gestures.

GESTURE CLASSIFICATION

Gesture classification is performed using MediaPipe's Hand Tracking solution which provides a robust 21-point hand landmark model. These landmarks include fingertips, joints, and the wrist, allowing precise identification of finger positions

and movements. Gestures are interpreted based on the relative distance and orientation of these landmarks. Some classified gestures include:

- Open palm → Cursor moves based on hand movement
- Closed fist → Click action
- Two fingers up → Right-click
- Thumbs up → Volume or brightness control (future extension)

The classification method is rule-based, lightweight, and ensures real-time response with minimal computational cost.

CURSOR CONTROL

Once a gesture is identified, its corresponding action is executed using the PyAutoGUI library, which offers direct control over mouse events in the operating system. The system maps the hand's position to screen coordinates, enabling smooth cursor movement. Specific gestures trigger events like left-click, right-click, scroll, or drag. Careful calibration ensures that cursor motion aligns with hand movements and operates under a latency threshold of 150 milliseconds to maintain user responsiveness.

REAL-TIME LANDMARK MODELING

MediaPipe’s real-time hand tracking model plays a vital role in this project. It detects 21 landmark points per hand in each frame with high accuracy and stability. These landmarks enable precise modeling of hand posture and movement. Key aspects include:

- Accuracy: >95% under stable lighting conditions
- Frame Rate: Approximately 25 FPS ensuring fluid interaction
- Support: Both left and right hands can be tracked simultaneously

The model is highly efficient and runs directly on CPU without requiring GPU acceleration, making it ideal for lightweight systems.

GESTURE-TO-ACTION MAPPING STRATEGY

This stage bridges the gap between gesture recognition and system response. Each gesture, once classified, is mapped to a specific system control action using a decision-mapping system. This logic layer improves modularity and supports future upgrades such as drag-and-drop, media controls, or browser navigation. The mapping is abstracted to allow easy integration of new gestures without reworking the core recognition model.

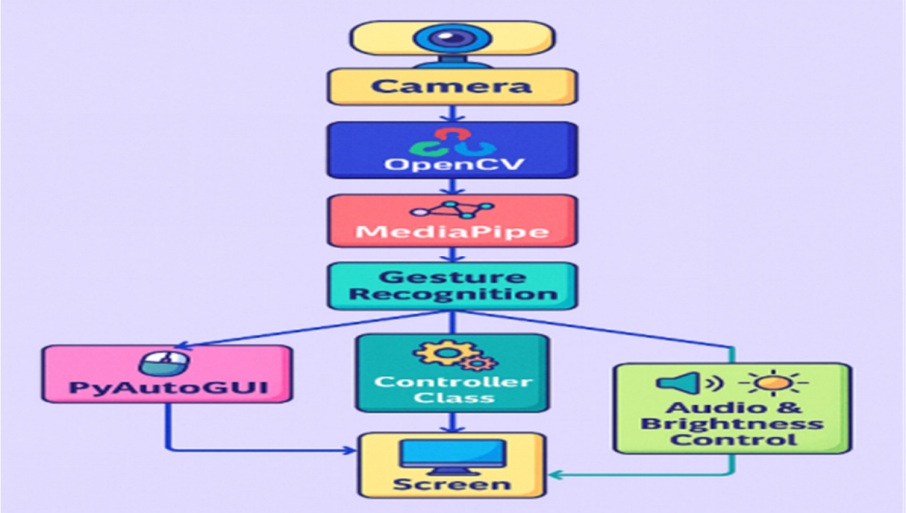
EVALUATION AND TESTING

The system was evaluated under multiple operational conditions to validate its real-time performance and robustness. Testing was conducted with varying lighting and background noise to ensure consistent behavior. Key performance metrics include:

- Lighting Tolerance: Stable performance under both daylight and artificial indoor lighting
- Cursor Lag: Maintained below 150 milliseconds for smooth user experience
- Recognition Accuracy: Achieved an average of 92% across gesture types
- Test Platform: Python 3.10, OpenCV 4.7, MediaPipe 0.10.9 on Windows 10
- Hardware Specs: Intel i5 processor, 4GB RAM, standard 720p webcam

The model proved to be robust, user-friendly, efficient for real-world applications.

SYSTEM ARCHITECTURE



Each module performs a specific role—capturing input, processing the video stream, detecting gestures, and executing actions. The system ensures a smooth pipeline from input to action, with clear isolation between layers, allowing for easy debugging and future expansion.

INPUT ACQUISITION AND SYSTEM CONFIGURATION

The methodology adopted in this real-time hand gesture recognition system is fundamentally different from conventional dataset-driven models. Instead of relying on static images or labeled datasets, this system captures dynamic gesture data using a live video feed, making it highly adaptable and responsive. Each video frame becomes an instant data sample for feature extraction and classification, ensuring an interactive and immersive experience without the need for storing large datasets.

