Prashant Mishra, 2026, 14:5
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Gesture Controlled Virtual Mouse: A Review and Camera-Based System Design

Prashant Mishra[1], Yash Kumar[2], Yash Chaudhary[3], Ankit Kumar[4] Mr. Abhishek Chaudhary[5]
Department Of Computer Science And Engineering, Mahatma Gandhi Mission'S College Of Engineering And Technology, Noida, Uttar Pradesh, India
Email: Prashantmishra3535@Gmail.Com

Abstract— A conventional mouse is the standard tool for moving a pointer, but hand gestures offer another way to interact with a computer. The availability of low-cost webcams and efficient hand-tracking models has made this approach more practical. This paper reviews major methods for gesture-based mouse control, including wearable sensors, depth cameras, colour markers, skin segmentation, neural-network detectors and landmark tracking, and discusses their advantages and drawbacks. It also presents the analysis and specification of a webcam-based virtual mouse, covering use cases, data flow, processing stages, system states and feasibility. The proposed design uses a hand-landmark model to estimate 21 points in each frame. Rules based on finger positions identify gestures and translate them into cursor movement, clicking, scrolling and drag-and-drop operations. Key challenges include changing illumination, pointer jitter, unintended commands and arm fatigue; possible improvements are also discussed.

Index Terms— gesture recognition, virtual mouse, human–computer interaction, hand tracking, computer vision, OpenCV, MediaPipe.

I. INTRODUCTION

A teacher moving around a classroom may need to change slides without returning to a laptop, while a surgeon may need computer access without touching a keyboard during a procedure. In both situations, reaching a physical mouse can be inconvenient. Ordinary users also face limitations: a mouse requires a suitable surface, shared devices can raise hygiene concerns, and repeated use may be difficult for people with limited hand strength or mobility. These needs have encouraged research into controlling a computer pointer by presenting hand gestures to a camera.

Vision-based hand interaction has been studied for decades. Pavlovic and co-authors reviewed visual hand modelling and recognition for human–computer interaction in the 1990s [1], followed by broader surveys covering gesture recognition and vision-based methods [2], [3]. Wider access to inexpensive computing hardware has helped these techniques become more usable. Most laptops now include a webcam, and publicly available software libraries can track hands at video rates without requiring a dedicated graphics processor.

This paper reviews common approaches to gesture-controlled mouse systems and outlines the main limitations of each. It then specifies a webcam-based design that uses hand landmarks as the basis for recognition and control. The discussion proceeds through related work in Section II, a comparison of approaches in Section III, and the proposed system in Section IV, followed by its challenges, applications, future directions and conclusion.

II. BACKGROUND AND RELATED WORK

1. Static And Dynamic Gestures

Gestures can broadly be grouped into static and dynamic categories. A static gesture is identified from a hand pose, such as raising one finger or closing the hand into a fist. A dynamic gesture depends on movement over time, for example swiping, drawing a circle or moving a pinch. A virtual mouse benefits from both categories: the pose can indicate the intended command, while hand movement determines the pointer location or direction of scrolling.

© 2026 Prashant Mishra, This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited.

2. Wearable Sensors

Wearable devices measure hand activity directly. Gloves, inertial sensors and muscle-signal bands can capture finger bending, hand orientation or muscle activity without using a camera. Their measurements are therefore less affected by lighting or background conditions and may provide high accuracy. However, users must obtain and wear the equipment, calibrate it, and maintain its power supply. These requirements add cost and reduce the convenience of a hands-free system.

3. Depth And Dedicated Sensors

Depth cameras help distinguish a hand from its surroundings by providing distance information in addition to image colour. The Kinect brought depth sensing to many research settings [4], and Suarez and Murphy discussed how depth images can simplify hand segmentation [5]. Leap Motion is designed specifically for hand tracking; one controlled study reported sub-millimetre errors within its limited operating area [6]. Marin and colleagues also explored combining Leap Motion and Kinect data [7]. Despite these benefits, dedicated sensors cost extra, cover a restricted space and may be affected by strong sunlight.

