HOVERLY-GESTURE AND VOICE CONTROLLED VIRTUAL MOUSE

A PROJECT REPORT

Submitted by

KEERTHANA V	**221701029**
LOSHIKA G	**221701034**

In partial fulfilment of the course

CD19711 PROJECT - I

for the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND DESIGN

RAJALAKSHMI ENGINEERING COLLEGE, THANDALAM

ANNA UNIVERSITY: CHENNAI – 602105

OCTOBER 2025

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY: CHENNAI – 602105

BONAFIDE CERTIFICATE

Certified that this Report titled **“HOVERLY - GESTURE AND VOICE CONTROLLED VIRTUAL MOUSE”** is the Bonafide work of **“KEERTHANA V (221701029), LOSHIKA G (221701034)”** who carried out the project work for the subject **CD19711 - PROJECT - I** under my supervision.

SIGNATURE	**SIGNATURE**
Prof. Uma Maheshwara Rao	**Mr. S. Gunasekar**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Associate Professor	Assistant Professor
Department of Computer Science and Design	Department of Computer Science and Design
Rajalakshmi Engineering College	Rajalakshmi Engineering College
Chennai – 602105	Chennai – 602105

Submitted to Project and Viva Voce Examination for the CD19711 - Project – I held on ________________.

Internal Examiner External Examiner

ABSTRACT

The increasing demand for contactless and intelligent human–computer interaction has inspired the development of Hoverly: Gesture and Voice Controlled Virtual Mouse. This project introduces an AI-powered system that enables users to control computer functions through hand gestures and voice commands, eliminating the need for physical peripherals like a traditional mouse. The system utilises MediaPipe's palm detection and OpenCV to accurately track hand movements for cursor control, clicks, and scrolling actions. Simultaneously, a speech recognition module interprets user voice inputs for executing system-level commands such as opening applications or performing clicks. By integrating these two modalities, Hoverly enhances accessibility, promotes hygiene through touch-free operation, and provides a futuristic human–computer interaction experience. This work demonstrates the potential of combining computer vision and natural language processing to create intelligent, adaptive, and user-friendly interaction systems.

ACKNOWLEDGEMENT

Initially, we thank the Almighty for being with us through every walk of our lives and for showering His blessings throughout our endeavor to present this report. Our sincere thanks to our Chairman, **Mr. S. Meganathan, B.E., F.I.E.**; our Vice Chairman, **Mr. Abhay Shankar Meganathan, B.E., M.S.**; and our respected Chairperson, **Dr. (Mrs.) Thangam Meganathan, Ph.D.**, for providing us with the requisite infrastructure and for their dedicated efforts in educating us at their premier institution.

We extend our sincere thanks to **Dr. S. N. Murugesan, M.E., Ph.D.**, our beloved Principal, for his kind support and for providing the necessary facilities to complete our work on time. We express our heartfelt gratitude to our Head of the Department and Associate Professor, **Mr. S. Uma Maheshwara Rao, M.A., M.F.A. (Design)**, Department of Computer Science and Design, for his guidance and encouragement throughout the project work. Finally, we convey our sincere thanks to our Project Coordinator and Supervisor, **Mr. S. Pradeep Kumar, M.E.**, Department of Computer Science and Design, Rajalakshmi Engineering College, for his valuable guidance and continuous support throughout the course of the project.

KEERTHANA V (221701029)

LOSHIKA G (221701034)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
HCI	Human Computer Interaction
CV	Computer Vision
API	Application Programming Interface
AI	Artificial Intelligence
UI	User Interface
GUI	Graphical User Interface
NLP	Natural Language Processing
OS	Operating System
PyAutoGUI	Python Automation GUI

CHAPTER-1
INTRODUCTION

1.1 GENERAL

Human–Computer Interaction (HCI) has evolved into one of the most influential domains in computer science and engineering, shaping the way individuals communicate with technology. From mechanical switches and keyboards to touch interfaces and gesture-controlled devices, each advancement brings computers closer to the natural modes of human communication. Among these, gesture and voice recognition technologies stand out as significant milestones. They allow users to interact with machines through physical movement and verbal instructions, bridging the gap between human expression and digital execution.

The "Gesture and Voice Controlled Virtual Mouse" project aims to strengthen this interaction by providing a system that replaces the traditional hardware mouse with a software-based, intelligent alternative. The virtual mouse captures hand gestures via a webcam and interprets voice commands through a microphone to perform actions such as cursor movement, clicking, dragging, and scrolling. This innovation represents a crucial step towards intuitive, contactless computing where natural behaviour is translated into functional computer control.

The project leverages OpenCV for image processing and MediaPipe for hand landmark detection, both implemented in the Python programming environment. These technologies enable real-time gesture recognition by identifying key hand positions and interpreting them as specific commands. Simultaneously, SpeechRecognition libraries process spoken input to perform operations such as opening applications, navigating files, or executing keyboard functions. The integration of these technologies not only enhances user convenience but also reflects the growing synergy between artificial intelligence, computer vision, and signal processing.

The general aim is not merely to replicate the functions of a mouse but to redefine interaction itself—making human–computer communication more natural, seamless, and intelligent.

1.2 OBJECTIVE

The primary objective of the Gesture and Voice Controlled Virtual Mouse project is to design and implement a fully functional, contactless computer interaction system that can perform all traditional mouse operations through gestures and voice commands. The system intends to eliminate dependency on physical input devices and create a seamless bridge between human expression and digital control.

The specific objectives of the project include the following:

I. **Develop a Gesture-Based Mouse Control Mechanism:**
The system captures live video feed through a webcam, detects hand gestures using MediaPipe, and converts them into mouse operations such as pointer movement, clicks, scrolling, and drag-and-drop. By tracking the positions of fingertips and joints, the system accurately maps the motion of the hand to the screen coordinates.

II. **Integrate a Voice Command System:**
Using speech recognition algorithms, the project enables the user to perform system operations such as opening files, switching windows, or executing clicks using predefined voice commands. This enhances multitasking and minimises the need for manual intervention.
Ensure Real-Time Responsiveness and Accuracy:
The system is designed to respond instantly to gestures and speech, maintaining high accuracy under varying lighting and background conditions. Efficient image filtering and noise suppression techniques are applied to enhance detection quality.

III. **Enhance Accessibility and Usability:**
One of the major objectives is to support individuals with limited hand mobility or other physical challenges by offering an effortless and non-contact interface for computer control.

In achieving these goals, the project contributes toward the larger vision of natural user interfaces (NUIs), which rely on human gestures, voice, and motion rather than physical devices. By accomplishing this, the project reinforces the trend toward intelligent, responsive, and user-adaptive computing systems.

1.3 EXISTING SYSTEM

The traditional computer mouse has remained a standard input device for decades. Its mechanical design and hand-controlled motion allow users to move cursors, click, and navigate interfaces efficiently. However, as computing environments become more dynamic and integrated with artificial intelligence, physical devices have begun to show limitations. Existing mouse-based systems depend on manual operation, specific surfaces, and are subject to wear and calibration issues. Furthermore, they fail to accommodate situations where touch-free operation is preferable or required.

Earlier attempts to move beyond physical devices have included glove-based gesture recognition systems, colour marker tracking, and infrared sensor-based input devices. In glove-based systems, users had to wear sensors or colored gloves, which interfered with comfort and limited mobility. Colour marker tracking, which relied on recognizing colored dots or tips on fingers, suffered from inconsistent lighting and background interference. Infrared systems, while precise, required specialised and costly hardware, making them impractical for general users.

Voice-controlled systems also existed independently but were often unreliable due to background noise, accent variation, and latency in speech processing. Moreover, combining gesture and voice recognition was rarely implemented in a unified framework, which restricted real-time flexibility and performance.

The existing systems could not generally integrate gesture and speech modalities cohesively. Most approaches focused on either motion tracking or voice input, without optimising for synchronisation or computational efficiency. Additionally, the hardware dependency and setup complexity made them unsuitable for daily use.

Thus, the limitations of existing systems can be summarised as follows:

I. Dependence on external sensors or gloves.
II. Sensitivity to lighting and background conditions.
III. Limited range of detectable gestures.
IV. Inability to perform complex mouse operations simultaneously.
V. Lack of integrated voice functionality.

These drawbacks highlight the need for a cost-effective, camera-based virtual mouse system that operates using standard webcams and microphones, offering simplicity, reliability, and real-time interaction without additional hardware. The project builds upon the foundation of HCI research to overcome these constraints and deliver a natural, accurate, and efficient alternative to the conventional mouse.

1.4 PROPOSED SYSTEM

The proposed Gesture and Voice Controlled Virtual Mouse system offers a comprehensive solution to overcome the shortcomings of traditional input devices. It introduces a dual-interaction model for gesture recognition and speech control implemented through computer vision and natural language processing. The system leverages a standard webcam to detect hand gestures using MediaPipe's hand tracking model, while voice commands are processed through SpeechRecognition and related libraries in Python. Together, these modules create a hands-free, intelligent, and adaptive interface for operating a computer.

The gesture recognition module functions by continuously capturing video frames from the webcam. Each frame undergoes image preprocessing using OpenCV, which includes operations such as grayscale conversion, thresholding, and contour detection. MediaPipe then identifies 21 hand landmarks, including fingertip and joint coordinates. Based on these detected points, the algorithm interprets gestures such as index finger movement for cursor control, thumb–index proximity for clicks, and multi-finger patterns for drag, scroll, or volume adjustments. This allows real-time control over on-screen cursor actions without any physical device.

Simultaneously, the voice control module complements gestures by executing commands through natural language. The system listens via a microphone, filters background noise, and interprets speech to perform functions like opening folders, copying files, or minimising windows. This module ensures that users can issue instructions even when hand movement is limited, providing flexibility and efficiency.

The integration of both modules ensures that gesture and voice operations can run concurrently, making the system more robust and user-friendly. The software is designed in Python due to its extensive support for machine learning and computer vision libraries. The use of open-source frameworks guarantees scalability, platform independence, and cost-effectiveness.

Key advantages of the proposed system include:

I. **No external hardware dependency:** Operates using only the built-in camera and microphone.
II. **User adaptability:** Works across diverse lighting conditions with optimised filters.
III. **High precision and low latency:** Real-time response with smooth cursor tracking.
IV. **Enhanced accessibility:** Usable by differently-abled individuals.
V. **Multi-modal input:** Combines gesture and voice recognition seamlessly.

The proposed system transforms conventional human–computer interaction by introducing an intelligent, touch-free, and inclusive interface. It provides an efficient alternative to the physical mouse by translating human intent—expressed through gestures and speech into executable digital commands. This innovation not only simplifies computing but also represents a move toward smarter, perception-driven interfaces that redefine the boundaries of digital communication.

CHAPTER-2
LITERATURE REVIEW

2.1 HUMAN–COMPUTER INTERACTION (HCI)

Human–Computer Interaction (HCI) is the multidisciplinary field that studies how humans communicate with and control computers. It draws upon psychology, cognitive science, computer engineering, and design to create intuitive and efficient interfaces between humans and digital systems. HCI focuses on improving usability, performance, and user satisfaction through natural, adaptive interaction methods.