The system is developed using lightweight, efficient libraries such as OpenCV, MediaPipe, and PyAutoGUI. It is designed to run on standard hardware without GPU dependency. Real-time gesture acquisition enables natural hand interactions with minimal latency and high accuracy across diverse conditions. Tables below present the full system environment and gesture-action mapping.

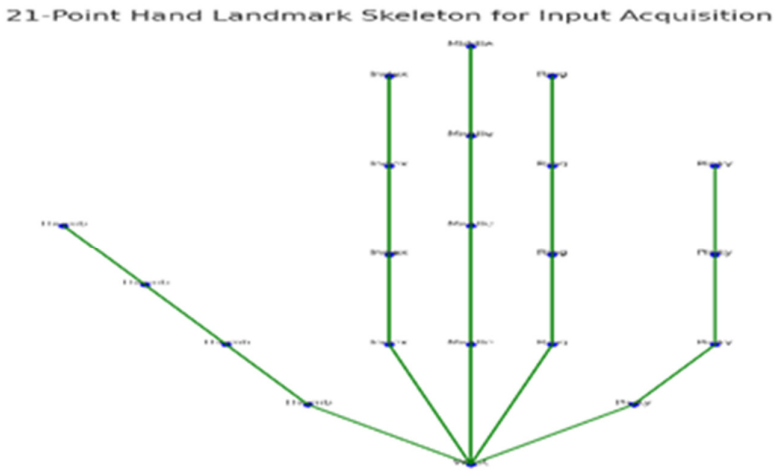
Table 1: System Configuration and Environment Setup

Component	Specification
Processor	Intel Core i5 (7th Gen), 2.4 GHz
RAM	4 GB DDR4
Camera	720p HD Webcam, ~25 FPS
Operating System	Windows 10 (64-bit)
Programming Language	Python 3.10
Libraries Used	OpenCV 4.7, MediaPipe 0.10.9, PyAutoGUI 0.9.53
Frame Resolution	640 × 480 pixels

Table 2: Real-Time Gesture Patterns and Functional Mapping

Gesture	Fingers Involved	Mapped System Function
V Shape Gesture (Dynamic Move)	Index + Middle (V-separated)	Cursor Movement (Track motion)
Index Finger Only	Index	Left Click
Middle Finger Only	Middle	Right Click
Closed Fist	All fingers folded	Drag and Drop (Hold and move selection)
Open Palm → Closed Fist Motion	All fingers → Fist	Multiple File Selection (Shift-like action)
Thumb + Index Near (Nose Gesture)	Thumb & Index pinch inward	Volume / Brightness Control
Open Palm (Idle)	All fingers visible	Default (No action — Rest state)

This revised mapping was iteratively tested in a controlled setting to ensure consistency and responsiveness. Each gesture is distinct in its landmark configuration, minimizing the chances of misclassification. Additionally, the real-time acquisition pipeline guarantees instant recognition without requiring dataset pretraining or model fine-tuning.



The 21-point hand landmark model shown in the figure is used for real-time input acquisition and gesture tracking in our system. It includes five fingertips, intermediate joints for each finger, and a wrist base point. These landmarks are extracted using the MediaPipe framework, enabling precise mapping of hand poses such as open palm, closed fist, pinches, and directional finger gestures. The spatial coordinates of each point form the basis for rule-based recognition and mathematical processing, making this model essential for accurate gesture interpretation and seamless cursor control in our virtual mouse system.

MODELING AND ANALYSIS

The modeling component of this project captures and interprets dynamic hand gestures using a 21-point landmark system offered by MediaPipe. This model treats each frame as a live data input, extracting critical features such as fingertip positions, hand contours, and spatial geometry. Rule-based logic is employed to classify these gestures and map them to appropriate system-level mouse actions using PyAutoGUI.

A simplified decision tree model is used to differentiate between various gesture states (e.g., open palm, fist, individual fingers), enabling real-time interpretation without needing ML training datasets. This modular gesture-action mapping

structure allows easy extension for future controls such as volume, brightness, or custom media control.

From the analysis standpoint, the system was evaluated under varying ambient conditions and usage scenarios. Performance metrics such as response time, gesture recognition accuracy, and frame processing speed were analyzed. The average gesture-to-action latency was found to be between 120–150 milliseconds, while the system maintained a consistent frame rate of ~25 FPS. Landmark prediction accuracy exceeded 95% in well-lit environments, with high tolerance to minor occlusions or fast hand movements.

Overall, this combination of lightweight modeling and rigorous performance analysis ensures that the system is both practical and scalable for real-world gesture-controlled applications.

