# Gesture Controlled Virtual Mouse using Deep Learning

Shaik Shahnaz Begum[1], Ranga Jyothika Vijaya Sravya[2], Siddabattula Madhu[3], Pilla Pavan Sailendra[4], Dr.D. Kavitha[5]
[1,2,3,4]*Student, Prasad V Potluri Siddhartha Institute of Technology*
[5]*Associate Professor, Prasad V Potluri Siddhartha Institute of Technology*

***Abstract*—As human-computer interaction is evolving rapidly, new approaches to touchless device control are becoming essential. Our system implements computer vision and machine learning techniques to create foundational technologies for future AR applications. Gesture-controlled virtual mouse system comprises three main components: gesture recognition, gesture training, and control implementation. Our system enables users to control computer operations through hand gestures captured by a webcam. At its core, the system uses a hand landmark detection model that identifies 21 key points on the hand, enabling precise tracking of finger positions and movements. The gesture recognition module analyzes these landmarks to classify hand poses into predefined gestures, while the custom gesture training component allows users to define and save their own gesture mappings. Our project provides real-time gesture detection and control by utilizing MediaPipe for hand tracking with the help of Computer Vision (CV) for video capturing and Machine Learning (ML) algorithms for customization. The implementation combines MediaPipe and Pybind11 to track hand movements accurately for smooth real-time detection. Our system offers intuitive cursor control, adjustable system parameters, and support for multiple hand tracking with dynamic gesture recognition. This comprehensive approach enables fluid and natural human-computer interaction.**

## I. INTRODUCTION

Imagine using natural motions that flow through the air to control a computer in the contemporary digital era instead of a mouse or touchpad. This goal is realized by our project gesture-controlled virtual mouse technology, which uses natural hand gestures to revolutionize computer interaction. by combining machine learning with cutting- edge computer vision technology. We have developed a system that can precisely translate human hand movements into computer commands in real time. Using your standard webcam, the system detects 21 distinct spots on your hand and learns to identify your distinct motions for various functions, ranging from basic clicks to adjusting brightness and volume. You can teach our solution your favorite hand gestures, which makes it genuinely unique.

Our technology provides a hands-free option that is both effective and convenient, whether you're in a hospital operating room that requires sterile control, presenting a presentation, or just want a more comfortable method to use your computer. Our concept is a viable choice for anyone interested in entering the future of computer interaction because it can be done using any common webcam. Gesture controlled system is helpful for both personal and professional use since it can adjust to the preferences of each user while still retaining excellent
accuracy and responsiveness.

Our gesture-controlled virtual mouse is a significant advance in integrating technology more organically into our daily lives as we transition to more user-friendly approaches
to human-computer interaction.

## II. RELATED WORKS

Several foundational studies have contributed to the development of gesture-controlled interfaces while also pointing up areas for development:

HMM-based gesture recognition with Forward/Viterbi algorithms and Baum-Welch training was introduced by Rabiner and Juang (1986) [1]. Through Python-Arduino connection, their Arduino-ultrasonic sensor technology made it possible to control basic media. Although it was novel for interactive applications, its precision was limited.

Liou and Hsieh (2018) [2] identified six hand gestures (two static, four dynamic) by combining face-based skin color identification with motion history imaging. Their system's accuracy was 94.1% across five test participants using face-based ROI analysis and Haar-like features. One of Liou and Hsieh's model's limitations is that it only includes six preset gestures.

"Hand Gesture Controlled Computer Mouse" was created by Kuraparthi et al. (2019) [3] utilizing image processing methods. Their method implemented simple motions for mouse control and used OpenCV for hand detection and tracking. Despite being novel, it had trouble being accurate in complicated backgrounds and different lighting conditions.

Using ultrasonic sensors and an Arduino Uno, Kulkarni and Potdar (2019) [4] created an affordable RADAR-like object detecting system. Using a Raspberry Pi 3 for data processing and a SIM808 for SMS notifications of object distance, angle, and timestamps, their solution combined IoT hardware. It was only capable of proximity sensing, but it worked well for simple object detection.