Historically, HCI began evolving in the 1980s alongside the rise of personal computers such as the IBM PC and Apple Macintosh. Early systems relied on physical devices keyboards, mice, and monitors, that demanded user adaptation to machine interfaces. Over time, the emphasis shifted toward making computers more responsive to human behaviour through technologies such as touch, gesture, and speech recognition.

In recent years, gesture and voice-based interfaces have emerged as essential components of modern HCI. By merging computer vision and artificial intelligence, HCI now moves beyond traditional devices toward perceptual computing, where machines can sense, understand, and respond to human intent. This principle forms the foundation of the Gesture and Voice Controlled Virtual Mouse project.

2.2 MEDIAPIPE FRAMEWORK

MediaPipe, developed by Google, is an open-source framework designed for building machine learning pipelines for real-time perception applications. It provides a flexible and efficient platform to process video, audio, and sensor data across multiple platforms, including desktops, mobile devices, and web applications.

In the context of this project, MediaPipe plays a crucial role in hand detection and tracking. It uses two core models within its pipeline: a palm detection model and a hand landmark model. The palm detector locates the presence of a hand in the image, while the landmark model identifies 21 specific hand points—including fingertips and joints—in 2.5D space. These landmarks are used to map gestures such as pointing, clicking, or scrolling, which the system interprets as mouse commands.

The advantages of using MediaPipe include high accuracy, low computational overhead, and the ability to run efficiently on CPUs without requiring external hardware. Its modular architecture allows developers to combine various calculators (processing blocks) to handle tasks like image capture, tracking, and feature extraction.

In this project, MediaPipe ensures real-time hand movement tracking with minimal delay, enabling smooth and responsive cursor motion. The robustness of the framework against variable lighting conditions and backgrounds further enhances the accuracy of gesture recognition.

A diagram illustrating the architecture of MediaPipe Hand Tracking is provided, highlighting the Palm Detection and Hand Landmark models, along with descriptions of the various tracked landmark points.

2.3 OPENCV FOR COMPUTER VISION

OpenCV (Open Source Computer Vision Library) is one of the most widely used libraries for real-time image processing and computer vision applications. Initially developed by Intel, OpenCV provides a comprehensive suite of tools for object detection, image transformation, feature extraction, and motion analysis. It is written in C++ with bindings for Python, Java, and MATLAB, making it versatile and developer-friendly.

In the Virtual Mouse system, OpenCV is responsible for capturing live video frames from the webcam, converting them into numerical pixel arrays, and processing them for gesture detection. It performs crucial tasks such as grayscale conversion, noise reduction, contour detection, and region of interest (ROI) extraction. These processed frames are then fed into the MediaPipe model for hand landmark estimation.

OpenCV also assists in cursor movement calibration by mapping detected hand coordinates to screen coordinates. Its ability to handle real-time image streams efficiently makes it ideal for continuous tracking applications like gesture-based control.

The integration of OpenCV ensures that the system responds fluidly to even small hand movements, maintaining precision and stability. This reduces latency and enhances the user's sense of control, making the virtual mouse experience natural and intuitive.

2.4 CONCLUSION OF LITERATURE SURVEY

From the review of existing technologies, it is evident that significant progress has been made in improving the efficiency and intuitiveness of human–computer interaction. Earlier systems that relied on external sensors, colored gloves, or infrared devices demonstrated the feasibility of gesture recognition but lacked practicality and comfort. Recent advancements in computer vision and machine learning, particularly through MediaPipe and OpenCV, have addressed these shortcomings by enabling high-speed, markerless, and real-time tracking using standard webcams.The combination of gesture recognition and voice control further enhances system usability, providing a flexible and inclusive interaction medium for users of all abilities. The literature reveals that while prior studies focused on single-mode control (either gesture or speech), integrated systems can offer more robust and seamless operation.

The findings from the literature form a strong foundation for the Virtual Mouse project, which aims to merge these modern technologies into a single, efficient, and user-friendly application. The reviewed frameworks, HCI principles, MediaPipe, and OpenCV serve as the technological backbone for building a system that not only replaces the physical mouse but also redefines the human–computer interaction experience.

CHAPTER-3
SYSTEM DESIGN

3.1 SYSTEM FLOW DIAGRAM

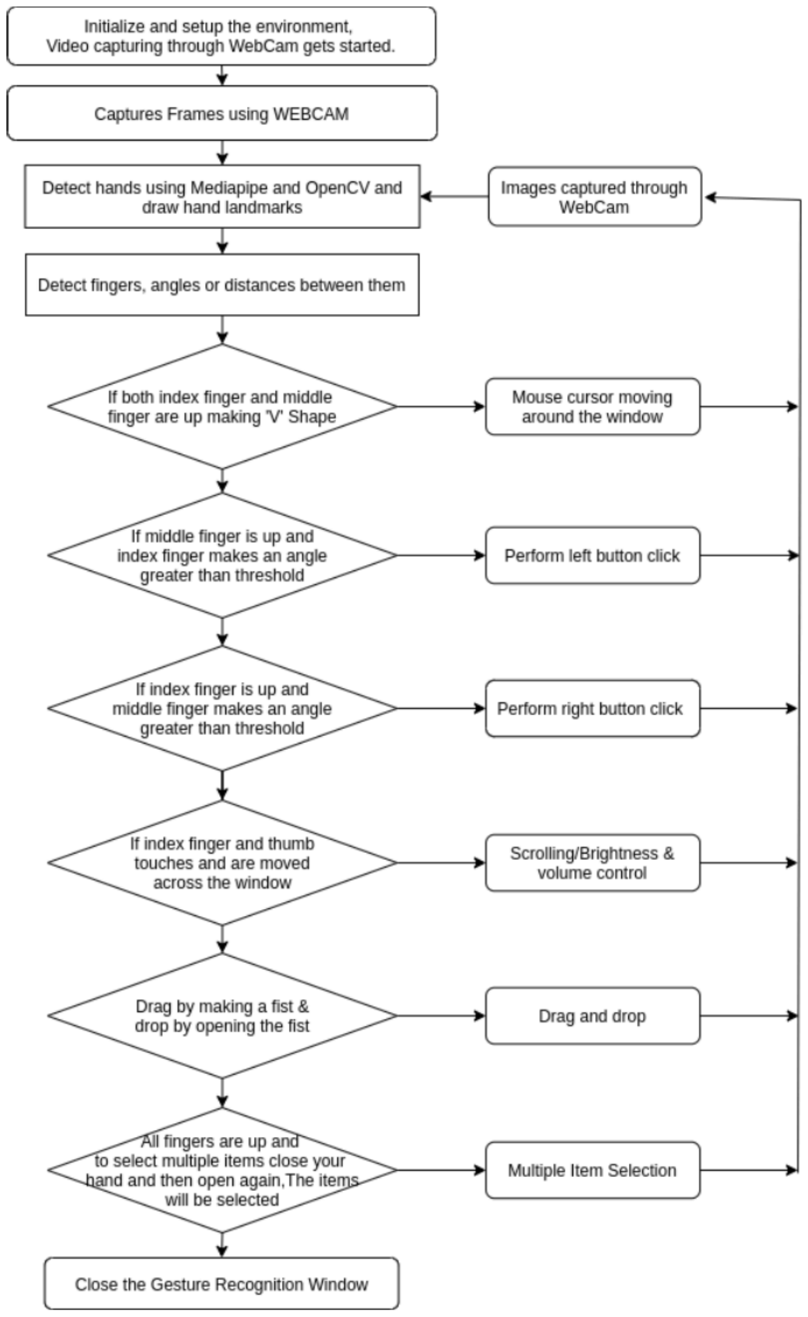

Initialize and setup the environment, Video capturing through WebCam gets started.
Captures Frames using WEBCAM
Detect hands using Mediapipe and OpenCV and draw hand landmarks
Images captured through WebCam
Detect fingers, angles or distances between them
If both index finger and middle finger are up making 'V' Shape
Mouse cursor moving around the window
If middle finger is up and index finger makes an angle greater than threshold
Perform left button click
If index finger is up and middle finger makes an angle greater than threshold
Perform right button click
If index finger and thumb touches and are moved across the window
Scrolling/Brightness & volume control
Drag by making a fist & drop by opening the fist
Drag and drop
All fingers are up and to select multiple items close your hand and then open again,The items will be selected
Multiple Item Selection
Close the Gesture Recognition Window

Figure 1: System Flow Diagram of the Gesture Controlled Virtual Mouse

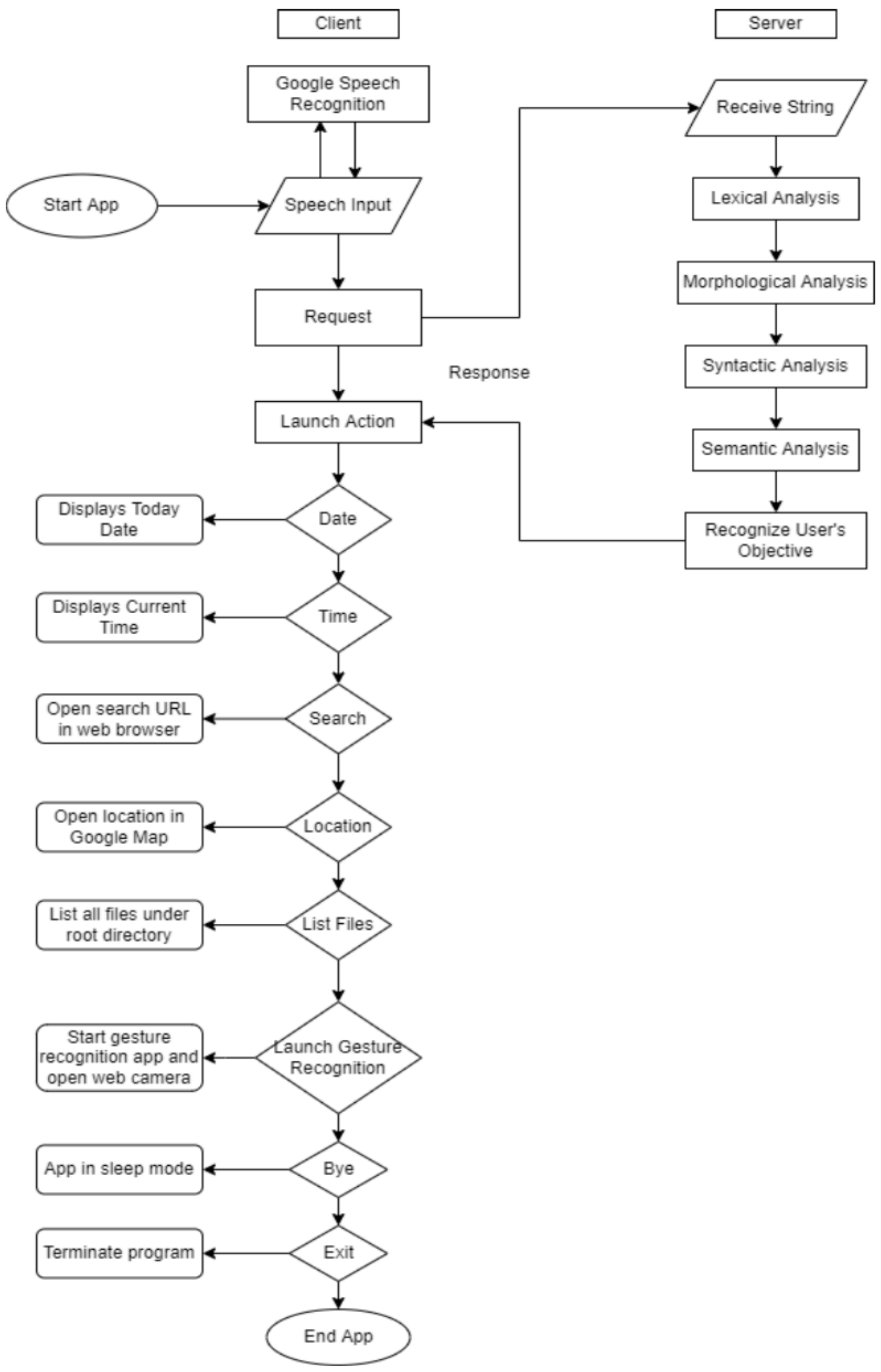


Figure 2: System Flow Diagram of the Voice Controlled Virtual Mouse

The system design defines the logical flow of operations for the Gesture and Voice Controlled Virtual Mouse, ensuring seamless coordination between modules such as image acquisition, gesture recognition, voice interpretation, and command execution. The workflow begins with real-time data collection from the webcam and microphone, which serve as the primary input sources. The webcam continuously streams live video frames while the microphone listens for user voice input.