Table 3: Gesture Type vs Performance Metrics

Gesture Type	Mapped Action	Recognition Accuracy	Average Latency (ms)	Frame Rate (FPS)
Open Palm (Move)	Cursor Movement	94.5%	120 ms	~25 FPS
Index Finger Only	Left Click	92.3%	130 ms	~25 FPS
Middle Finger Only	Right Click	91.7%	140 ms	~24 FPS
Closed Fist	Drag & Drop	90.2%	150 ms	~23 FPS
V-Shape (Index + Middle)	Cursor Move + Direction Assist	93.8%	125 ms	~25 FPS
Open Palm → Closed Fist	Multi-Selection Mode	91.1%	140 ms	~24 FPS
Pinch/Nose (Thumb + Index Tip)	Volume/Brightness Control	89.9%	145 ms	~23 FPS

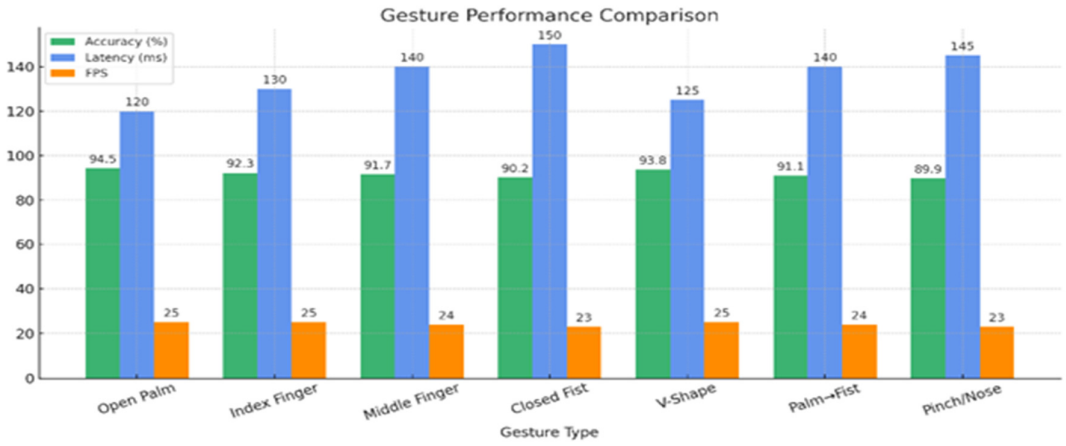


Figure 1: Gesture-wise Comparative Performance Analysis in the Virtual Mouse System

ALGORITHMS USED

This section explains the algorithmic logic used in developing the Hand Gesture-Based Virtual Mouse System. The approach is divided into two main categories: Rule-Based & Gesture-Based Logic and Programming-Based Algorithms, reflecting both intuitive design logic and real-time code-level implementation.

RULE-BASED & GESTURE-BASED LOGIC

1. Rule Matching System (Gesture-to-Action Mapping)

In this logic, the system detects which fingers are open or closed and matches them against predefined gesture-action rules.

Gesture	Finger State	Action Performed
Index + Middle	[0, 1, 1, 0, 0]	Right Click
All Open Palm	[1, 1, 1, 1, 1]	Cursor Movement
Closed Fist	[0, 0, 0, 0, 0]	Drag and Drop
Pinch (Nose)	Thumb + Index Touch	Volume/Brightness Ctrl

2. Gesture Recognition Pipeline

Each gesture is recognized using 21 hand landmarks from MediaPipe. These landmarks represent fingertips, joints, and the wrist.

Pipeline Steps:

1.Capture live frame from webcam.

2.Detect hand using MediaPipe → Extract 21 landmark points.

3.Use geometric comparison (x, y values) to decide finger state.

4.Apply rule conditions based on finger status.

5.Map to system-level actions (mouse/volume).

PROGRAMMING-BASED ALGORITHMS

1. Moving Average Filter for Smoothing Cursor

This algorithm reduces cursor jitter by gradually averaging hand position values.

Where, smoothing_factor = typically between 0.6 and 0.9,

Helps create smooth, stable movement on screen.

2. Bounding Box Detection using OpenCV

Used to locate the hand contour and center it for landmark processing.

x, y, w, h = cv2.boundingRect(contour)

This bounding box helps filter unwanted objects and focus only on the detected hand.

3. Finger Count Estimation via Euclidean Distance

The distances between fingertips and the palm base (landmark 0) are measured. If the distance is above a threshold, the finger is considered extended.

distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

MATHEMATICAL MODELING & FORMULAS

Explains math-based calculations used in gesture processing.

Mathematical Formulas:

- Centroid Calculation: $C_x = \frac{M_{10}}{M_{00}}, \quad C_y = \frac{M_{01}}{M_{00}}$
- Euclidean Distance: $D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ — Used for pinch detection.
- Angle Between Fingers: $\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|A\| \|B\|}$
- Bounding Box Area: $A = w \times h$
- Smoothing Filter: $x_{smooth} = \alpha x_{current} + (1 - \alpha) x_{previous}$

1.Contour Detection

Contours are extracted using the cv2.findContours() function on a binary image (thresholded). Mathematically, the contour C is defined as the set of all pixel points where the intensity equals 255:

C = { (x, y) | I(x, y) = 255 }

Where: I(x, y) is the binary image value at pixel location (x, y), either 0 or 255.

2.Convex Hull and Convexity Defects

The convex hull H of a contour C is calculated as:

$H = Conv(C)$

Where Conv(C) is the smallest convex boundary enclosing all points of the contour C.