A marker-free, reasonably priced gesture recognition system for complete HCI operations, such as system functions and application control, was put into place by Subramanian et al. (2020) [5]. Although they prioritized accessibility and wide device compatibility, their system could only use programmed gestures.

CNN models were used in the presentation of "Vision-Based Hand Gesture Recognition System for Virtual Mouse" by Jain and Meena (2020) [6]. Their method demonstrated latency problems in real-time applications and
needed significant computer resources, although achieving 92% accuracy in gesture detection.

"DeepHand: Real-time Hand Gesture Recognition" was presented by Park and Kim (2021) [7] utilizing the MediaPipe framework. Their method handled numerous motions and showed 96% accuracy in hand landmark detection. It did not, however, include system-level integration or customization possibilities.

Using colored fingertip tracking, Dhyanchand and Reddy (2021) [8] created a virtual mouse system that achieved 90% detection accuracy in office illumination (500–600 lux). Their technology accommodated diverse skin tones and provided basic mouse capabilities with different colored cap arrangements. It was inventive, but it needed colored markers and was light-sensitive.

Both static and dynamic gestures were incorporated into the "AIController: Adaptive Hand Gesture Interface" created by Zhang et al. (2022) [9]. Their 94% accurate gesture detection system used transfer learning, but it needed specialized depth cameras.

Sharma and Patel (2023) [10] proposed "GestureFlow: Natural Interface Computing" combining MediaPipe with custom neural networks. Their implementation supported multi-hand tracking and achieved low latency, though limited to predefined gesture sets.

The accessibility focusses of "HandMouse: Accessible Computer Control" was developed by Liu et al. (2023) [11]. Despite using common webcams and achieving 91% accuracy in gesture recognition, their solution lacked customization choices and system control functions.

## III. MOTIVATION

Our motivation behind developing a gesture-controlled virtual mouse system stems from the growing need for touchless human-computer interaction in various settings. Traditional input devices like mice and keyboards can be impractical in sterile environments such as operating rooms, or challenging for users with mobility limitations. Our system addresses these needs by enabling natural hand gesture control using standard webcams, making computer interaction more accessible and intuitive. By incorporating MediaPipe's advanced hand tracking with custom gesture training, we've created a solution that adapts to individual user preferences while maintaining high accuracy and responsiveness. This technology not only enhances accessibility but also opens new possibilities for interactive presentations, virtual education, and hands-free computing across diverse professional and personal scenarios.

## IV. PROBLEM STATEMENT

The development of a gesture-controlled virtual mouse system addresses critical limitations in current computer interaction methods. Traditional input devices pose challenges in sterile environments like operating rooms, accessibility needs for users with mobility impairments, and
scenarios requiring hands-free operation. While

existing gesture recognition solutions offer alternatives, they often require specialized hardware, have limited gesture sets, struggle with environmental variations, and lack comprehensive system integration. These systems frequently suffer from accuracy issues in real-time applications and high computational demands. Therefore,Gesture control system focuses on natural human behaviour, providing easier way for users to interact with computer. Gesture control can offer an immersive and futuristic experience, blending seamlessly with augmented reality (AR) and virtual reality (VR) systems. Our approach aims to support hands-free computing by eliminating the need for specialized equipment while providing reliable, adaptable, and efficient computer interaction for diverse user needs and creating co-friendly environments by reducing plastic for hardware (like mouse). Below definitions are used throughout research.

*Definition 4.1:* Computer Vision is used for Image processing and analysis science utilizing core techniques like feature detection, pattern recognition, and image segmentation.

*Definition 4.2:* Machine Learning: A subset of AI where computers learn from data and make decisions or predictions without being explicitly programmed

*Definition 4.3:* OpenCV**:** A library for real-time computer vision tasks like image processing, object detection, and face recognition.