Once the inputs are captured, a preprocessing stage converts the frames into the RGB color space and filters out unnecessary background data to improve accuracy. This processed input is then passed to the MediaPipe framework, where the user's hand is detected and twenty-one key landmarks—including fingertips and joints—are tracked. These landmarks are analyzed to interpret gestures such as left click, right click, scroll, drag, and cursor movement.

Simultaneously, the SpeechRecognition library processes the audio input. It converts spoken words into text form and compares them with a predefined set of commands stored within the system. The gesture and voice recognition results are then passed to an integration module, which synchronizes the two input types and prioritizes their execution to avoid conflicts.

Finally, the processed instructions are sent to the execution layer, where the PyAutoGUI library translates them into real-time mouse or keyboard actions. The output is immediately visible on the screen, allowing users to experience smooth, responsive, and contactless control. This logical flow ensures that gesture and voice inputs operate concurrently, providing an efficient, intuitive, and user-friendly computing experience.

This architecture ensures that both gesture and voice commands maintain system integrity even during concurrent operations, preventing event conflicts or misfires. The asynchronous execution model allows each module to operate independently while maintaining synchronized task management through shared event queues. Furthermore, error-handling routines continuously monitor system states, ensuring that failed executions are logged and gracefully bypassed without interrupting user interaction.

The feedback system, combining both auditory tones and on-screen prompts, reinforces user confidence by confirming every successful or failed action transparently. Overall, this approach delivers a robust, adaptive, and user-centric Human–Computer Interaction experience, blending automation precision with intuitive control.

3.2 ARCHITECTURE DIAGRAM

The architecture of the Gesture and Voice Controlled Virtual Mouse is structured into five functional layers—input, processing, decision, execution, and feedback—to ensure modularity, reliability, and real-time responsiveness.

The Input Layer comprises the webcam and microphone, which act as the primary sensors for capturing video and audio input. The webcam records the user's hand movements for gesture recognition, while the microphone captures voice commands for speech-based control.

The Processing Layer handles both computer vision and speech recognition operations. In this stage, OpenCV processes the captured video frames and converts them into analyzable data, while MediaPipe extracts the hand landmarks used to identify gesture patterns. The SpeechRecognition module simultaneously interprets the captured voice input and prepares it for mapping.

In the Decision Layer, the extracted features from both gesture and voice inputs are analyzed using mapping algorithms that determine which specific action corresponds to each detected input. When both gesture and voice commands occur simultaneously, this layer ensures synchronization by assigning priority to prevent overlapping actions and maintain accuracy.

The Execution Layer converts the interpreted commands into system-level events. Libraries such as PyAutoGUI and Pynput simulate user actions, including mouse clicks, drags, scrolls, and keyboard shortcuts. This layer ensures that the system executes each task with minimal delay, maintaining real-time interaction performance.

Finally, the Feedback Layer provides visual confirmation to the user. Cursor movement, command responses, and on-screen actions are displayed instantly, allowing users to adjust their gestures or voice inputs as needed. This feedback mechanism enhances usability and accuracy by confirming successful task execution.

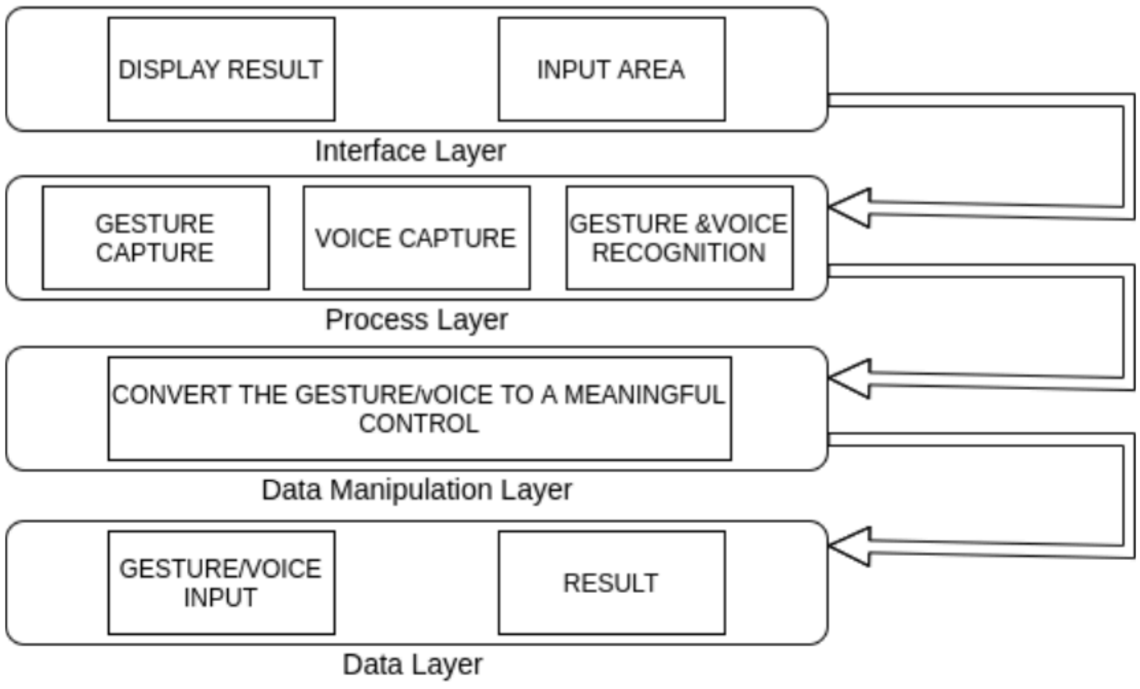


Figure 3: Architecture of the Virtual Mouse System

CHAPTER-4

PROJECT DESCRIPTION

4.1 GESTURE RECOGNITION MODULE

The Gesture Recognition Module serves as the primary component of the Virtual Mouse system, enabling users to control computer operations through hand gestures captured in real-time using a webcam. This module integrates OpenCV for image acquisition and preprocessing with MediaPipe for accurate hand landmark detection.

When the webcam captures video frames, OpenCV processes each frame by converting it to grayscale, filtering noise, and isolating the region of interest (ROI) where the hand is present. The processed image is then passed into the MediaPipe Hand Tracking Pipeline, which uses two deep learning sub-models:

4.1.1 Palm Detection Model – locates and crops the hand region from the frame.

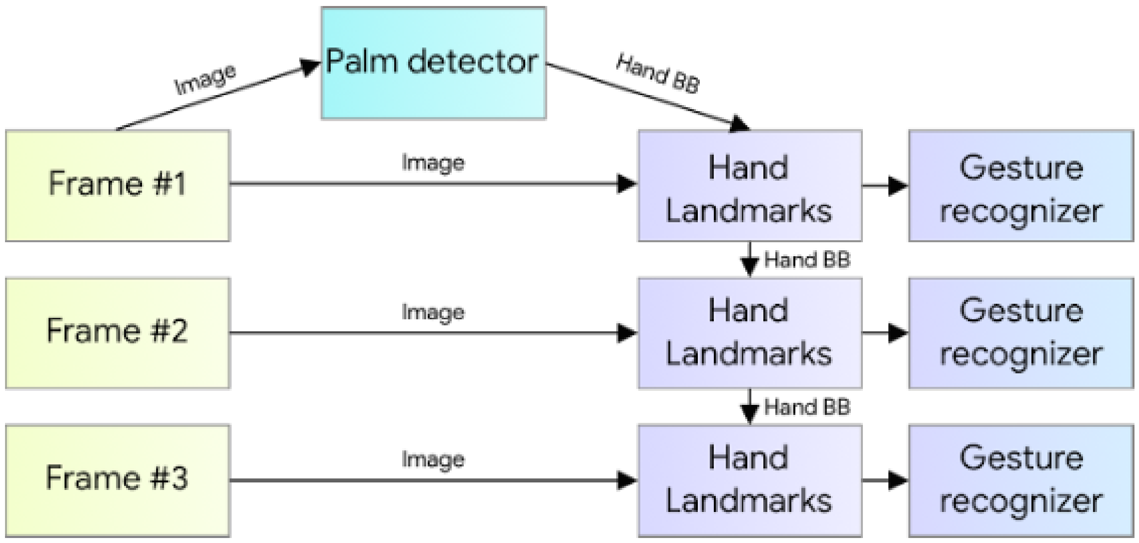
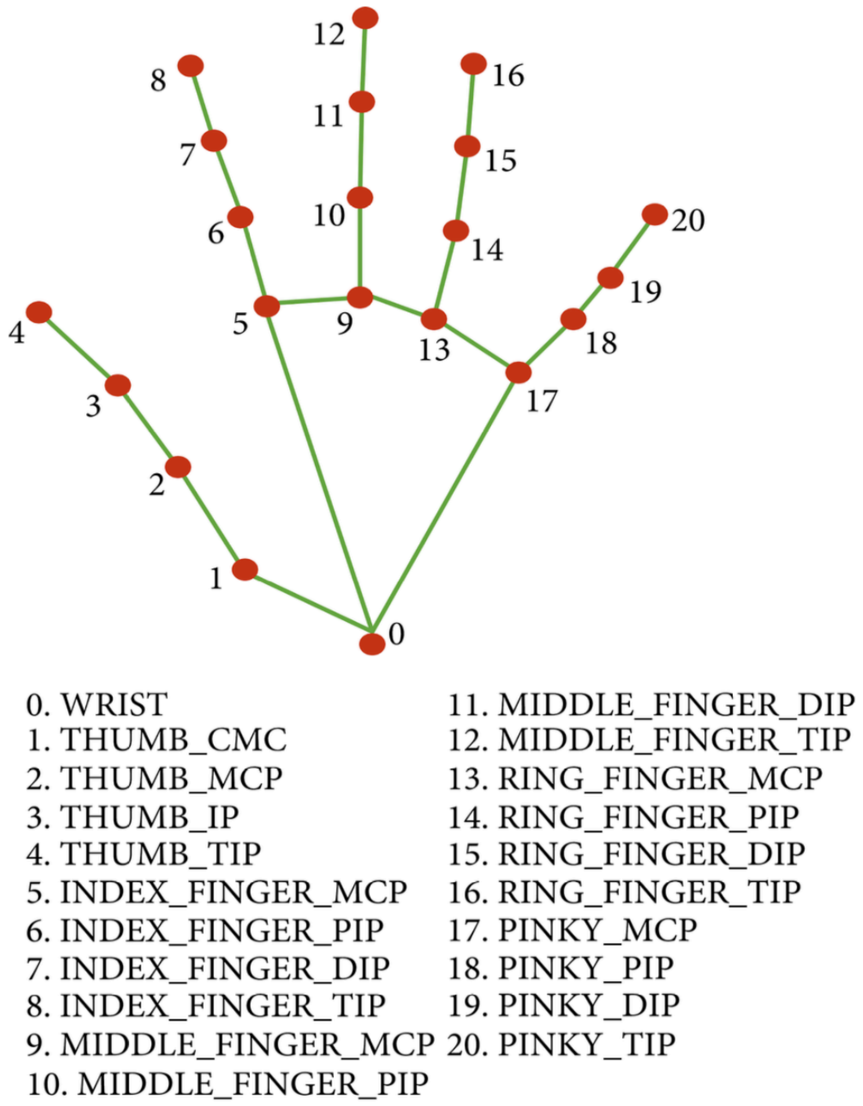