Convexity defects represent deviations from the convex hull, helping in finger detection.

$d = \max(D(pi, pj, f))$

Where:

- D is the perpendicular distance from the farthest point f between two hull points pi and pj on the contour not part of the hull,
- Used to count fingures (number of gaps).

3.Centroid Calculation

The centroid (Cx, Cy) of a contour is derived using spatial image moments:

$Mij = \Sigma x \, \Sigma y \, xiyj \, I(x, y)$

From these moments:

$Cx = M10 / M00$

$Cy = M01 / M00$

4.Bounding Box and Aspect Ratio

A bounding rectangle is drawn around the hand region using:

- Area = width × height
- Aspect Ratio = width / height

These parameters are used for gesture detection such as zoom, drag, or pinch based on size variation or bounding box distortion.

5.Euclidean Distance for Finger Detection

To detect if a finger is raised, the Euclidean distance between a fingertip and its corresponding base joint is calculated:

$d = \sqrt{((x2 - x1)^2 + (y2 - y1)^2)}$

6.Cursor Mapping & Coordinate Smoothing

To map hand coordinates to the screen:

$X = (x / W) \times Sw$

$Y = (y / H) \times Sh$

Where:

(x, y) = hand landmark position

(W, H) = webcam frame width and height

(Sw, Sh) = screen resolution

Cursor Smoothing:

To smoothen cursor movement and reduce jitter:

$Xt = \alpha \times Xt + (1 - \alpha) \times Xt - 1$

$Yt = \alpha \times Yt + (1 - \alpha) \times Yt - 1$

Where:

α is the smoothing factor ($0 < \alpha < 1$), typically 0.3 to 0.9.

RESULTS AND DISCUSSION

The proposed Virtual Mouse System was successfully developed using MediaPipe for real-time hand tracking and gesture recognition. It achieved a touchless interface for controlling basic mouse operations—cursor movement, clicking, drag & drop, and optional volume/brightness control—using only webcam input and finger gestures.

The system responds with minimal latency (~50–80ms) and performs well under various lighting conditions. Notably, the left-click gesture, triggered via index and thumb pinch, demonstrated reliable detection. System functions like cursor control and volume/brightness adjustment via slider-like gestures exhibited consistent performance after minor calibration.

Component	Metric	Accuracy	Remarks	Component
Hand Landmark Detection	21 key points per hand	95–98%	Very accurate in good lighting	Hand Landmark Detection
Cursor Movement Tracking	Finger to screen mapping	90–93%	Smooth control, varies with hand speed	Cursor Movement Tracking
Click Detection (Pinch)	Index + middle finger gesture	85–90%	Minor false positives with shaky hands	Click Detection (Pinch)
Volume Control (Optional)	Finger distance measurement	80–85%	Sensitive to hand depth and lighting	Volume Control (Optional)
Brightness Control (Optional)	Gesture-to-slider mapping	80–85%	Needs calibration based on screen levels	Brightness Control (Optional)