4. Camera-Only Approaches

Camera-based systems using standard RGB cameras can be divided into several main approaches.Colour markers. In marker-based systems, users attach coloured tape or caps to their fingertips. Each video frame is converted to HSV colour space, the selected colours are isolated, and morphological filtering—often implemented with OpenCV—cleans the detected regions [8]. This approach is relatively simple and fast to implement. Its reliability can decline when illumination changes or when similar colours appear elsewhere in the scene.

Skin segmentation. This method identifies hand regions using colour thresholds, sometimes combined with background subtraction, so no fingertip markers are needed. The hand contour can then be analysed with convex hulls and the spaces between fingers to estimate finger positions. Haar-feature cascades have also been used for hand detection [9]. Results can be unreliable when the background, lighting conditions or skin appearance differs from what the method expects.

Learned detectors. Neural-network methods can handle more visual variation than simple colour rules. Models such as YOLO [10], SSD [11] and Faster R-CNN [12] can be trained to detect hands or classify gestures. Recurrent 3D convolutional networks have also been investigated for recognising motion-based gestures in video [13]. These approaches usually require labelled training examples and additional computing resources, particularly for larger models.

Landmark tracking. Landmark-based tracking is a practical option for many student projects because it provides useful hand geometry with limited training effort. MediaPipe includes a pipeline that detects a palm and estimates 21 hand landmarks from a colour image [14], [15]. Once these points are available, simple geometric rules can identify poses—for example, by comparing a fingertip with its middle joint or measuring the distance between the thumb and index fingertips.

III. COMPARATIVE ANALYSIS

Table I compares the main technique families. Each method involves trade-offs, so the best choice depends on the required accuracy, available hardware and project budget.

Comparison of approaches

For a system intended to run using only a laptop webcam, landmark tracking provides a practical balance of processing speed, accuracy and implementation effort. Applications that require very fine positional accuracy may still justify the cost of dedicated sensors.

A useful evaluation should consider several aspects of performance rather than relying on one accuracy figure:

- the proportion of gestures classified correctly;
- processing delay and the number of frames handled per second;
- pointer speed and target-selection accuracy, including tests informed by ISO 9241-9 and Fitts' law;
- the frequency of missed clicks and unintended clicks; and
- user comfort and reported arm fatigue.

Differences in cameras, lighting, tasks and test procedures make reported results hard to compare. A shared evaluation protocol for virtual mouse systems would make future results more meaningful.

TABLE I

Approach	Main advantages	Main limitations
Wearable sensors	Can be accurate; not affected by scene lighting	Higher cost; must be worn and calibrated
Depth / IR sensors	Provides 3D information; separates hand from background	Requires extra hardware; restricted range; sunlight may interfere
Colour markers	Fast and straightforward to implement	Markers are required; colour and lighting changes cause errors
Skin segmentation	No markers needed; relatively lightweight	Affected by background, illumination and skin-colour variation
CNN detectors	Can handle varied hand appearance	Requires labelled data, training and more computation
Landmark tracking	Real-time CPU operation; 21 landmarks; limited training	Hand must remain visible; jitter and limited rule-based gestures

IV. PROPOSED SYSTEM: ANALYSIS AND SPECIFICATION

The proposed virtual mouse uses a webcam as its input device. For each frame, the camera captures the hand, a tracking model estimates its landmarks, and a lightweight set of rules identifies the gesture. The recognised command is then sent to the operating system as a standard mouse event. The design is described through its functions, data, processing flow and behavioural states, followed by a feasibility assessment.

1. Functional Model

The user is the system's external actor. Figure 1 summarises the available interactions. The user initiates functions such as hand tracking, gesture recognition, scrolling, dragging and calibration by moving a hand in view of the camera. Once a valid gesture is detected, cursor movement and clicking are performed automatically. The main use cases are summarised in Table II.

Table II describes the main user-facing functions and the action that causes each one to run. The user first presents a hand to the webcam. Once tracking is stable, a pointing pose controls the cursor, fingertip pinches request clicks, finger movement requests scrolling, and a held fist is used for dragging. Calibration and a pause option provide a way to adjust or temporarily stop control.

