GESTURES CONTROLLED VIRTUAL MOUSE

Project Reference No.: *48S_BE_4544*

College: *B.L.D.E.A'S V.P DR P.G.Halakatti College Of Engineering And Technology, Vijayapura*
Branch: *Department Of Computer Science And Engineering*
Guide: *Prof. Santoshkumar.S. Dewar*
Student(S): *Ms. Sindhu Badiger*
Ms. Suma Borannawar
Ms. Sunayana Deshpande
Ms. Rakshita Patil

Introduction:

Traditional input devices such as the keyboard and mouse have been the primary tools for human-computer interaction for decades. However, as technology advances, alternative and more intuitive interaction methods have emerged. Gesture-based computing, which allows users to interact with computers using hand movements, is gaining popularity due to its natural, touch-free, and efficient operation. The Gesture-Controlled Virtual Mouse is an innovative project that enables users to control a computer's mouse functions using hand gestures, without physically touching a mouse or trackpad. By leveraging a webcam, computer vision, and machine learning, the system detects and interprets various hand gestures to perform mouse actions such as left click, right click, double click, scroll, and drag & drop. This project is developed using Python and powerful libraries like OpenCV (for image processing), Mediapipe (for real-time hand tracking), and PyAutoGUI (for controlling mouse actions). The system captures hand landmarks from the webcam feed, identifies specific finger positions or movements, and maps them to mouse commands on the screen. λ The main objective of this project is to provide a touchless, intuitive, and interactive way to control the computer, which can be useful in situations like: Presentations (hands-free control) λ λ λ λ Accessibility for people with limited mobility Gaming and interactive applications Reducing physical contact in public or shared computer setups. By combining computer vision and automation, the Gesture-Controlled Virtual Mouse showcases the potential of human-computer interaction through natural gestures.

Objectives:

1. Develop a touchless mouse control system: Create a system that allows users to control mouse functions using only hand gestures, eliminating the need for physical mouse devices.

2. Implement real-time gesture recognition: Use a webcam and computer vision techniques to detect hand gestures in real time with high accuracy and minimal delay.

3. Map gestures to mouse actions: Assign specific hand gestures to common mouse operations such as left click, right click, double click, scroll, and drag & drop.

4. Enhance human-computer interaction: Provide a natural, intuitive way for users to interact with the computer, improving usability and user experience.

5. Improve accessibility and usability: Offer an alternative control method for users with physical disabilities or limited mobility who may find using a traditional mouse difficult.

6. Minimize physical contact: Enable hands-free interaction with public or shared computers to reduce physical contact and improve hygiene.

Methodology:

The implementation of the Gesture-Controlled Virtual Mouse involves a structured, step- by-step approach that integrates computer vision, real-time video analysis, hand landmark detection, gesture recognition, and system-level automation. The primary goal is to enable users to control mouse functions—such as cursor movement, clicking, right-clicking, and scrolling—through intuitive hand gestures using only a webcam and open-source libraries.

1. Setting Up the Development Environment

The development was carried out using Python, due to its rich ecosystem of libraries for computer vision, machine learning, and GUI automation. The following tools and

libraries were used:

 OpenCV: For capturing real-time video from the webcam and image preprocessing.

 MediaPipe: A framework by Google that provides fast and reliable detection of 21

hand landmarks.

 PyAutoGUI: A cross-platform GUI automation library used to simulate mouse movements and clicks.

 NumPy: For mathematical operations and coordinate calculations.

The first step in the implementation involved installing the required libraries using pip and setting up the Python environment. The webcam was initialized through OpenCV, which served as the primary video input device for gesture detection.

2. Capturing Real-Time Video Feed

Using OpenCV's VideoCapture() function, frames are captured from the webcam in real time. Each frame is flipped horizontally to provide a mirror-like interface for the user, making gesture control more intuitive. These frames are then resized and passed to the hand detection module.

The system ensures a continuous loop that processes each video frame for detecting gestures. It includes a check to terminate the loop when a specific key is pressed, usually the "ESC" key or 'q'.

3. Hand Detection and Landmark Extraction

The core of the gesture recognition system relies on MediaPipe's Hand module, which uses deep learning to identify and track hands. It outputs 21 landmarks per detected hand. These landmarks include fingertips (thumb to pinky), intermediate joints, and the wrist.