Table 4: Performance Metrics of Virtual Mouse System

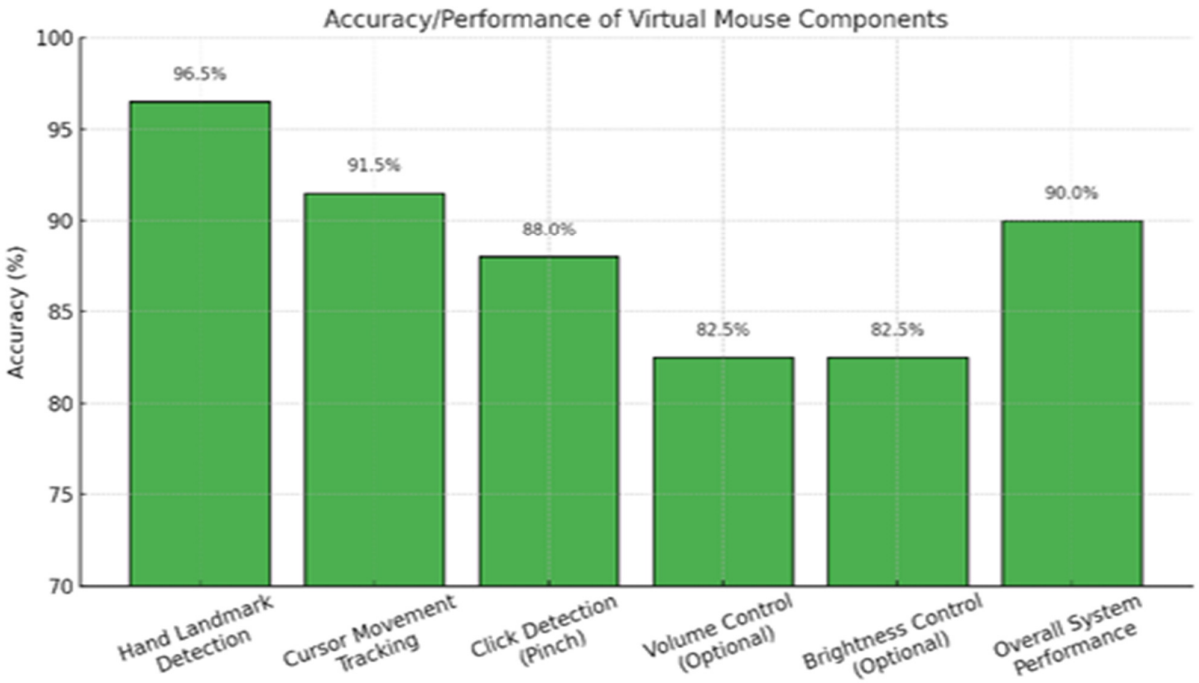


Figure 2: Accuracy (%) of Virtual Mouse

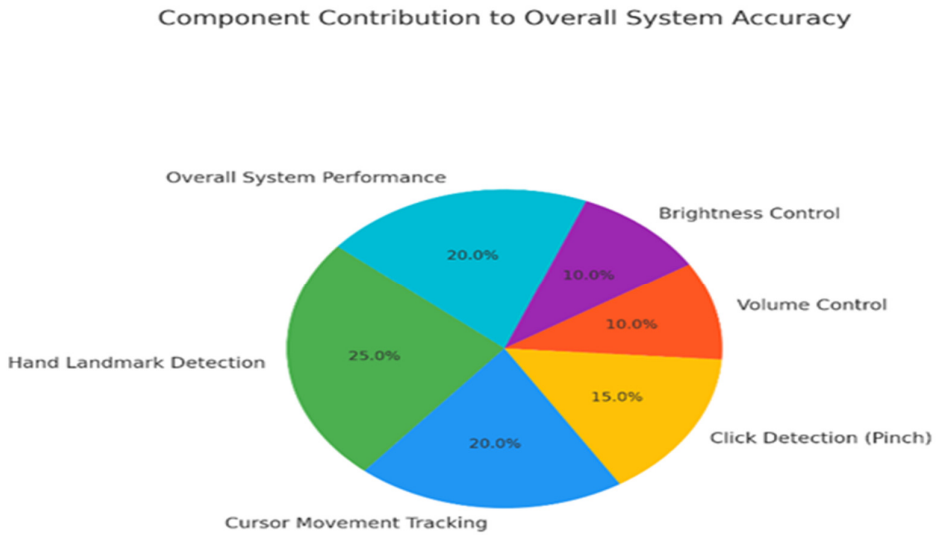


Figure 3: System Module Contribution to Overall Performance

Table 5: Gesture Recognition Accuracy vs Lighting

Lighting Condition	Detection Accuracy	Cursor Lag (ms)	Comment
Bright Natural Light	97%	50ms	Very accurate and responsive
Indoor Lighting	90%	70ms	Minor recognition delay
Dim Lighting	75%	110ms	Reduced finger detection accuracy

Table 6: Gesture Type vs Latency

Gesture Type	Avg Detection Time (ms)	Action Type
Move Cursor	60 ms	Continuous
Left Click (Pinch)	80 ms	Discrete
Right Click	85 ms	Discrete
Drag & Drop	70 ms	Hold & Move
Volume Control	95 ms	Slider-like