Fig. 1. Use-case diagram of the virtual mouse. Dashed arrows show included use cases.

The information model follows the same sequence as the processing pipeline. A camera frame is converted into landmark coordinates; those coordinates are interpreted as finger states and a gesture label. The selected gesture is checked against the configured thresholds and calibration values before a mouse event is produced. This separation makes the stages easier to test and adjust independently.

The main input is the current webcam frame, while the main output is a conventional mouse event. Intermediate values include landmark coordinates, finger-extension flags and a gesture label. Calibration settings determine how the camera area is mapped to the display and how much pointer smoothing is applied.

A context-level view is useful because it makes the boundary of the application clear: the webcam supplies visual input, the operating system receives mouse

commands, and an optional visual overlay gives feedback to the user.

Table III describes the information handled at each stage, beginning with webcam pixels and ending with the mouse event delivered to the operating system.

The three data stores separate short-term tracking information from fixed recognition rules and user-specific settings. Keeping these items distinct means that landmark history can change every frame without altering the gesture definitions, while sensitivity and screen margins can be tuned without redesigning the recognition pipeline.

Only a recognised gesture should produce an operating-system command. This rule helps avoid sending mouse events when the hand is missing or when the pose does not match a supported action.

At the highest level, the application can be treated as a single process, as shown in Fig. 2. It receives video of the user's hand and sends cursor, click and scroll commands to the operating system. Visual feedback, such as an overlay showing detected landmarks, can help the user understand what the system is tracking.

The state-based description highlights how the application recovers from interruptions. Losing the hand should stop command generation and return the program to an idle condition; after the hand is detected again and tracking settles, recognition can resume. A dedicated pause control gives the user an explicit way to suspend interaction.

The gesture set intentionally uses a small number of easily distinguished poses. A pointing pose is reserved for cursor movement, two different fingertip pinches distinguish left and right clicks, vertical movement of two fingers controls scrolling, and a closed fist is assigned to drag-and-drop. A calibration gesture or keyboard shortcut is used to adjust or pause the system.

Figure 3 expands the application into five stages: acquire a video frame, detect the hand and landmarks, recognise the gesture, map the gesture to a mouse command, and execute that command. Three data stores support this flow. D1 keeps recent landmark coordinates, D2 contains gesture rules and thresholds, and D3 stores calibration and sensitivity values. The landmark data in D1 is used during gesture classification; the resulting label is then evaluated with the rules and settings in D2 and D3 to determine the action performed in the final stage.

Fig. 3. Level 1 data flow diagram.

For a camera frame, the horizontal screen position is calculated by removing the left margin, dividing by the usable camera width and multiplying by the screen width:

Figure 4 illustrates one cycle of the control loop. After initialisation, the program captures a frame and prepares it for hand tracking. If no hand is detected, processing continues with the next frame. When a hand is found, the system estimates which fingers are extended and

measures relevant landmark distances. It compares these features with the defined gesture rules. Frames that do not match a rule produce no mouse command; a recognised gesture is converted into an operating-system action. The loop continues until the user exits or disables the system.

The equivalent operation maps the vertical coordinate. The final values are limited to valid screen coordinates. This approach avoids using the outermost camera pixels, where reaching a display corner could otherwise require an uncomfortable arm movement.

Pointer smoothing should be adjustable because users may prefer different balances between precision and response speed. A simple incremental update moves the cursor a fraction of the distance towards the target coordinate each frame:

2. Behavioural Model

The system can also be described as a state machine, shown in Fig. 5. It starts in the Idle state and waits for a hand to appear. Depending on whether setup is required, the system either performs calibration or proceeds to gesture recognition after the landmarks become stable. A recognised gesture can activate cursor movement, clicking, scrolling or dragging. When the gesture ends, control returns to recognition. If the hand disappears from the camera view or the user disables control, the system resets through the hand-lost state to Idle.

The feasibility review suggests that the design can be implemented with standard student-project tools and does not depend on specialised sensing equipment. Its practical performance still needs to be measured on the intended computer and under different lighting conditions rather than assumed from the design alone.