*Definition 4.4:* MediaPipe**:** A framework for building real- time AI applications, with tools for tasks like facial detection and hand tracking.

*Definition 4.5:* PyAutoGUI**:** A Python library to automate GUI tasks, controlling the mouse, keyboard, and screenshots.

*Definition 4.6:* Screen Brightness Control**:** The ability to adjust screen brightness programmatically, often using libraries like screen-brightness-control.

*Definition 4.7:* Tkinter**:** A Python library for creating simple desktop GUI applications with windows, buttons, and text boxes.

*Definition 4.8:* JSON (JavaScript Object Notation) is a lightweight, text-based data in key value pair format used to store and exchange data. It's easy for both humans to read and write, and machines to parse and generate.

These components work together to enable accurate hand tracking, gesture recognition, and system control through natural hand movements.

## V. DESIGN AND IMPLEMENTATION

Our project implements a gesture-based computer control system using computer vision, pyautogui and mediapipe. The backbone of our system is MediaPipe for hand landmark detection and PyAutoGUI for system control. Main controller maintains the camera feed and processes hand gestures in real-time. It supports single hand operations (either right or left only for customization), with customizable gesture recognition. The system can distinguish between major (dominant) and minor hands, enabling different controls for each.

Gesture recognition works by analyzing finger positions and their relationships. The controller tracks specific landmarks on the hand to determine finger states (open/closed) and complex gestures like pinching. Each recognized gesture maps to specific system controls such as mouse movement, clicks, scrolling, brightness adjustment, and volume control.

An enhanced version adds support for custom gesture training. Users can record new gestures by demonstrating them to the camera, and the system saves the finger state patterns in a JSON file. This allows for personalized controls like right-click, left-click, double- click, drag-and-drop, and multi-select, scroll operations.

The training component includes functionality to manage custom gestures, allowing users to add new ones or remove existing ones. It records multiple frames (upto 300 frames) for a gesture to ensure accurate recognition and uses statistical analysis to determine the most consistent finger state pattern.

For pinch gestures, it tracks both vertical and horizontal movements to control various system parameters. It has another feature to reset the gesture training data, clearing all custom gestures and returning the system to its default state. The entire system is modular, with clear separation between gesture recognition, control implementation, and training functionality.

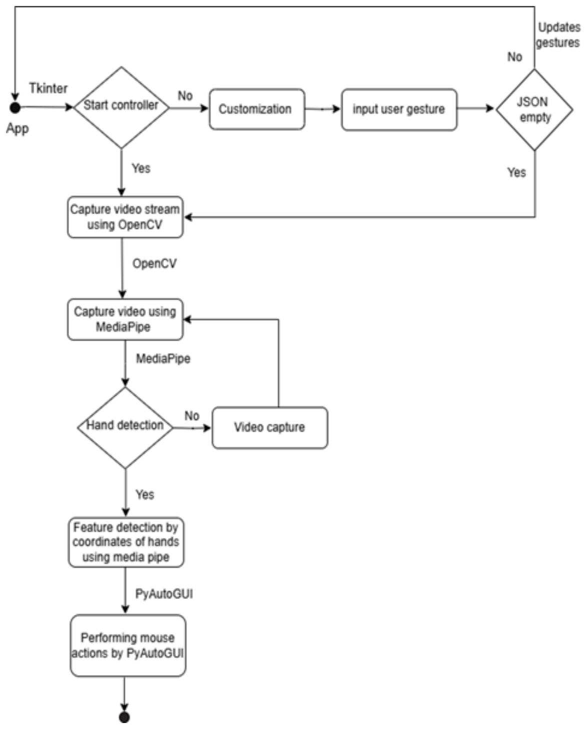

Updates gestures
No
Tkinter
App
Start controller
No
Customization
input user gesture
JSON empty
Yes
Yes
Capture video stream using OpenCV
OpenCV
Capture video using MediaPipe
MediaPipe
Hand detection
No
Video capture
Yes
Feature detection by coordinates of hands using media pipe
PyAutoGUI
Performing mouse actions by PyAutoGUI

Fig 1: Activity diagram for gesture controlled virtual mouse

Fig 1 illustrates gesture control system architecture, starting from application initialization with Tkinter. At the entry point, a controller check determines whether to proceed with customization or main control flow. The customization path enables users to input and store custom gestures in a JSON file, which can be updated based on user preferences.