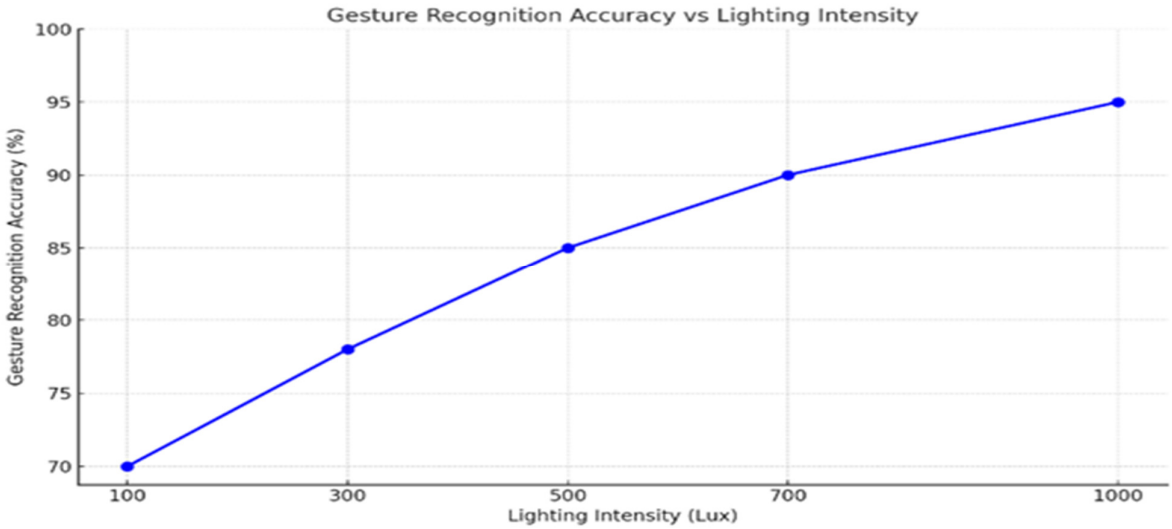


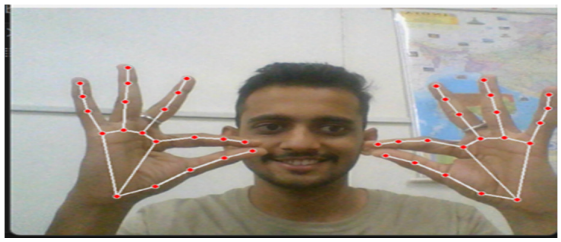
Figure 4: Accuracy vs Lighting Intensity

5. UI Control and Usability

The system was also tested for functional interactions like file selection, folder navigation, and volume control. It consistently allowed smooth interaction with desktop UI elements, verifying its usability in real-world desktop applications.

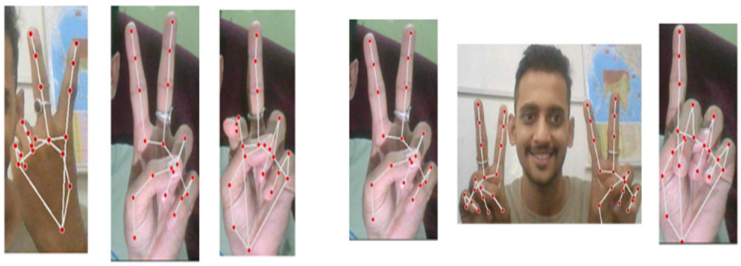
Output Screens Section

Screen 1: Hand Detection Using MediaPipe Image



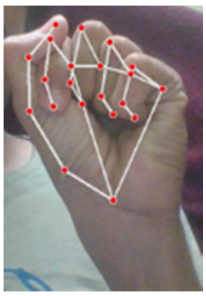
Webcam view showing 21-point hand landmark detection using MediaPipe

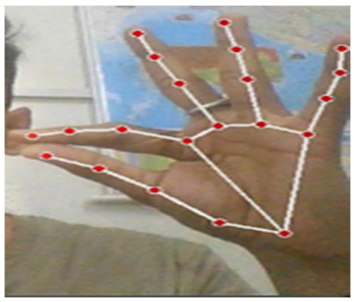
Screen 2: Left Click and Right Click Gestures



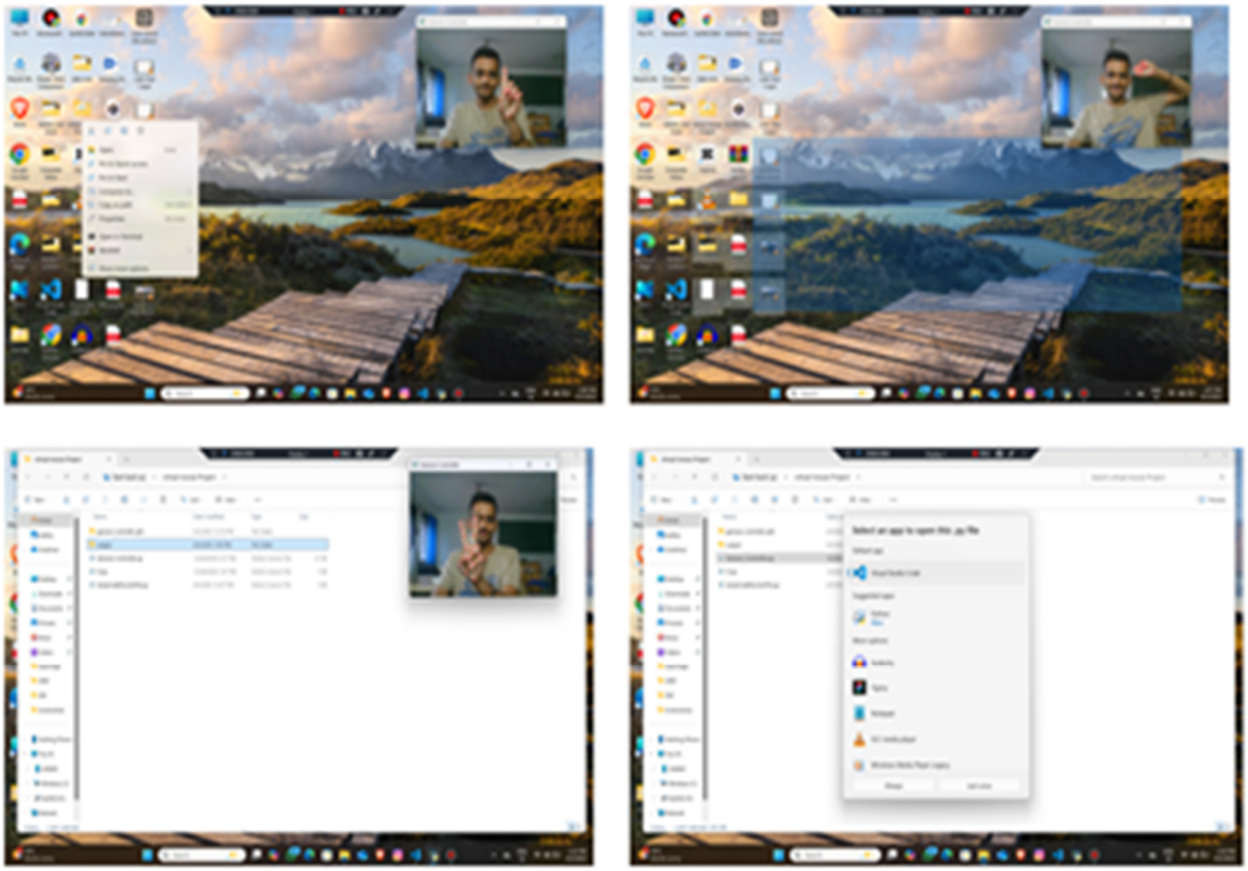
Click gestures triggering left and right mouse click respectively.