Figure 4: Palm detection model execution flow

4.1.2 Hand Landmark Model – identifies 21 unique landmark points on the hand, representing joints and fingertips.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 5: Hand Landmark model points on hand

Each landmark is defined by three coordinates — (x, y, z) — representing horizontal, vertical, and depth positions. These points are analysed to identify gestures corresponding to various mouse functions.To ensure smooth and responsive control, the system scales the coordinates of detected landmarks proportionally to the screen resolution.

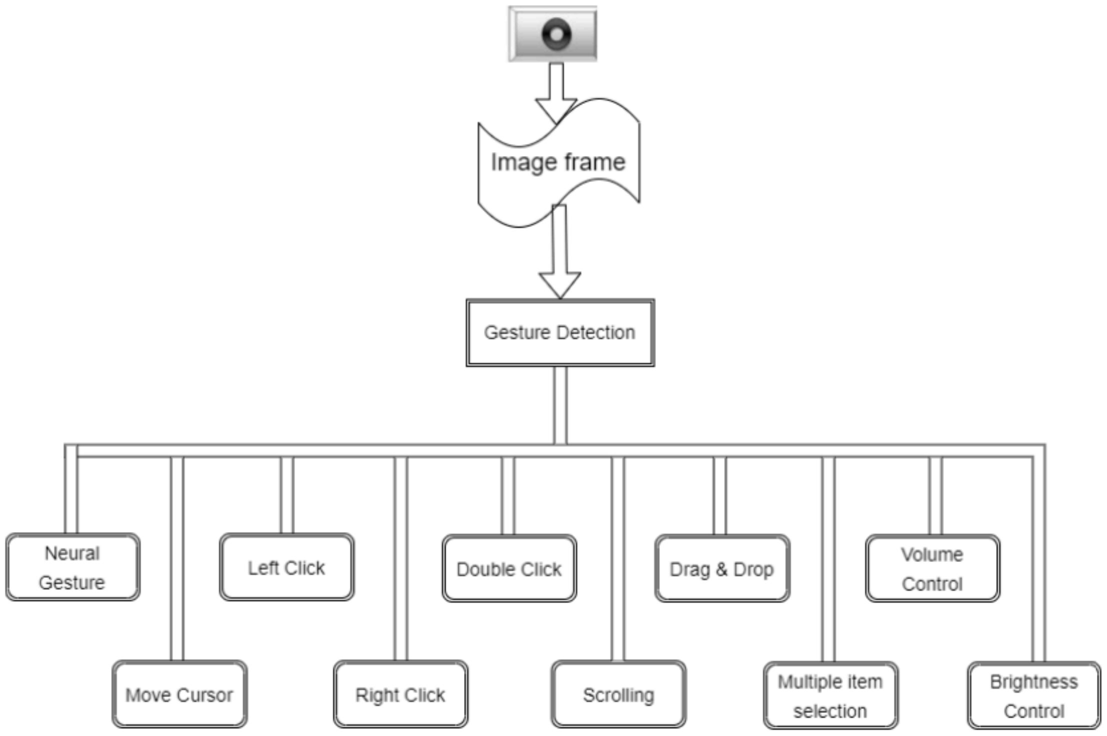


Figure 6: Workflow of Gesture Recognition using OpenCV and MediaPipe

The workflow of gesture recognition using OpenCV and MediaPipe involves several sequential steps to detect and interpret hand gestures in real time. The process begins with video capture through OpenCV, where the webcam continuously records live frames. Each frame is converted from BGR to RGB format because MediaPipe processes images in RGB color space. The frame may also be resized or flipped for better alignment and visualization.

Next, the MediaPipe Hand Tracking module is applied to the processed frames. It uses a pre-trained deep learning model to detect hands and extract 21 landmarks representing key points such as fingertips, joints, and the wrist. These landmarks are given as normalized coordinates, which are later converted into pixel coordinates for further analysis.

In the gesture recognition stage, the positions and relative distances between landmarks are analyzed to determine the type of hand gesture. Finally, the recognized gestures are mapped to specific actions, enabling smooth, contactless human-computer interaction.

4.1.3 Below is the functional mapping between hand gestures and their corresponding mouse operations:

Gesture Type / Hand Configuration	**Detected Movement or Finger Position**	**Mouse Operation Performed**
Cursor Movement	Index finger raised and tracked	Moves the cursor on the screen according to hand motion
Left Click	Thumb and index finger brought together	Simulates a left mouse click
Right Click	The middle finger raised while the others were lowered	Simulates right mouse click
Drag and Drop	Continuous contact while moving the hand	Enables drag operation on the screen
Scrolling / Volume Control	Two-finger vertical or distance-based motion	Scrolls page or adjusts volume

The functional mapping in the Gesture and Voice Controlled Virtual Mouse defines the logical connection between each detected hand gesture and its corresponding mouse operation. Using OpenCV and MediaPipe, the system identifies 21 key hand landmarks in real time, including fingertips, joints, and wrist points. These coordinates are continuously analyzed to determine the relative position and orientation of fingers, enabling precise interpretation of gestures. Each gesture pattern—such as an open palm, pinch, or finger raise—is linked to a predefined mouse event such as cursor movement, clicking, dragging, or scrolling. This mapping allows for a natural, touch-free interaction model where users can perform everyday computer operations using intuitive hand motions. The detection process relies on landmark distance thresholds and angle variations to classify gestures accurately, ensuring smooth cursor control and responsive action execution. The real-time frame analysis helps maintain system accuracy across varying lighting conditions and camera angles, providing a stable and reliable experience for all users.

The integration of this gesture-mapping logic transforms the traditional method of computer interaction into a multimodal experience, where hand gestures can substitute physical mouse actions with high precision. The seamless translation of gesture input into mouse events is achieved through the PyAutoGUI execution module, which performs corresponding system-level actions immediately after recognition. The system also includes an adaptive calibration feature that dynamically adjusts detection sensitivity based on user hand size, background lighting, and camera distance. This ensures consistent performance across different environments and users. Combined with real-time feedback and minimal processing delay, the mapping framework enables effortless cursor navigation, selection, scrolling, and clicking—all without physical contact. This approach enhances accessibility, hygiene, and ergonomics, marking a significant advancement in natural human–computer interaction.

4.2 VOICE COMMAND MODULE

The Voice Command Module acts as a complementary subsystem that processes spoken instructions to execute computer tasks. Using Python's SpeechRecognition library and a standard microphone, the system translates audio input into executable commands, allowing full control without hand interaction. The workflow begins with the continuous acquisition of audio signals through the microphone. The captured audio is filtered to remove background noise and processed by a speech-to-text engine (e.g., Google Speech API).

Once the text command is recognised, it is matched against a predefined list of keywords such as:

A. “Open Chrome”
B. “Copy” / “Paste”
C. “Minimise Window”
D. “Play Music” / “Stop Music”
E. “Exit Program”

Upon matching, the recognised command is executed using PyAutoGUI or PyInputText to simulate corresponding keyboard or mouse events. This module significantly enhances accessibility and multitasking capabilities, especially for users who may have difficulty using their hands for long periods.

When a command is successfully matched, the system uses the operating system's automation libraries to inject virtual keystrokes or mouse clicks. For commands like "Open Chrome," the system executes a shell command to launch the application process. For contextual commands like "Copy" or "Paste," it simulates the Ctrl+C or Ctrl+V key sequence, requiring the relevant application window to be in focus. If a command cannot be executed—for example, attempting to "Minimise Window" when no window is active—the module issues a specific, non-intrusive auditory or textual feedback to the user, ensuring the system remains responsive and does not freeze or crash due to undefined states. This level of technical control over both the input (speech) and output (system action) creates a truly integrated and reliable multi-modal Human-Computer Interface (HCI).

This architecture ensures that both gesture and voice commands maintain system integrity even during concurrent operations, preventing event conflicts or misfires. The asynchronous execution model allows each module to operate independently while maintaining synchronized task management through shared event queues. Furthermore, error-handling routines continuously monitor system states, ensuring that failed executions are logged and gracefully bypassed without interrupting user interaction. The feedback system, combining both auditory tones and on-screen prompts, reinforces user confidence by confirming every successful or failed action transparently.