The main control flow begins with video stream capture using OpenCV, which feeds into MediaPipe for advanced hand tracking and detection. This creates a continuous processing loop where video frames are analyzed for hand presence. When hands are detected, the system processes feature detection using MediaPipe coordinate system to identify specific gestures.

Once gestures are recognized, PyAutoGUI translates these detected features into corresponding mouse actions on the computer system. The workflow maintains a cyclical process of capturing, detecting, and executing actions, with the JSON storage system maintaining gesture configurations. This architecture ensures real-time response while allowing for gesture customization and system adaptability.

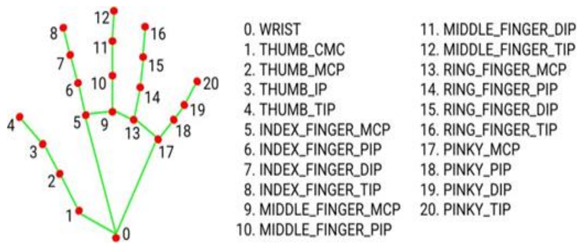


Fig 2: Hand Land marks detection using MediaPipe [13]

Fig 2 shows hand landmark mapping system used in hand gesture recognition, particularly with MediaPipe hand tracking solution. It depicts 21 key points (numbered 0-20) representing crucial hand landmarks, with the wrist as point 0 and finger joints mapped systematically. The mapping includes metacarpophalangeal (MCP), proximal interphalangeal (PIP), and distal interphalangeal (DIP) joints, plus fingertips for each finger. These landmarks correspond to anatomical features including the wrist (0), and points along each finger from base to tip - thumb (1-4), index finger (5-8), middle finger (9-12), ring finger (13-16), and pinky (17-20). The points are connected by lines showing their relationships, creating a skeletal representation that's used for tracking hand movements and gestures.

Pseudo code for default gestures:

```
# Initialize system and state function initialize ():
start_webcam() initialize_mediapipe_opencv() state =
{
'last_click_time': 0, 'is_dragging': False, 'selection':
None, 'neutral_timeout': 2
}
# Main program loop function main ():
state = initialize ()
while not check_stop_condition():
frame = process_frame(capture_webcam_frame())
hands = detect_hands(frame)
if not hands: reset_states(state) continue
# Process gestures
gesture = detect_gesture(hands) match gesture:
case 'neutral': # All fingers extended
   reset_states(state)
case 'move': # Index + middle up
handle_cursor_movement(hands.position,
state.is_dragging)
case 'click': # Middle up + index bent
handle_click(state.last_click_time)
case 'drag': # Index up + thumb pressed
```

```
handle_drag(hands.position, state)
case 'select': # Three fingers up
handle_selection(hands.position, state)
case 'system': # System controls
handle_system_controls(gesture, hands)
case 'custom': process_custom_gesture(hands)
# Gesture handlers
function handle_click(last_time): current_time =
get_current_time()
    if current_time - last_time < 0.3:  # Double click
threshold
    perform_double_click() else:
    perform_single_click() return current_time
function handle_system_controls(gesture, hands):
match gesture:
case 'volume': # Vertical movement
adjust_volume(hands.movement)
case 'brightness': # Horizontal movement
adjust_brightness(hands.movement)
case 'browser':  # Pinky + ring open_chrome()
case 'search':  # Ring + middle open_search()
case 'email':  # Index + pinky open_email()
# Start program
main ()
```