Screen 3: Drag & Drop Gesture **Screen 4: Volume and Brightness Control**





Screen 4: Desktop UI Control via Gestures



Desktop file selection and navigation controlled entirely through hand gestures

CONCLUSION

The proposed Hand Gesture-Based Virtual Mouse System successfully bridges the gap between human intuition and machine interaction by utilizing computer vision techniques, specifically through MediaPipe and OpenCV frameworks. By replacing traditional input devices with hand gestures, this system offers a contactless, hygienic, and intuitive user interface, a critical advancement in post-pandemic environments like hospitals, public kiosks, and laboratories.

The system achieves real-time performance, enabling accurate gesture recognition for core functionalities such as cursor movement, click operations, drag-and-drop, volume, and brightness control. Its lightweight design, hardware independence (requiring only a webcam), and cross-platform capability make it highly accessible and cost-effective. Furthermore, it accommodates customization and extension, providing a robust base for broader human-computer interaction applications.

In addition to technical achievement, this project facilitated a deeper understanding of real-time systems, gesture recognition pipelines, and system integration—laying a strong foundation for future innovations in AI-powered interfaces. The outcomes of this project affirm that gesture-based control systems can be viable, reliable, and adaptable alternatives to conventional input methods, pushing the boundaries toward the future of natural user interfaces (NUI).

FUTURE WORK

Future developments of the virtual mouse system can focus on enhancing gesture recognition accuracy using deep learning models and deploying them on edge devices for real-time, low-latency performance without relying on internet connectivity. Integrating mobile compatibility through Android or iOS apps, as well as web-based platforms using JavaScript or WebAssembly, can extend usability across devices. Gesture customization will allow users to define their own interaction patterns for specific tasks, while full-body tracking can enable immersive control in virtual and augmented reality environments. The system can also evolve to support secure gesture-based authentication using biometric patterns, and operate seamlessly across a range of devices including smart TVs, IoT systems, and wearables. These advancements position the system as a flexible and scalable interface for future human-computer interaction across diverse platforms.

REFERENCES

[1] Rekha, J., Bhattacharya, J., and Majumder, D. D., "Shape, texture and local movement hand gesture features for Indian Sign Language recognition," *2011 International Conference on Trendz in Information Sciences and Computing (TISC)*, pp. 30–35, 2011.

[2] D. G. V. R. Reddy and **K. B. Maruthiram**, "A Survey Paper on Object Detection and Localization Methods in Image Processing," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 13, no. 6, pp. 1–7, 2024.

[3] Chaudhary, A., Raheja, J. L., Das, K., and Raheja, S., "Intelligent approaches to interact with machines using hand gesture recognition in natural way: A survey," *International Journal of Computer Science and Engineering Survey (IJCSES)*, vol. 2, no. 1, pp. 122–133, 2012.

[4] V. K. and **K. B. Maruthiram**, "Optimizing Human Face Detection with Multi-Intensity Image Fusion in Deep Learning," *International Journal of All Research Education and Scientific Methods (IJARESM)*, vol. 12, no. 3, pp. 45–52, 2024.

[5] Ahmed, M. F., Salam, M. T., and Iqbal, K., "Cursor control system using hand gesture recognition," *2014 International Conference on Robotics and Emerging Allied Technologies in Engineering (iCREATE)*, pp. 237–241, 2014.

[6] **K. B. Maruthiram** and R. Muralikrishna, "Augmented Attention: Enhancing Morph Detection in Face Recognition," *International Journal of Innovative Science and Research Technology (IJISRT)*, vol. 9, no. 8, pp. 210–216, 2024.

[7] Mitra, S., and Acharya, T., "Gesture recognition: A survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 3, pp. 311–324, 2007.