The user interface should make the control state clear. Calibration, visible tracking feedback and an easy pause command can help users learn the gestures and prevent the virtual mouse from interfering when they want to use a conventional input device.

Because the camera and software are commonly available, the main project cost is implementation and testing time. An external webcam may be needed for a desktop computer, but dedicated depth hardware is not required by this design.

Table IV lists the gestures used by the design. The selected poses are based on familiar actions such as pointing, pinching the fingertips together and making a grip.

TABLE IV

Use case	Function and trigger
Track hand / detect landmarks	Finds the hand and estimates 21 landmarks when a hand enters the camera view.
Recognize gesture	Uses landmark positions and finger states to determine a pose; stable tracking is helpful.
Move cursor	Moves the pointer according to the index fingertip when the user points.
Left click	Issues a left-button click when the thumb and index fingertip are pinched together.
Right click	Issues a right-button click when the thumb and middle fingertip are pinched together.
Scroll	Moves the scroll wheel when the index and middle fingers move upward or downward.
Drag and drop	Keeps the button pressed while following the hand, then releases; activated by holding a fist.
Calibrate / enable-disable	Changes sensitivity and screen boundaries or pauses control, using a calibration gesture or keyboard shortcut.

V. GESTURE-TO-ACTION MAPPING

The camera image and the display generally have different dimensions. To map hand movement to the screen, the system uses an inner region of the camera frame, leaving a margin m around its edges, and scales that region to the full display. For camera coordinate xcam in a frame of width Wcam and screen width Wscr, the horizontal mapping is:

xscr = (xcam←m) / (Wcam←2m) × Wscr (1)

The same mapping is applied vertically, and coordinates are restricted to the screen boundaries. Limiting control to the inner camera region helps users reach screen edges without stretching their arms fully.

Smoothing and clicks. Landmark estimates can shift slightly between successive frames, causing visible pointer shake. Moving averages, exponential smoothing and Kalman filtering are common ways to reduce this effect [16]. A simple method updates the pointer by moving it partway from its previous position towards the current target:

xt = xt←1 + (xtarget←xt←1) / k (2)

The parameter k is the smoothing factor set during calibration. Increasing k reduces pointer shake but also slows the response, so a suitable value must balance steadiness and speed. Click detection also needs protection against repeated activation: checking fingertip distance and adding a short debounce interval can prevent one pinch from generating several clicks.

1. Feasibility

Technical feasibility. The design relies on widely available tools: a webcam, Python, OpenCV for image

handling, MediaPipe Hands for landmark estimation and a tool such as PyAutoGUI to issue mouse events. No specialised sensor is required. A typical mid-range computer may support around 20–30 frames per second, which can be sufficient for responsive pointer control.

Operational feasibility. The system should be straightforward to learn because its commands use familiar hand poses. Calibration and an on-screen landmark overlay can help users position their hand correctly. A toggle to pause gesture control also allows users to return to a physical mouse whenever needed.

Economic feasibility. The software components are free and open source, and many laptops already contain a webcam. There are generally no software licence or cloud-service charges; the main investment is development time, with an external webcam being an optional expense.

2. Challenges And Limitations

Lighting and background. Changes in brightness or a visually cluttered scene can reduce tracking quality, especially for approaches that depend on colour thresholds. Jitter and delay. Stronger smoothing produces steadier movement but can make the pointer less responsive. Accurate selection of very small targets can therefore remain difficult.

Unintended commands. The hand may form a recognised pose even when the user does not intend to issue a command, a problem often called the "Midas touch" effect. A neutral resting pose and a clear enable/disable control can help reduce unwanted actions.

Fatigue. Holding an arm up for extended periods can be tiring, so continuous gesture control may be better suited to shorter sessions.

Occlusion and distance. Tracking becomes less reliable when fingers overlap, the hand moves partly outside the image or the user is too far from the camera.

Fairness and privacy. A model evaluated on a limited range of hands may work less reliably across different skin tones and hand shapes. Camera privacy is another concern; processing frames locally and avoiding unnecessary storage can reduce risk.