Each landmark is assigned a unique index. For example:

 Index 8 → Tip of the index finger

 Index 4 → Tip of the thumb

 Index 12 → Tip of the middle finger

The pixel coordinates of these points are calculated based on the camera frame resolution.

The detection is robust to minor variations in hand orientation and distance from the

camera.

4. Gesture Recognition Logic

The hand gestures are recognized based on:

 Which fingers are raised or folded

 The distance between fingertips

 The relative orientation of fingers

For example:

 If only the index finger & middle finger is raised, it is interpreted as a cursor movement gesture.

 If the index is down and middle finger is up , a left-click is performed.

 If the index is up and middle finger is down , a right-click is triggered.

 If index & middle finger are joined , double click is triggered.

To detect a click gesture, the system calculates the Euclidean distance between two landmarks (e.g., between landmarks 8 and 12). If the distance is below a threshold, the gesture is validated, and the corresponding mouse action is executed.

5. Cursor Movement and Control

One of the key innovations in this implementation is the use of relative movement tracking for cursor control. Instead of directly mapping the fingertip's x, y position to screen coordinates (which could cause pointer jumps), the algorithm computes the difference (dx, dy) between the current and previous fingertip positions. The cursor is moved using pyautogui.moveRel(dx, dy), creating smooth and controlled motion. This approach is more user-friendly and stable, as it allows for repositioning the hand within the frame without disturbing the pointer's location.

6. Executing Mouse Actions

The PyAutoGUI library is responsible for translating recognized gestures into actual

mouse actions. The major commands include:

 pyautogui.moveRel(x, y): Moves the cursor by a relative offset.

 pyautogui.click(): Performs a left-click.

 pyautogui.rightClick(): Performs a right-click.

 pyautogui.scroll(units): Scrolls vertically based on finger gestures.

 pyautogui.mouseDown() and pyautogui.mouseUp(): Used together for drag-and-drop

functionality.

These functions are platform-independent and simulate native mouse behavior on

Windows, macOS, or Linux.

7. Performance Considerations and Testing

The system was tested under different lighting conditions, hand sizes, and background scenarios. It performed best under consistent lighting with a plain background. Lag was minimal when tested on systems with moderate processing power, thanks to the lightweight design of MediaPipe.

Edge cases—such as quick hand movement, overlapping fingers, or partial hand

visibility—were addressed using error handling and landmark validation logic. If the hand is not detected in a frame, the system temporarily pauses mouse actions to avoid unexpected behavior.

Key Implementation Steps (Summary)

 Python + OpenCV: For video feed and frame manipulation.

 MediaPipe Hands: For real-time 21-point hand landmark detection.

 Landmark Analysis: For determining which fingers are raised and their positions.

 Gesture-to-Action Mapping: Based on finger combinations and distances.

 Relative Cursor Movement: Ensures smooth pointer tracking.

 PyAutoGUI: Executes system-level mouse events like click, scroll, drag-and-drop.

 Testing and Tuning: Adjusting thresholds, gesture timing, and frame refresh rates.

The implementation of the Gesture-Controlled Virtual Mouse demonstrates a seamless integration of computer vision and automation technologies to create a contactless, intuitive user interface. The methodology is modular, scalable, and hardware- independent—requiring only a webcam and software tools. By leveraging real-time hand tracking and intelligent gesture recognition, the system offers a practical alternative to conventional input devices, with room for expansion into more advanced applications such as multi-gesture controls, AI-based recognition, and integration with AR/VR environments.

Performance and Results:

The effectiveness of the Gesture-Controlled Virtual Mouse has been evaluated based on several performance metrics:

Accuracy: The system demonstrates high accuracy in recognizing predefined hand

gestures and voice commands under controlled lighting conditions. However, variations in lighting and background can affect recognition performance.

Responsiveness: With efficient processing pipelines, the system offers real-time

responsiveness, ensuring minimal latency between user gestures or voice commands and the corresponding system actions.

User Experience: Users have reported an intuitive experience, finding the gesture and voice controls to be natural extensions of traditional input methods.