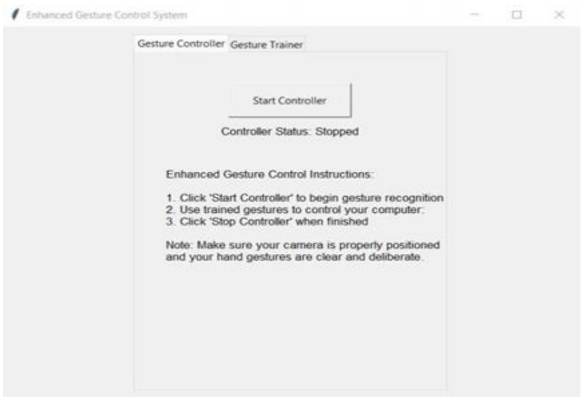
Above pseudo code explains about functionality of gesture controlled virtual mouse. Firstly, system begins by initializing components like webcam and computer vision tools (MediaPipe and OpenCV). The system maintains a state object that keeps track of various interaction parameters such as click timing, dragging status, and selection states, ensuring smooth and consistent user experience.

The system supports various interactions including cursor movement (using index and middle fingers), clicking operations (through finger bends), volume and brightness controls (via hand movements), and application launching (using specific finger combinations). It also handles more complex operations like drag-and-drop and multiple selections. A neutral gesture (all fingers extended) serves as a reset mechanism.

Our project has customization of gesture sets to accommodate different user needs and preferences. Code captures stable frame rates between 25-30 FPS on standard hardware configurations, providing smooth cursor movement through effective position dampening algorithms. Our system exhibits some vulnerability to lighting conditions, with performance degradation in low-light environments. Similar finger positions occasionally trigger false positives, and overlapping hands can create gesture conflicts that impact recognition accuracy.

## VI. RESULTS AND ANALYSIS

Fig 4 is Main application with GUI. Creates tabbed interface for gesture control and training. Handles video capture and display. Manages gesture training workflow. Two-tab interface for main application. Gesture Controller and Gesture Trainer. In Controller tab, video feed, start/stop controls, status display are present.In Trainer tab, recording, removing, and reset options are present.



Controller Status: Stopped

Enhanced Gesture Control Instructions:

1. Click 'Start Controller' to begin gesture recognition
2. Use trained gestures to control your computer:
3. Click 'Stop Controller' when finished

Note: Make sure your camera is properly positioned and your hand gestures are clear and deliberate.

Fig 3: Gesture Controller tab

Enhanced gesture controller. Extends base controller with custom gesture support. Handles trained gesture recognition. Maps gestures to system actions (clicks, scrolling, volume, brightness).

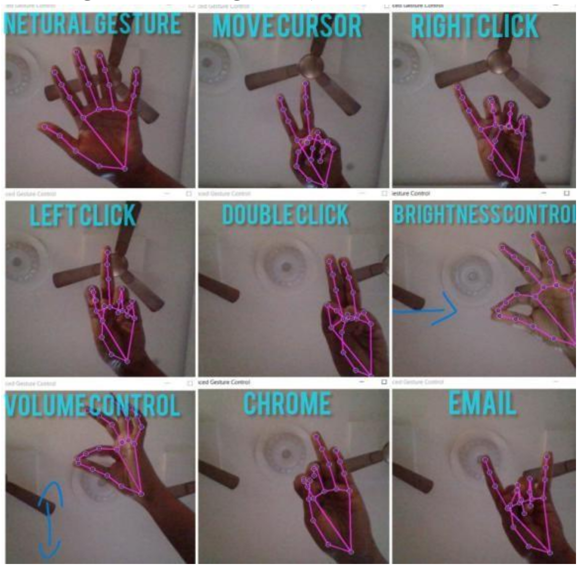
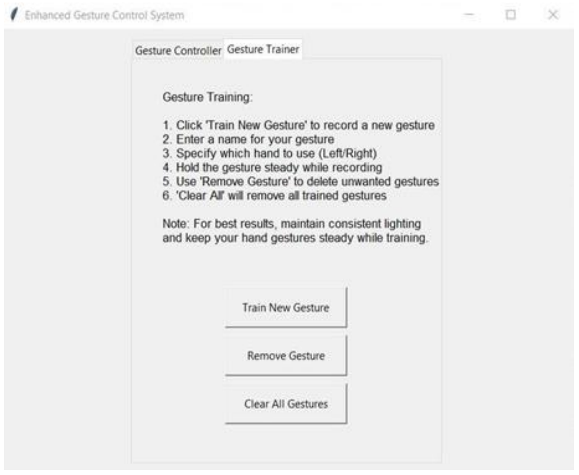