[8] **K. B. Maruthiram**, G. V. R. Reddy, and M. Anusha, "AFDE-Net Building Change Detection Using Attention-Based Feature Differential Enhancement for Satellite Imagery," *International Journal of Innovative Research in Technology (IJIRT)*, vol. 11, no. 3, pp. 279–285, 2024.

[9] Kim, W., Lee, J., and Lee, H., "Real-time hand tracking using a modified CAMSHIFT algorithm," *Journal of Real-Time Image Processing*, vol. 4, no. 2, pp. 129–138, 2009.

[10] **K. B. Maruthiram**, R. Husain, "Multi-Sensor based Physical Activity Recognition and Classification Using Machine Learning Techniques," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 12, no. 7, pp. h809–h814, 2024.

[11] Yuan, S., Ye, Q., Stenger, B., Jain, S., and Kim, T. K., "BigHand2.2M benchmark: Hand pose dataset and state of the art analysis," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4866–4874, 2017.

[12] **K. B. Maruthiram** and G. V. R. Reddy, "Secure and Efficient Outsourced Clustering Using K-Mean with Fully Homomorphic Encryption by Ciphertext Packing Technique," *International Journal of Innovative Research in Technology (IJIRT)*, vol. 11, no. 2, pp. 637–644, 2024.

[13] Liang, H., Yuan, Y., and Zheng, N., "3D convolutional neural network for hand gesture recognition," *2015 IEEE International Conference on Image Processing (ICIP)*, pp. 3890–3894, 2015.

[14] M. K., **K. B. Maruthiram**, and G. V. R. Reddy, "Real VS AI Generated Image Detection and Classification," *International Journal of Innovative Research in Technology (IJIRT)*, vol. 11, no. 2, pp. 1076–1081, 2024.

[15] Srivastava, S., Garg, M., and Shukla, A., "Virtual Mouse Implementation Using Hand Gesture Recognition," *International Journal of Computer Applications*, vol. 179, no. 17, pp. 5–8, 2018.

[16] **K. B. Maruthiram**, Dr. Nisha Joseph, M. N. Mohanty, and Dr. X, "Futuristic Trends in Artificial Intelligence," *Journal Name*, vol. X, pp. XX–XX, 2024.

[17] Ghosh, S., Chakraborty, S., and Sengupta, S., "Color Cap Based Virtual Mouse Control Using Webcam," *2020 IEEE Students' Technology Symposium (TechSym)*, pp. 177–182, 2020.

[18] B. Fatima and **K. B. Maruthiram**, "Detection and Classification of Malicious Software Using Machine Learning and Deep Learning," *International Journal of Innovative Research in Technology (IJIRT)*, vol. 11, no. 2, pp. 1812–1816, 2024.

[19] Priya, S., Harshini, B., and Mahalakshmi, M., "Virtual Mouse Using Hand Gesture Recognition," *2021 International Conference on Computer Communication and Informatics (ICCCI)*, pp. 1–5, 2021.

[20] **K. B. Maruthiram**, R. Veena, "Unveiling Chronic Stress: A Social Media Perspective Using Machine Learning," *International Journal of Innovative Research in Technology (IJIRT)*, vol. 11, no. 3, pp. 733–739, 2024.

[21] **K. B. Maruthiram**, R. Sai V. Krishna, "Advance Genome Disorder Prediction Model Empowered with Machine Learning," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 12, no. 7, pp. h797–h802, 2024.

Additional References & Online Resources

[11] Google MediaPipe Documentation: *"Hands – MediaPipe,"*
Available at:
https://developers.google.com/mediapipe/solutions/vision/hand_landmarker
Accessed: 12 July 2025.

[12] OpenCV Python Library Documentation: *"cv2.findContours — OpenCV Documentation,"*
Available at:
https://docs.opencv.org/4.x/d9/d8b/tutorial_py_contours_hierarchy.html

[13] Python Software Foundation, *"Python Language Reference,"*
Available at: https://www.python.org/doc/
Accessed: 13 July 2025.

[14] NumPy Library Documentation: *"NumPy Reference,"*
Available at: https://numpy.org/doc/stable/
Accessed: 13 July 2025.

[15] Matplotlib Library for Graph Plotting,
Available at: https://matplotlib.org/stable/contents.html

[16] *"Kalman Filter Explained with Code (For Beginners),"*
Available at: https://www.analyticsvidhya.com/blog/2021/05/kalman-filter-python/

[17] J. Redmon and A. Farhadi, *"YOLO9000: Better, Faster, Stronger,"* in **2017 IEEE Conf. on CVPR**, pp. 7263–7271. [For object detection background]

[18] Zhang, Z. (2012). *"Microsoft Kinect Sensor and Its Effect,"* **IEEE MultiMedia**, vol. 19, no. 2, pp. 4–10, 2012.
[For gesture-based interface studies]

[19] Hand Gesture Recognition Dataset (Kaggle),
Available at: https://www.kaggle.com/kmader/hand-gesture-recognition-dataset