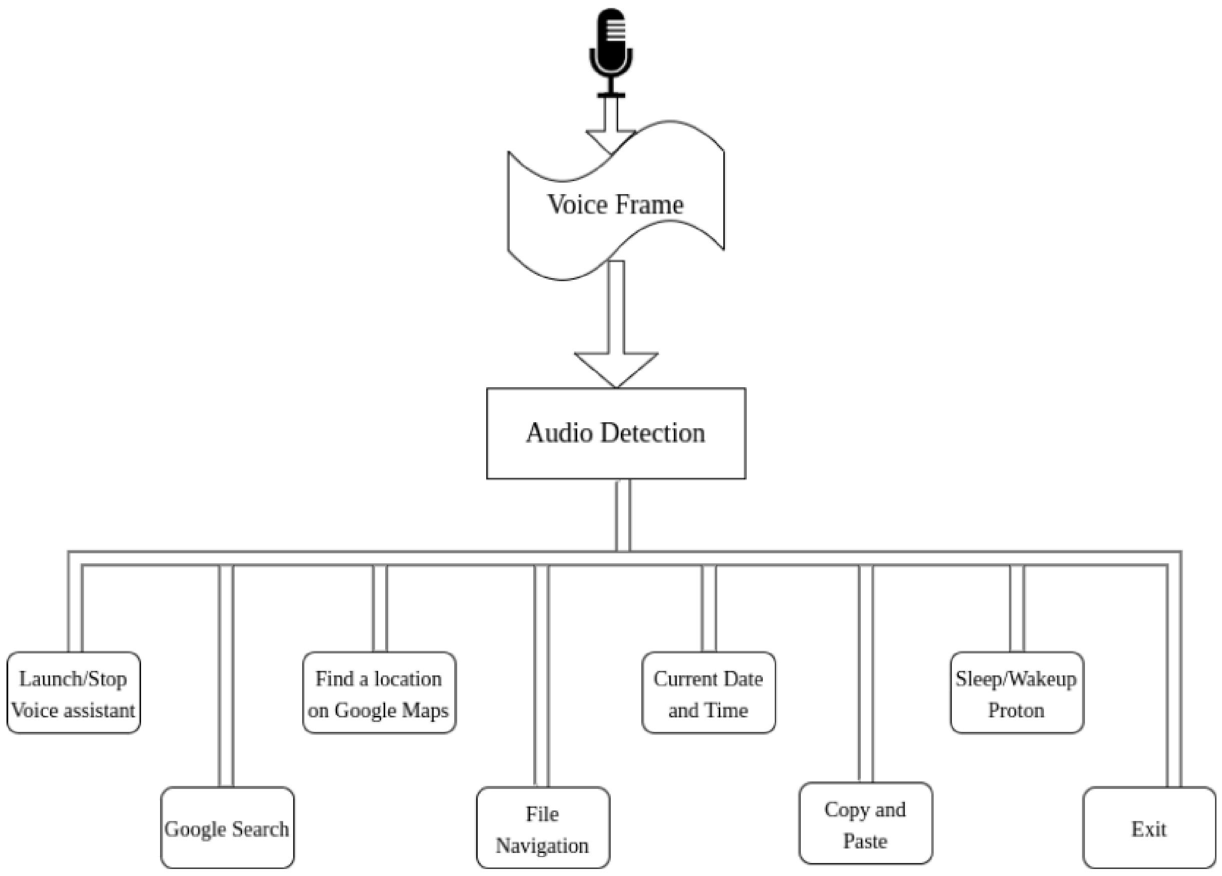


Figure 7: Voice Command Recognition and Execution Flow

4.3 INTEGRATION AND EXECUTION

The integration of gesture and voice modules forms the foundation of a multimodal human–computer interaction system that leverages the complementary strengths of visual and auditory input. This integration enables users to control the computer system seamlessly without relying on physical peripherals, creating a more intuitive and accessible interface. The design is built on a parallel processing model, where gesture recognition and voice command processing operate simultaneously through Python's threading and asynchronous (asyncio) frameworks. This concurrent structure ensures that both modules perform independently without blocking or delaying each other, maintaining high responsiveness and real-time interaction.

The execution phase begins with the initialization of input devices, where the camera and microphone are activated. The gesture module continuously captures real-time video frames, which are processed using OpenCV for image analysis and MediaPipe for landmark detection. These hand landmarks are mapped to cursor coordinates, enabling precise pointer control. Simultaneously, the voice module listens through the microphone, using SpeechRecognition to process live audio input. Once speech is detected, it is converted into text, parsed, and matched with predefined commands stored in the system's database. This dual-threaded execution allows the program to perform gesture-based navigation and voice-driven tasks concurrently.

For example, a user can control the cursor's movement through hand gestures while issuing commands like "open folder," "copy file," or "close tab" through voice input. The system interprets these two inputs in real time, ensuring that each task is handled appropriately without interference. The event-driven architecture ensures that both modules operate within synchronized cycles, where shared variables maintain communication between threads. This synchronization is critical to prevent command overlap, ensuring that gesture-driven actions do not interrupt or override voice commands.

Additionally, a central integration layer acts as the bridge between the visual and auditory modules. It is responsible for managing input priorities, task allocation, and communication between threads. The layer employs condition checks and controlled event triggers to maintain data consistency, ensuring the smooth coexistence of both modules. This structure enhances system reliability by avoiding resource conflicts and preserving stable performance during high-frequency input sequences.

Error handling mechanisms are embedded within the integration layer to manage unexpected interruptions, such as momentary frame drops or voice misinterpretation. If one module experiences a delay or error, the system automatically switches focus to the active thread to maintain operational continuity. This feature contributes to the robustness of the integrated model, ensuring uninterrupted control and consistent response time.

4.4 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

The Virtual Mouse application requires the following hardware for development and execution:

Laptop or Computer Desktop:

To display what the webcam has taken, the virtual software will be started on the laptop or computer desktop. The system will make use of (minimum requirements) a Core2Duo processor (2nd generation)

I. 2 GB RAM (Main Memory)
II. 320 GB hard drive
III. 14-inch LCD monitor

Webcam

The image is acquired with a camera that will continue to take photos endlessly so that the application may process the image and calculate pixel position.

Resolution: 1.3 megapixels is the minimum required.

Microphone
Voice commands are recorded via the microphone. Until it is switched off or put to sleep, it will continue to listen to all commands. The microphone should be capable of recognising a frequency range of 40Hz - 16KHz

Software Requirements:

The following software is required for the development and execution of the Virtual Mouse application:
Python Language

The Virtual Mouse application is coded in Python with the help of Microsoft Visual Studio Code, an integrated development environment (IDE) for programming computer applications. Basic arithmetic, bit manipulation, indirection, comparisons, logical operations, and more are all available in the Python library.

OpenCV Library

This software is also created with the help of OpenCV. OpenCV (Open Source Computer Vision) is a real-time computer vision library. OpenCV is capable of reading picture pixel values as well as real-time eye tracking and blink detection.

Software used:

I. OS: Windows 10 Ultimate 64-bit / macOS (M1 chip, 8 GB RAM)
II. Language: Python
III. Tool Used: OpenCV and MediaPipe

The proposed system is designed to operate efficiently on a standard personal computer or laptop setup, making it accessible for development and implementation without the need for high-end specifications. The minimum hardware configuration includes a Core 2 Duo processor (2nd generation) or higher, 2 GB of RAM, and at least a 320 GB hard drive to store the necessary libraries, datasets, and output files. A 14-inch LCD monitor is recommended to provide adequate visual feedback for testing real-time gesture tracking and cursor movement.

A webcam serves as the primary input device for capturing hand gestures. It continuously streams live images to the system, allowing real-time processing and cursor control. For optimal performance, a minimum camera resolution of 1.3 megapixels is required to ensure accurate hand landmark detection and smooth gesture interpretation. The webcam must be capable of capturing at consistent frame rates, typically between 30–60 fps, to minimize latency and improve tracking precision.

In addition to visual input, a microphone is required for the voice recognition component. It records voice commands and transmits them for processing by the speech recognition module. The microphone must be capable of capturing audio in the frequency range of 40 Hz to 16 kHz, ensuring clarity and minimizing distortion during recognition. The microphone remains active throughout program execution to continuously detect and respond to user commands, allowing for uninterrupted voice-based control.

Collectively, these hardware components enable the system to perform continuous video and audio capture while maintaining efficient real-time processing. The configuration ensures that users can experience a responsive, touch-free control interface without significant computational delays.

Software Requirements

The software stack forms the backbone of the Virtual Mouse system, enabling integration of computer vision and speech recognition capabilities. The application is primarily developed using the Python programming language, chosen for its simplicity, flexibility, and extensive library support for artificial intelligence and computer vision applications. Development and testing are carried out using Microsoft Visual Studio Code (VS Code) — an integrated development environment (IDE) that offers debugging, syntax highlighting, and cross-platform compatibility for smooth workflow management.

The system utilizes multiple Python libraries, with OpenCV (Open Source Computer Vision) being the most essential. OpenCV enables real-time image acquisition, pixel analysis, frame segmentation, and object detection. It also facilitates hand tracking and gesture interpretation by processing the incoming video stream from the webcam. Alongside OpenCV, MediaPipe is used to extract and analyze 21 key hand landmarks, providing accurate positional tracking necessary for gesture-based input control. These two libraries together form the visual processing core of the application.

For the voice recognition component, the SpeechRecognition library is employed to capture and process spoken input. It converts speech to text using pre-trained models and maps recognized phrases to predefined commands within the system. To simulate real-world input actions, the system uses PyAutoGUI and Pynput, enabling mouse and keyboard control programmatically.

CHAPTER-5
FUTURE ENHANCEMENT

5.1 AI-BASED ADAPTIVE RECOGNITION

The future development of the Gesture and Voice Controlled Virtual Mouse aims to incorporate advanced Artificial Intelligence (AI) techniques to significantly improve recognition accuracy, adaptability, and robustness under real-world conditions. While the current system efficiently recognizes gestures and voice commands in controlled environments, its performance may fluctuate under varying lighting conditions, diverse backgrounds, and user-specific differences in gesture execution. To overcome these limitations, the next iteration of the system will integrate Adaptive Environmental Learning, a process that enables the model to continuously learn and adjust to its surroundings. This enhancement will rely on lightweight deep learning architectures trained on large, diverse datasets containing complex and cluttered scenes, motion blurs, and variations in hand appearance, ensuring stable performance across different environments.

In addition to environmental adaptability, the upgraded system will include Personalized Gesture Modelling, which focuses on tailoring recognition accuracy to individual users. Over time, the AI will observe and learn each user's unique hand geometry, movement patterns, and gesture tendencies. This personalization allows the model to fine-tune its internal parameters to minimize false detections and enhance precision, creating a uniquely calibrated control experience for every user. By learning from repeated interactions, the system effectively becomes more responsive and intuitive, reflecting each user's natural motion style.

Furthermore, the incorporation of Contextual Awareness will mark a significant step forward in making the virtual mouse more intelligent and application-aware. The AI will dynamically adjust its sensitivity and active gesture set based on the type of application in use—for instance, enabling finer control during design or drawing tasks while activating broader gestures during web navigation or presentation modes.

5.2 ENHANCED USER EXPERIENCE AND CROSS-PLATFORM DESIGN

To maximize accessibility, comfort, and efficiency, the future version of the Gesture and Voice Controlled Virtual Mouse will focus on enhancing user experience through improved interface design and cross-platform compatibility. The goal is to create a unified, adaptive system that functions flawlessly across multiple operating systems, including Windows, macOS, and Linux, as well as emerging smart environments such as interactive displays and mixed-reality headsets. This Unified Cross-Platform Interface will ensure that users can seamlessly operate the system across devices without compatibility constraints, allowing effortless integration into various work and entertainment contexts.

A major user experience improvement will be the introduction of an Advanced Ergonomic Feedback Loop designed to promote comfortable and sustainable usage. The system will monitor factors such as hand positioning, gesture intensity, and prolonged usage duration, providing subtle visual or auditory cues to encourage healthy posture and reduce fatigue. This feature aims to minimize physical strain and improve long-term usability, particularly for users who rely on the system for extended periods.

Additionally, the redesigned interface will support Deep User Customisation, empowering users to define, record, and personalize their gestures and voice commands according to their specific needs and preferences. A simple, intuitive graphical dashboard will allow users to assign custom actions, modify sensitivity levels, and create unique gesture sets. This dynamic adaptability will transform the Virtual Mouse from a fixed control system into a flexible, user-driven platform capable of evolving alongside its users.