Fig 4: Hand Gestures for mouse operations

The hand gestures shown in these images gives a touchless way to control your computer using hand movements. The purple lines and dots on each hand help the system track and understand different gestures.
Starting with a neutral position (all fingers spread), users can perform various actions: move the cursor with two fingers pointing, click using single finger gestures (middle finger for left, index for right), and double-click using a peace sign.
System adjustments like volume and brightness are controlled through hand movements up/down and left/right. Quick gestures like ring and pinky fingers up launch Chrome, while index and pinky fingers open email.
Fig 5 is training interface with three options. First, records new gesture patterns. Second, delete selected gestures. Third, clear all customized gestures and default gestures will work.



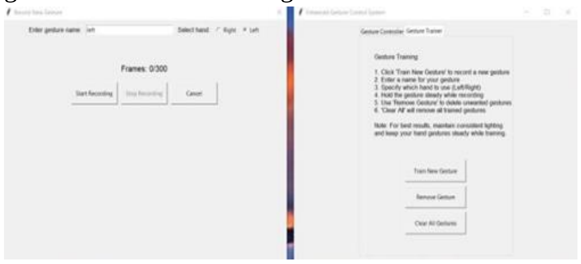
Gesture Training:

1. Click 'Train New Gesture' to record a new gesture
2. Enter a name for your gesture
3. Specify which hand to use (Left/Right)
4. Hold the gesture steady while recording
5. Use 'Remove Gesture' to delete unwanted gestures
6. 'Clear All' will remove all trained gestures

Note: For best results, maintain consistent lighting and keep your hand gestures steady while training.

Fig 5: Gesture Trainer

Training the gestures and by recording user given gesture. Saves gesture data to JSON replace default gestures with customized gestures.



Enter gesture name: left    Select hand: Right [✓ Left]

Frames: 0/300

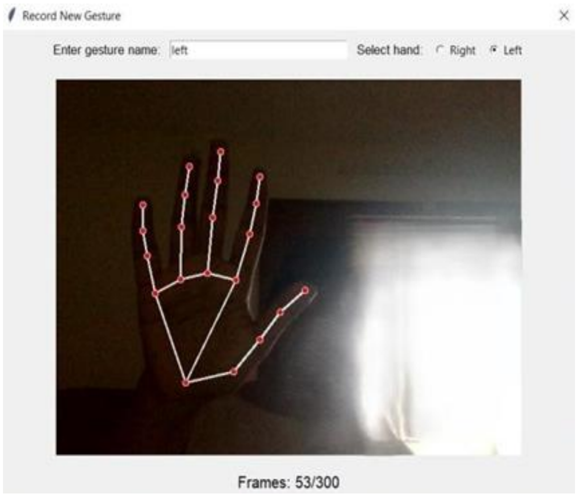
Gesture Training:

1. Click 'Train New Gesture' to record a new gesture
2. Enter a name for your gesture
3. Specify which hand to use (Left/Right)
4. Hold the gesture steady while recording
5. Use 'Remove Gesture' to delete unwanted gestures
6. 'Clear All' will remove all trained gestures

Note: For best results, maintain consistent lighting and keep your hand gestures steady while training.