Lack of tactile feedback. Unlike a physical mouse, an air gesture provides no physical sensation when a click occurs, so users must rely on visual or audio feedback.

VI. APPLICATIONS

Hands-free computer operation in hospitals and operating rooms, where limiting contact with shared equipment can be important. Slide control during presentations and lessons, allowing speakers to move around the room.

Public kiosks and information displays where reducing contact with shared surfaces is useful. Assistive computer access for people who have difficulty gripping or moving a conventional mouse. Interaction in games, augmented- and virtual-reality environments, and smart televisions.

VII. FUTURE SCOPE

Evaluate the system against a physical mouse using ISO 9241-9 pointing tasks and collect user feedback on comfort and usability.

Combine hand gestures with voice commands or eye-gaze input to reduce unintended actions and physical strain.

Use trained gesture classifiers, including models that learn motion over time, to support more gestures and user-defined commands.

Develop adaptive smoothing that provides stable control for precise selection while remaining responsive during rapid movement. Create lightweight models that run locally on low-power devices, improving accessibility and helping protect privacy.

Provide individual calibration for hand size, skin tone and seating distance, and integrate the system more closely with operating-system accessibility tools.

VIII. CONCLUSION

A gesture-controlled virtual mouse can provide a low-cost, touch-free alternative to a conventional mouse. The reviewed approaches suggest that landmark tracking with a standard webcam offers a useful

combination of speed, accessibility and implementation simplicity, while dedicated sensors remain valuable when precision is the main priority. The proposed specification describes the system from its use cases and data flow to its frame-processing loop and operating states, using readily available software and hardware. Important challenges remain, including illumination changes, pointer jitter, unintended activation and user fatigue. Shared benchmarks, adaptive filtering, improved gesture models and combinations with other input methods are promising areas for further work.

REFERENCES

1. V. I. Pavlovic, R. Sharma, and T. S. Huang, "Visual interpretation of hand gestures for human-computer interaction: A review," IEEE Trans. Pattern Anal. Mach. Intell., vol. 19, no. 7, pp. 677–695, 1997.
2. S. Mitra and T. Acharya, "Gesture recognition: A survey," IEEE Trans. Syst., Man, Cybern. C, Appl. Rev., vol. 37, no. 3, pp. 311–324, 2007.
3. S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: A survey," Artif. Intell. Rev., vol. 43, no. 1, pp. 1–54, 2015.
4. Z. Zhang, "Microsoft Kinect sensor and its effect," IEEE MultiMedia, vol. 19, no. 2, pp. 4–10, 2012.
5. J. Suarez and R. R. Murphy, "Hand gesture recognition with depth images: A review," in Proc. IEEE RO-MAN, 2012, pp. 411–417.
6. F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler, "Analysis of the accuracy and robustness of the Leap Motion Controller," Sensors, vol. 13, no. 5, pp. 6380–6393, 2013.
7. G. Marin, F. Dominio, and P. Zanuttigh, "Hand gesture recognition with Leap Motion and Kinect devices," in Proc. IEEE Int. Conf. Image Process. (ICIP), 2014, pp. 1565–1569.
8. G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.
9. P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Proc. IEEE CVPR, 2001, pp. I-511–I-518.
10. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proc. IEEE CVPR, 2016, pp. 779–788.
11. W. Liu et al., "SSD: Single shot multibox detector," in Proc. ECCV, 2016, pp. 21–37.
12. S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in Proc. NIPS, 2015, pp. 91–99.
13. P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz, "Online detection and classification of dynamic hand gestures with recurrent 3D convolutional neural networks," in Proc. IEEE CVPR, 2016, pp. 4207–4215.
14. C. Lugaresi et al., "MediaPipe: A framework for building perception pipelines," arXiv:1906.08172, 2019.
15. F. Zhang et al., "MediaPipe Hands: On-device real-time hand tracking," arXiv:2006.10214, 2020.
16. R. E. Kalman, "A new approach to linear filtering and prediction problems," J. Basic Eng., vol. 82, no. 1, pp. 35–45, 1960.