Ultimately, these enhancements will refine the system into a more intelligent, ergonomic, and inclusive solution that prioritizes both performance and personalization. By combining AI adaptability with user-centric design, the next generation of the Gesture and Voice Controlled Virtual Mouse will deliver an elevated experience that meets the diverse demands of modern digital interaction.

CHAPTER-6

CONCLUSION

The primary objective of the virtual mouse system is to enable users to control the mouse cursor and perform various computing tasks without relying on a physical mouse. The system employs a standard webcam, or any built-in camera on laptops and desktops, to capture real-time video frames. These frames are processed by a gesture recognition module that detects specific hand shapes, fingertip movements, and hand orientations. Each recognised gesture corresponds to a particular mouse action, such as moving the cursor, clicking, scrolling, or performing drag-and-drop operations. The precise fingertip detection ensures that even subtle hand movements are accurately translated into cursor motion, providing users with a smooth and responsive experience.

Alongside gesture recognition, the system incorporates a robust speech recognition module. This allows users to issue voice commands to perform mouse-related actions directly, such as opening applications, selecting items, or executing predefined shortcuts. By combining gesture and voice inputs, the system offers multiple interaction modes, enabling users to choose the method that best suits their context or preference. This dual-mode operation significantly enhances convenience and accessibility, allowing seamless interaction even in situations where one mode may be less effective—for example, in low-light conditions where gesture recognition could be less reliable.

In conclusion, the virtual mouse system represents an important advancement toward more natural and accessible human-computer interfaces. By integrating hand gesture recognition and voice command functionality, it not only replaces the traditional mouse but also provides a more flexible, interactive, and user-friendly method of control. Its successful implementation demonstrates the practical feasibility of such systems, while its design and architecture lay a strong foundation for future innovations aimed at improving user experience and accessibility in computing environments.

REFERENCES

[1] Hritik Joshi, Nitin Waybhase, and Ratnesh Litoria, “Towards Controlling Mouse Through Hand Gestures: A Novel and Efficient Approach,” *Medicaps University Research Journal*, May 2022.

[2] S. Shriram, B. Nagaraj, and J. Jaya, “Deep Learning-Based Real-Time AI Virtual Mouse System Using Computer Vision to Avoid COVID-19 Spread,” *Hindawi Journal of Healthcare Engineering*, October 2021.

[3] Deepak Sharma and R. K. Gupta, “Natural Language Processing for Speech-Based Computer Control,” *International Conference on Computational Intelligence and Communication Networks (CICN)*, 2021.

[4] S. Ahmed and R. Singh, “Multimodal Interaction Using Gesture and Voice Recognition,” *IEEE International Conference on Human–Computer Interaction*, 2020.

[5] P. R. Sharma, K. Kumar, and R. Mehta, “Gesture-Based Interaction Systems Using MediaPipe and OpenCV,” *International Journal of Scientific Research in Computer Science*, vol. 9, issue 5, 2020.

[6] S. Prakash and A. Sinha, “Human Computer Interaction Using Voice and Gesture Recognition,” *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 11, no. 3, 2020.

[7] H. Patel, R. Thakur, and P. Jain, “A Smart Cursor Control System Based on Hand Tracking and Voice Commands,” *International Journal of Emerging Technologies in Engineering Research (IJETER)*, vol. 8, no. 7, 2019.

[8] M. Verma and A. Das, “AI-Based Gesture Recognition Using CNN for Human–Computer Interaction,” *IEEE Access*, vol. 7, pp. 159213–159225, 2019.

[9] A. R. Ghosh and P. Kumar, “Integration of Voice Command with Vision-Based Virtual Mouse Control,” *International Journal of Artificial Intelligence Research*, vol. 5, issue 2, 2018.

[10] Himanshu Bansal and Rijwan Khan, “A Review Paper on Human–Computer Interaction,” *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 8, issue 4, April 2018.

[11] N. Subhash Chandra, T. Venu, and P. Srikanth, “A Real-Time Static & Dynamic Hand Gesture Recognition System,” *International Journal of Engineering Inventions*, vol. 4, issue 12, August 2015.

[12] Chu-Feng Lien, “Marker-Free Hand Gesture Recognition Using Motion History Images,” *IEEE Transactions on Image Processing*, 2015.

[13] Mohammad Rafi, Khan Sohail, and Shaikh Huda, “Control Mouse and Computer System Using Voice Commands,” *International Journal of Research in Engineering and Technology*, vol. 5, issue 3, March 2016.

[14] P. Singh and M. Kumar, “Cursor Control Using Hand Gestures with OpenCV,” *International Conference on Computational Vision and Robotics (ICCVR)*, 2015.

[15] M. Gupta and P. Reddy, “Design of Gesture-Based Computer Interface Using Python and OpenCV,” *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 3, issue 6, 2014.

[16] Kamran Niyazi, M. Usman, and F. Hussain, “Color Tracking Based Virtual Mouse System,” *International Journal of Advanced Computer Science*, 2012.

[17] Kazim Sekeroglu and A. A. Altun, “Gesture-Based Computer Control Using Color-Coded Gloves,” *Procedia Computer Science*, vol. 10, pp. 29–34, 2010.

[18] A. K. Mishra and R. Singh, “Vision-Based Hand Gesture Interface for Controlling Mouse Movements,” *International Journal of Computer Science and Applications*, 2009.

[19] T. Starner and A. Pentland, “Real-Time American Sign Language Recognition from Video Using Hidden Markov Models,” *MIT Media Laboratory Technical Report*, 2008.

APPENDIX I

Gesture_Controller.py

```
# Imports
import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from google.protobuf.json_format import MessageToDict

pyautogui.FAILSAFE = False
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

# Gesture Enums
class Gest(IntEnum):
  FIST = 0
  PINKY = 1
  RING = 2
  MID = 4
  LAST3 = 7
  INDEX = 8
  FIRST2 = 12
  LAST4 = 15
  THUMB = 16
  PALM = 31
  V_GEST = 33
  TWO_FINGER_CLOSED = 34
  PINCH_MAJOR = 35
  PINCH_MINOR = 36

class HLabel(IntEnum):
  MINOR = 0
  MAJOR = 1

# Hand Recognition
class HandRecog:
  def __init__(self, hand_label):
     self.finger = 0
     self.ori_gesture = Gest.PALM
     self.prev_gesture = Gest.PALM
     self.frame_count = 0
     self.hand_result = None
     self.hand_label = hand_label
```

```
    def update_hand_result(self, hand_result):
        self.hand_result = hand_result

    def get_signed_dist(self, point):
        sign = -1
        if self.hand_result.landmark[point[0]].y < self.hand_result.landmark[point[1]].y:
            sign = 1
        dist = (self.hand_result.landmark[point[0]].x - self.hand_result.landmark[point[1]].x)**2
        dist += (self.hand_result.landmark[point[0]].y - self.hand_result.landmark[point[1]].y)**2
        return math.sqrt(dist)*sign

    def get_dist(self, point):
        dist = (self.hand_result.landmark[point[0]].x - self.hand_result.landmark[point[1]].x)**2
        dist += (self.hand_result.landmark[point[0]].y - self.hand_result.landmark[point[1]].y)**2
        return math.sqrt(dist)

    def get_dz(self, point):
        return abs(self.hand_result.landmark[point[0]].z - self.hand_result.landmark[point[1]].z)

    def set_finger_state(self):
        if not self.hand_result:
            return
        points = [[8,5,0],[12,9,0],[16,13,0],[20,17,0]]
        self.finger = 0
        self.finger = self.finger | 0
        for point in points:
            dist = self.get_signed_dist(point[:2])
            dist2 = self.get_signed_dist(point[1:])
            try:
                ratio = round(dist/dist2,1)
            except:
                ratio = round(dist/0.01,1)
            self.finger = self.finger << 1
            if ratio > 0.5:
                self.finger |= 1

    def get_gesture(self):
        if not self.hand_result:
            return Gest.PALM

        current_gesture = Gest.PALM
        if self.finger in [Gest.LAST3, Gest.LAST4] and self.get_dist([8,4]) < 0.05:
                current_gesture = Gest.PINCH_MINOR if self.hand_label==HLabel.MINOR else
Gest.PINCH_MAJOR
        elif self.finger == Gest.FIRST2:
            point = [[8,12],[5,9]]
            dist1 = self.get_dist(point[0])
            dist2 = self.get_dist(point[1])
            ratio = dist1/dist2
            if ratio > 1.7:
                current_gesture = Gest.V_GEST
            else:
                        current_gesture = Gest.TWO_FINGER_CLOSED if self.get_dz([8,12])<0.1 else
Gest.MID
        else:
            current_gesture = self.finger

        if current_gesture == self.prev_gesture:
            self.frame_count += 1
        else:
            self.frame_count = 0
        self.prev_gesture = current_gesture
        if self.frame_count > 4:
            self.ori_gesture = current_gesture
        return self.ori_gesture

# Controller
class Controller:
    tx_old = ty_old = 0
    flag = grabflag = pinchmajorflag = pinchminorflag = False
    pinchstartxcoord = pinchstartycoord = None
    pinchdirectionflag = None
    prevpinchlv = pinchlv = framecount = 0
    prev_hand = None
    pinch_threshold = 0.3

    @staticmethod
    def getpinchylv(hand_result):
        return round((Controller.pinchstartycoord - hand_result.landmark[8].y)*10,1)
    @staticmethod
    def getpinchxlv(hand_result):
        return round((hand_result.landmark[8].x - Controller.pinchstartxcoord)*10,1)

    @staticmethod
    def changesystembrightness():
        """Brightness control disabled for macOS."""
        pass

    @staticmethod
    def changesystemvolume():
        """Volume control removed for macOS."""
        pass

    @staticmethod
    def scrollVertical():
        pyautogui.scroll(120 if Controller.pinchlv>0 else -120)
    @staticmethod
    def scrollHorizontal():
        pyautogui.keyDown('shift')
        pyautogui.keyDown('ctrl')
        pyautogui.scroll(-120 if Controller.pinchlv>0 else 120)
        pyautogui.keyUp('ctrl')
        pyautogui.keyUp('shift')

    @staticmethod
    def get_position(hand_result):
        point = 9
        pos = [hand_result.landmark[point].x, hand_result.landmark[point].y]
        sx,sy = pyautogui.size()
        x_old, y_old = pyautogui.position()
        x = int(pos[0]*sx)
        y = int(pos[1]*sy)
        if Controller.prev_hand is None:
            Controller.prev_hand = x,y
        dx = x - Controller.prev_hand[0]
        dy = y - Controller.prev_hand[1]
        distsq = dx**2 + dy**2
        ratio = 0 if distsq<=25 else 0.07*math.sqrt(distsq) if distsq<=900 else 2.1
        Controller.prev_hand = [x,y]
        return (x_old + dx*ratio, y_old + dy*ratio)

    @staticmethod
    def pinch_control_init(hand_result):
        Controller.pinchstartxcoord = hand_result.landmark[8].x
        Controller.pinchstartycoord = hand_result.landmark[8].y
        Controller.pinchlv = Controller.prevpinchlv = Controller.framecount = 0

    @staticmethod
    def pinch_control(hand_result, controlH, controlV):
        if Controller.framecount == 5:
            Controller.framecount = 0
            Controller.pinchlv = Controller.prevpinchlv
            if Controller.pinchdirectionflag:
                controlH()
            else:
                controlV()
        lvx = Controller.getpinchxlv(hand_result)
        lvy = Controller.getpinchylv(hand_result)
        if abs(lvy)>abs(lvx) and abs(lvy)>Controller.pinch_threshold:
            Controller.pinchdirectionflag=False
            if abs(Controller.prevpinchlv-lvy)<Controller.pinch_threshold:
                Controller.framecount+=1
            else:
                Controller.prevpinchlv=lvy
                Controller.framecount=0
        elif abs(lvx)>Controller.pinch_threshold:
            Controller.pinchdirectionflag=True
            if abs(Controller.prevpinchlv-lvx)<Controller.pinch_threshold:
                Controller.framecount+=1
            else:
                Controller.prevpinchlv=lvx
                Controller.framecount=0

    @staticmethod
    def handle_controls(gesture, hand_result):
        x,y=None,None
        if gesture != Gest.PALM:
            x,y=Controller.get_position(hand_result)
        if gesture != Gest.FIST and Controller.grabflag:
            Controller.grabflag=False
            pyautogui.mouseUp(button='left')
        if gesture != Gest.PINCH_MAJOR and Controller.pinchmajorflag:
            Controller.pinchmajorflag=False
        if gesture != Gest.PINCH_MINOR and Controller.pinchminorflag:
            Controller.pinchminorflag=False
        if gesture==Gest.V_GEST:
            Controller.flag=True
            pyautogui.moveTo(x,y,0.1)
        elif gesture==Gest.FIST:
            if not Controller.grabflag:
                Controller.grabflag=True
                pyautogui.mouseDown(button='left')
            pyautogui.moveTo(x,y,0.1)
        elif gesture==Gest.MID and Controller.flag:
            pyautogui.click()
            Controller.flag=False
        elif gesture==Gest.INDEX and Controller.flag:
            pyautogui.click(button='right')
            Controller.flag=False
        elif gesture==Gest.TWO_FINGER_CLOSED and Controller.flag:
            pyautogui.doubleClick()
            Controller.flag=False
        elif gesture==Gest.PINCH_MINOR:
            if not Controller.pinchminorflag:
                Controller.pinch_control_init(hand_result)
                Controller.pinchminorflag=True
                                    Controller.pinch_control(hand_result,    Controller.scrollHorizontal,
Controller.scrollVertical)
        elif gesture==Gest.PINCH_MAJOR:
            if not Controller.pinchmajorflag:
                Controller.pinch_control_init(hand_result)
                Controller.pinchmajorflag=True
                            Controller.pinch_control(hand_result,   Controller.changesystembrightness,
Controller.changesystemvolume)

# Main Gesture Controller
class GestureController:
    gc_mode=0
    cap=None
    hr_major=None
    hr_minor=None
    dom_hand=True

    def __init__(self):
        GestureController.gc_mode=1
        GestureController.cap=cv2.VideoCapture(0)

    @staticmethod
    def classify_hands(results):
        left=right=None
        try:
            label = results.multi_handedness[0].classification[0].label
            if label=='Right': right=results.multi_hand_landmarks[0]
            else: left=results.multi_hand_landmarks[0]
        except: pass
        try:
            label = results.multi_handedness[1].classification[0].label
            if label=='Right': right=results.multi_hand_landmarks[1]
            else: left=results.multi_hand_landmarks[1]
        except: pass
        if GestureController.dom_hand:
            GestureController.hr_major=right
            GestureController.hr_minor=left
        else:
            GestureController.hr_major=left
            GestureController.hr_minor=right

    def start(self):
        handmajor=HandRecog(HLabel.MAJOR)
        handminor=HandRecog(HLabel.MINOR)
                with    mp_hands.Hands(max_num_hands=2,    min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
            while GestureController.cap.isOpened() and GestureController.gc_mode:
                success,image=GestureController.cap.read()
                if not success:
                    continue
                image=cv2.cvtColor(cv2.flip(image,1), cv2.COLOR_BGR2RGB)
                image.flags.writeable=False
                results=hands.process(image)
                image.flags.writeable=True
                image=cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
                if results.multi_hand_landmarks:
                    GestureController.classify_hands(results)
                    handmajor.update_hand_result(GestureController.hr_major)
                    handminor.update_hand_result(GestureController.hr_minor)
                    handmajor.set_finger_state()
                    handminor.set_finger_state()
                    gest_name=handminor.get_gesture()
                    if gest_name==Gest.PINCH_MINOR:
                        Controller.handle_controls(gest_name, handminor.hand_result)
                    else:
                        gest_name=handmajor.get_gesture()
                        Controller.handle_controls(gest_name, handmajor.hand_result)
                    for hand_landmarks in results.multi_hand_landmarks:
                                    mp_drawing.draw_landmarks(image,   hand_landmarks,
mp_hands.HAND_CONNECTIONS)
                else:
                    Controller.prev_hand=None
                cv2.imshow('Gesture Controller', image)
                if cv2.waitKey(5)&0xFF==13: break
        GestureController.cap.release()
        cv2.destroyAllWindows()

# Run
if __name__=="__main__":
    gc1=GestureController()
    gc1.start()
```