Fig 6: Training the gesture.

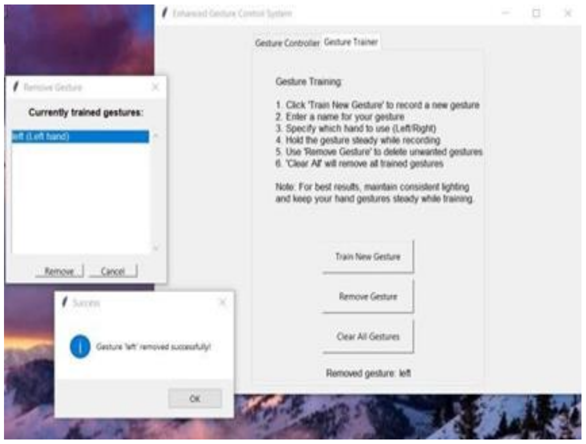
Here in fig 6, user can choose hands either left or right. User has to give the feature they want to replace with (e.g: left for left click) as input. In fig 7, system records user given gesture upto 300 frames and save the gesture in JSON file.



Enter gesture name: left    Select hand: Right [✓ Left]

Frames: 53/300

Fig 7: Record the gesture



Currently trained gestures:
left (Left hand)

Gesture 'left' removed successfully!

Gesture Training:

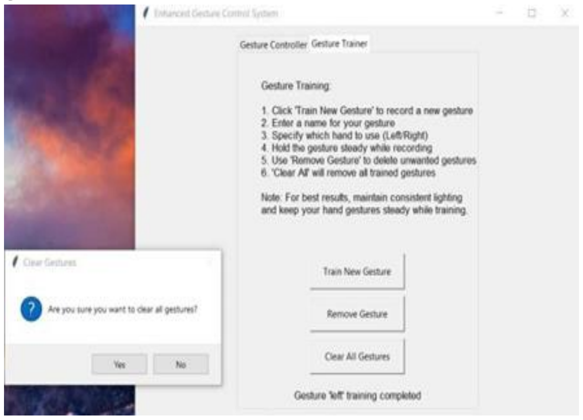
1. Click 'Train New Gesture' to record a new gesture
2. Enter a name for your gesture
3. Specify which hand to use (Left/Right)
4. Hold the gesture steady while recording
5. Use 'Remove Gesture' to delete unwanted gestures
6. 'Clear All' will remove all trained gestures

Note: For best results, maintain consistent lighting and keep your hand gestures steady while training.

Removed gesture: left

Fig 8: Remove gesture

In fig 8, user can select the gesture they want to remove. In fig 9, 10 users clear all the customized gestures. We can see the status for all operations in gesture trainer tab at bottom.



Are you sure you want to clear all gestures?

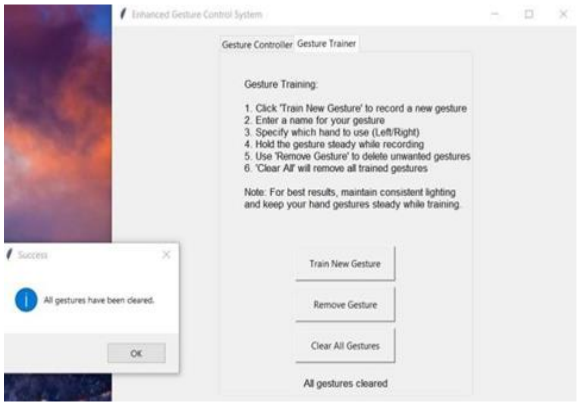
Gesture Training:

1. Click 'Train New Gesture' to record a new gesture
2. Enter a name for your gesture
3. Specify which hand to use (Left/Right)
4. Hold the gesture steady while recording
5. Use 'Remove Gesture' to delete unwanted gestures
6. 'Clear All' will remove all trained gestures

Note: For best results, maintain consistent lighting and keep your hand gestures steady while training.