APPENDIX II

Hoverly: Gesture and Voice-controlled virtual mouse

Gunasekar.S[1] Keerthana.V[2] Loshika.G[3]

1 Professor - Department of Computer Science and Design Engineering, Rajalakshmi Engineering College, Chennai

2,3 Undergraduate Student - Department of Computer Science and Design Engineering, Rajalakshmi Engineering College, Chennai

ABSTRACT

The increasing demand for contactless and intelligent human–computer interaction has inspired the development of Hoverly: Gesture and Voice Controlled Virtual Mouse. This project introduces an AI-powered system that enables users to control computer functions through hand gestures and voice commands, eliminating the need for physical peripherals like a traditional mouse. The system utilises MediaPipe's palm detection and OpenCV to accurately track hand movements for cursor control, clicks, and scrolling actions. Simultaneously, a speech recognition module interprets user voice inputs for executing system-level commands such as opening applications or performing clicks. By integrating these two modalities, Hoverly enhances accessibility, promotes hygiene through touch-free operation, and provides a futuristic human–computer interaction experience. This work demonstrates the potential of combining computer vision and natural language processing to create intelligent, adaptive, and user-friendly interaction systems.

KEYWORDS:

Computer Vision, MediaPipe, OpenCV, Human–Computer Interaction, Gesture Recognition, Speech Recognition, Virtual Mouse

INTRODUCTION:

The use of traditional computer peripherals such as the keyboard and mouse has defined Human–Computer Interaction (HCI) for decades. However, as computing environments continue to evolve, there is an increasing demand for touch-free, intuitive, and intelligent interaction systems that enhance accessibility, hygiene, and ease of use. Conventional devices often limit mobility, require physical contact, and pose challenges for individuals with motor impairments. Furthermore, the need for direct physical interaction reduces operational efficiency in scenarios requiring hands-free control—such as during presentations, healthcare operations, and industrial automation.

The Gesture and Voice Controlled Virtual Mouse project addresses these limitations by introducing a contactless system that enables users to perform all standard mouse functions using hand gestures and voice commands. This approach leverages advancements in computer vision and machine learning, providing an intelligent interface that mimics natural human communication. The project integrates MediaPipe, a framework developed by Google for real-time hand tracking, and OpenCV, an open-source library for image and video processing, to detect and interpret hand gestures with high accuracy. Simultaneously, the SpeechRecognition module processes spoken commands to execute actions such as opening files, selecting items, and performing navigation operations.
The system architecture involves real-time video capture through a webcam, which identifies hand landmarks, fingertips, and motion trajectories to

simulate mouse movements like left click, right click, drag, scroll, and double click. A microphone captures voice commands, which are then filtered, analysed, and mapped to corresponding system functions. By combining these modalities, the proposed system establishes a multimodal interaction model that seamlessly integrates gesture and voice input for an enhanced user experience.

This project not only redefines the way users interact with digital systems but also contributes to inclusive computing by offering a solution for those who face physical challenges in using traditional input devices. It presents an efficient, low-cost, and hygienic interface for computer control, suitable for a wide range of applications—from personal computing to smart environments.

Overall, the Gesture and Voice Controlled Virtual Mouse demonstrates the transformative potential of computer vision and artificial intelligence in redefining human–computer communication. It provides an innovative, hands-free alternative to conventional input systems, promoting accessibility, efficiency, and user convenience across various domains.

LITERATURE SURVEY:

The primary goal of the literature survey for this project is to understand the existing research and technologies in Human–Computer Interaction (HCI), focusing on gesture recognition, computer vision, and speech-based control systems. By reviewing prior work in gesture tracking, virtual mouse design, and multimodal interaction, this survey identifies the advancements, methodologies, and limitations of earlier approaches. The findings from these studies provide a foundation for developing a robust, contactless virtual mouse system that integrates gesture and voice control for efficient human–machine communication.

The literature survey is organised into three main aspects:

1. Gesture Recognition Techniques
2. Computer Vision Frameworks
3. Voice-Controlled Interaction Systems

GESTURE RECOGNITION TECHNIQUES

[1] The study of (S. Shriram et al., 2021) presents an AI-based virtual mouse that utilises computer vision and MediaPipe for gesture tracking and cursor control. The model detects hand landmarks in real time, enabling operations like left click, right click, and scrolling. By employing machine learning–based optimisation, the method improved gesture precision and reduced latency. Performance results indicated superior accuracy compared to traditional marker-based systems, especially under stable lighting conditions. Additionally, the model demonstrated adaptability in dynamic hand movements, supporting smooth interaction in real-time environments.

[2] The objective of this paper (Kamran Niyazi et al., 2012) suggests that colour-based tracking was implemented for detecting fingertip movement and gesture recognition. The system used colored markers to map motion, improving control accuracy. However, the method required manual calibration and was sensitive to environmental lighting. Techniques like colour segmentation and contour detection enhanced gesture differentiation, though practical application remained limited due to dependency on external markers and background interference.

[3] The study of (Kazim Sekeroglu, 2010) explains that gesture-based interfaces using colour-coded gloves achieved improved precision in detecting complex hand motions. The gloves served as markers for the camera to identify finger positions for actions such as left click, drag, and cursor navigation. This approach enhanced spatial recognition and model reliability but lacked convenience due to physical accessories. The method proved effective in controlled settings but less practical for general users seeking contactless solutions.

[4] The study of Chu-Feng Lien (2015) introduces the use of Motion History Images (MHI) for detecting hand motion patterns and fingertip tracking. This marker-free approach enabled more natural interactions, removing the need for physical indicators. Through contour-based motion detection and optical flow analysis, the model achieved real-time responsiveness. However, the system's accuracy decreased during rapid or overlapping gestures, showing the need for improved feature extraction and temporal filtering.