Gesture 'left' training completed

Fig 9: Clear all the gestures



Gesture Training:

1. Click 'Train New Gesture' to record a new gesture
2. Enter a name for your gesture
3. Specify which hand to use (Left/Right)
4. Hold the gesture steady while recording
5. Use 'Remove Gesture' to delete unwanted gestures
6. 'Clear All' will remove all trained gestures

Note: For best results, maintain consistent lighting and keep your hand gestures steady while training.

All gestures have been cleared.

All gestures cleared

Fig 10: After, clearing all the gestures.

## VII. CONCULSION

Our project work demonstrates robust hand gesture recognition for computer control. It supports both default and custom gestures, enabling natural interaction through hand movements for cursor control, system adjustments, and application launching.

While facing challenges in low-light conditions and complex gesture recognition, the system's adaptability to different users and customization options proves its practical value. The implementation balances accessibility with functionality, making it suitable for hands-free computer interaction and accessibility applications. Our project work establishes a viable foundation for gesture-based interfaces, with clear paths for enhancement in lighting adaptation, conflict resolution, and resource optimization.

## REFERENCES

[1] L. R. Rabiner and B. H. Juang, "An introduction to hidden Markov models," IEEE ASSP Magazine, vol. 3, no. 1, pp. 4-16, Jan 1986.

[2] D. H. Liou and C. C. Hsieh, "A Real Time Hand Gesture Recognition System Using Motion History Image," 2018 2nd International Conference on Signal Processing and Communication Systems, pp. 394-399.

[3] S. Kuraparthi, M. Kollati, and K. Kora, "Hand Gesture Controlled Computer Mouse," International Journal of Advanced Research in Computer Science, vol. 10, no. 2, pp. 23-27, 2019.

[4] A. U. Kulkarni and A. M. Potdar, "RADAR based Object Detector using Ultrasonic Sensor," International Journal of Engineering Research & Technology, vol. 8, no. 6, 2019.

[5] A. Subramanian, A. Haria, J. S. Nayak, N. Asokkumar, and S. Poddar, "Hand gesture recognition for human computer interaction," Journal of Physics: Conference Series, vol. 1427, 2020.

[6] A. Jain and R. Meena, "Vision-Based Hand Gesture Recognition System for Virtual Mouse," International Journal of Recent Technology and Engineering, vol. 9, no. 1, 2020.

[7] J. Park and S. Kim, "DeepHand: Real-time Hand Gesture Recognition using MediaPipe," IEEE Access, vol. 9, pp. 123034-123043, 2021.

[8] S. Dhyanchand and P. Reddy, "Virtual Mouse System Using Colored Fingertip Tracking," International Journal of Engineering Research & Technology, vol. 10, no. 3, 2021.

[9] L. Zhang, X. Wang, and Y. Liu, "AIController: Adaptive Hand Gesture Interface," IEEE Transactions on Human-Machine Systems, vol. 52, no. 1, pp. 89-98, 2022.

[10] R. Sharma and V. Patel, "GestureFlow: Natural Interface Computing," IEEE Sensors Journal, vol. 23, no. 4, pp. 3456-3467, 2023.

[11] K. Liu, J. Chen, and M. Wang, "HandMouse: Accessible Computer Control," International Journal of Human-Computer Interaction, vol. 39, no. 2, pp. 245-259, 2023.

[12] M. M. Akhtar, B. Thakur, P. Raj, G. Kumar, and P. Yadav, "Computer Cursor Control through Hand Gestures using Machine Learning," International Journal of Research Publication and Reviews, vol. 3, no. 10, pp. 1571-1576, Oct. 2022.

[13] Shukla, A. R. (2022). AI virtual mouse using hand gesture recognition. International Journal of Research Publication and Reviews, 168–173. https://doi.org/10.55248/gengpi.2022.3.10.13