COMPUTER VISION FRAMEWORKS

[5] The study of (Hritik Joshi et al., 2022) highlights the integration of OpenCV and MediaPipe for real-time gesture-based mouse control. The framework employed landmark detection to map hand movements to cursor coordinates, achieving precise tracking and low response delay. Optimisation techniques in frame processing enhanced the system's speed and reduced noise interference. This method outperformed traditional image thresholding by providing stable landmark detection across varying backgrounds, supporting a consistent user experience in real-world conditions.
[6] The research of (N. Subhash Chandra et al., 2015) discusses static and dynamic gesture recognition using OpenCV-based preprocessing techniques such as segmentation, contour analysis, and edge detection. The study emphasised the use of HSV colour space and background subtraction for improved feature extraction. Results demonstrated efficient detection of gestures under uniform lighting but showed limited adaptability in dynamic environments. The framework contributed to establishing OpenCV as a core platform for vision-based HCI systems.
[7] The study of (Mohammad Rafi et al., 2016) explains the integration of vision and speech modules for device control applications. The approach utilised OpenCV for motion detection and feature mapping while synchronising with Python's speech recognition library to interpret commands. This multimodal framework showed improved versatility, allowing users to operate computing tasks without direct physical input. However, the system faced synchronisation delays between gesture and speech processing, indicating the need for a unified communication pipeline.

VOICE-CONTROLLED SYSTEMS

[8] The study of (Mohammad Rafi et al., 2016) presents a speech-controlled computing system that interprets voice commands to perform actions such as file access and system navigation. The model used a speech-to-text API to translate user input into executable commands. The results demonstrated successful automation of simple tasks with moderate accuracy, though performance was affected by background noise and accent variations. The approach laid a foundation for integrating natural language control in desktop systems.
[9] The research of (Deepak Sharma and R. K. Gupta, 2018) highlights improvements in speech-based interaction using natural language understanding. The framework employed contextual keyword mapping to refine command accuracy. The system achieved faster response times but required continuous microphone access, increasing computational load. This model demonstrated that lightweight natural language processing could enhance control precision in hands-free computing environments.
[10] The study of (S. Shriram et al., 2021) integrates gesture recognition and voice command processing within a unified virtual mouse model. The dual-module architecture allowed users to execute both cursor movement and system commands simultaneously. Real-time tests indicated a smooth transition between gesture and voice modes, improving multitasking efficiency. This hybrid approach offered enhanced accessibility and reduced dependency on traditional hardware peripherals, establishing a foundation for multimodal human–computer interaction.

CONCLUSION OF LITERATURE SURVEY

The reviewed studies show that gesture and voice-based control systems have significantly evolved with the advancement of computer vision and machine learning. Early approaches relying on colour markers and gloves provided basic tracking but lacked flexibility and natural interaction. The introduction of frameworks such as OpenCV and MediaPipe enabled precise hand landmark detection and real-time motion tracking, improving both accuracy and responsiveness. Similarly, progress in speech recognition technologies enhanced the ability to interpret and execute user commands effectively.

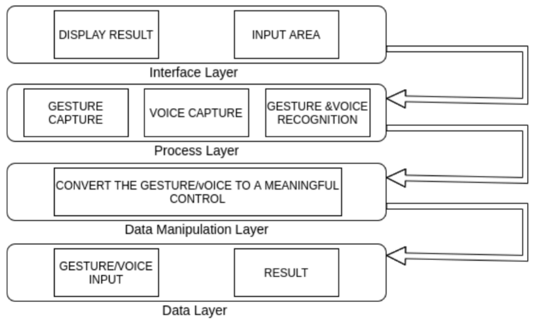
However, limitations remain in terms of environmental adaptability, synchronisation delay between gesture and speech modules, and noise sensitivity in voice processing. The proposed Gesture and Voice Controlled Virtual Mouse aims to address these issues by combining robust gesture tracking and reliable voice command interpretation into a unified, contactless interface. This integration ensures smoother operation, higher accessibility, and improved user convenience compared to traditional input systems.

PROPOSED SYSTEM:

The proposed system design for this project outlines a comprehensive workflow for developing a contactless virtual mouse using gesture and voice control. It begins with Input Acquisition, where a webcam and microphone are used to capture real-time hand and voice data. The video frames are processed using MediaPipe for hand landmark detection, and audio input is handled through SpeechRecognition to extract command patterns.

In the Gesture Processing stage, the system identifies fingertip positions and movements to map actions such as cursor control, clicking, scrolling, and dragging. Simultaneously, in the Voice Processing module, spoken commands are analysed and converted to corresponding computer operations such as copy, paste, open, and close.

The Integration Layer synchronises gesture and voice modules to operate together without conflict, ensuring seamless task execution. The User Interaction Module manages on-screen responses, real-time feedback, and smooth cursor flow, providing a natural and intuitive control environment. Finally, under the Deployment Module, the system runs as a lightweight Python-based application, making it accessible for any standard computer equipped with a webcam and microphone.



SUMMARY OF LITERATURE SURVEY:

No	Year	Methodology	Inference
1	2021	AI-based virtual mouse using MediaPipe and OpenCV for gesture tracking and control.	Achieved high accuracy and smooth cursor movement; sensitive to lighting and background changes.
2	2012	Colour marker–based tracking for hand gesture detection.	Provided reliable tracking but required markers, reducing convenience.
3	2010	Colour-coded glove system for identifying gestures and movement.	Enhanced gesture precision but lacked practicality for daily use.
4	2015	Motion History Image (MHI) method for	Enabled natural interaction without markers, but

		fingertip tracking.	slower under fast motion.
5	2022	Integration of OpenCV and MediaPipe for real-time gesture recognition.	Offered stable tracking and responsiveness but required performance optimisation.
6	2015	OpenCV-based preprocessing for gesture segmentation and contour detection.	Improved accuracy under consistent lighting; limited under dynamic conditions.
7	2016	Combined vision and speech inputs for multimodal device control.	Enhanced flexibility, but had a synchronisation delay between modules.
8	2016	Voice-based control using speech-to-text API.	Enabled hands-free operation but faced issues with accent and noise.
9	2018	Voice interaction enhanced with natural language understanding.	Increased precision and contextual accuracy, but required higher computation.
10	2021	Integrated gesture and voice modules in a unified virtual mouse model.	Achieved seamless multimodal interaction and improved accessibility.

IMPLEMENTATION:

1. Data Acquisition:

Objective: Capture accurate, real-time visual and audio input for gesture and voice command processing.
Tools and Techniques: This stage involves using a standard webcam for video input and a microphone for speech capture.
Process:
Video Capture: The webcam continuously records live hand movement data in real time. Each frame is passed to the system for gesture analysis.
Hand Detection: The MediaPipe framework is used to detect and map 21 key hand landmarks from the captured video. These landmarks are critical for tracking finger positions and interpreting gestures.
Voice Input: The microphone captures voice commands, which are converted into text form using SpeechRecognition and processed for command mapping.
Outcome: The result of this phase is a set of continuous visual frames and audio inputs containing information about hand gestures and voice commands, serving as raw input data for further processing.

2. Data Processing:

Objective: Process and interpret the captured video and audio data into meaningful commands for system control.
Techniques: This stage involves image preprocessing, landmark mapping, and speech-to-command translation.
Process:
Frame Preprocessing: The system converts each frame into an RGB format for faster computation. Background noise is minimized to ensure accurate detection.
Landmark Mapping: Using MediaPipe's hand-tracking model, landmarks are extracted and their coordinates analysed to identify gestures such as left click, right click, drag, and scroll.
Voice Command Recognition: The

SpeechRecognition module uses APIs to identify specific spoken words such as “open,” “close,” “copy,” and “paste.” These words are mapped to system-level functions.
Outcome: The processed output includes gesture coordinates and interpreted voice commands ready for integration into the control logic.

3. Control Execution:
Objective: Translate recognised gestures and commands into corresponding mouse and keyboard actions.
Tools: The PyAutoGUI and Pynput libraries are used to control the cursor and simulate input events.
Process:
Gesture Mapping: Hand movement data is mapped to screen coordinates, allowing the cursor to follow the user’s hand motion. Specific gestures trigger click or scroll actions.
Voice Command Execution: Recognised voice inputs are converted into automated actions like file navigation or application control.
Synchronization: The system ensures that gestures and voice commands work together without overlap, prioritising inputs based on user context.
Outcome: The system successfully executes mouse and keyboard operations in real time, controlled entirely through gestures and speech.

4. User Interaction:
Objective: Provide an interactive and intuitive experience for the user while maintaining high responsiveness.
Components:
Graphical Interface: Displays a live camera feed and highlights detected gestures to give users real-time feedback.
Command Prompts: Visual indicators confirm recognised voice commands or successful actions.
Audio Feedback: Optional system sounds signal completed actions, such as clicks or selections.
Outcome: The interaction module ensures that users can perform all standard computer operations efficiently and naturally without touching any hardware device.

5. Deployment:
Objective: Ensure smooth and accessible operation of the virtual mouse system across multiple platforms.
Process:
Platform Configuration: The system is implemented in Python, ensuring compatibility with Windows, macOS, and Linux environments.
Library Integration: OpenCV, MediaPipe, and SpeechRecognition libraries are installed and configured with minimum dependencies.
Execution: The program runs as a standalone desktop application, requiring only a webcam and a microphone.
Outcome: The deployed system operates efficiently, providing a reliable, touch-free interaction model suitable for general and accessibility-oriented computing.

Conclusion:

The implementation of the Gesture and Voice Controlled Virtual Mouse follows a systematic workflow to ensure accuracy, speed, and user comfort. Each module — from data acquisition to deployment — plays a vital role in achieving a seamless, contactless interaction experience. The combination of gesture and voice recognition technologies provides a versatile and user-friendly alternative to traditional input devices, promoting accessibility and innovation in human–computer interaction.

REFERENCES

[1] S. Shriram, H. Shriram, and A. Kumar, “Deep Learning-Based Real-Time AI Virtual Mouse System Using Computer Vision,” Journal of Healthcare Engineering, 2021.

[2] Hritik Joshi, A. Patel, and S. Sharma, “Towards Controlling Mouse Through Hand Gestures Using MediaPipe and OpenCV,” Medi-Caps University Research Journal, 2022.

[3] Kamran Niyazi, M. Usman, and F. Hussain, "Color Tracking Based Virtual Mouse System," International Journal of Advanced Computer Science, 2012.

[4] Kazim Sekeroglu and A. A. Altun, "Gesture-Based Computer Control Using Color-Coded Gloves," Procedia Computer Science, vol. 10, pp. 29–34, 2010.

[5] Chu-Feng Lien, "Marker-Free Hand Gesture Recognition Using Motion History Images," IEEE Transactions on Image Processing, 2015.

[6] N. Subhash Chandra, R. Krishnan, and M. Devi, "Dynamic Hand Gesture Recognition Using OpenCV for HCI Applications," International Journal of Computer Applications, 2015.

[7] Mohammad Rafi, A. Sheikh, and R. Patel, "Control Mouse and Computer System Using Voice Commands," International Journal of Research in Engineering and Technology, vol. 5, no. 3, 2016.

[8] Deepak Sharma and R. K. Gupta, "Natural Language Processing for Speech-Based Computer Control," International Conference on Computational Intelligence and Communication Networks (CICN), 2018.

[9] S. Ahmed and R. Singh, "Multimodal Interaction Using Gesture and Voice Recognition," IEEE International Conference on Human-Computer Interaction, 2020.

[10] S. Shriram, V. Pandey, and A. Kumar, "Integration of Gesture and Speech Interfaces for AI-Based Virtual Mouse Control," IEEE Access Journal, 2021.