

# **KYAAT - THE KINESICS RECOGNITION**

## **A FINAL YEAR PROJECT REPORT**

*Submitted by*

<b>BOOBALAN V</b>	<b>311820205008</b>
<b>IMAAM JAFFAR J</b>	<b>311820205010</b>
<b>MOHAMED GHAZALI P</b>	<b>311820205019</b>

*In partial fulfilment for the award of the degree  
of*

**BACHELOR OF TECHNOLOGY**

*in*

**INFORMATION TECHNOLOGY**

**MOHAMED SATHAK A.J. COLLEGE OF ENGINEERING  
SIRUSERI IT PARK, OMR, CHENNAI-603 103**



**ANNA UNIVERSITY: CHENNAI 600 025**

**MAY 2024**

# **ANNA UNIVERSITY: CHENNAI 600 025**

## **BONAFIDE CERTIFICATE**

It is certified that this project report “**KYAAT - The Kinesics Recognition**” is the bonafide work of “**BOOBALAN V (311820205008), IMAAM JAFFAR J (311820205010), MOHAMED GHAZALI P (311820205019)**” who carried out the project work under my supervision.

### **SIGNATURE**

Dr.D.PRAKASH, M.E., Ph.D.,

### **HEAD OF THE DEPARTMENT**

ASSOCIATE PROFESSOR

Department of Information  
Technology

Mohamed Sathak AJ College of  
Engineering, Siruseri IT park, OMR,  
Chennai - 603 103

### **SIGNATURE**

Dr.D.WESLIN, M.E., Ph.D.,

### **SUPERVISOR**

ASSOCIATE PROFESSOR

Department of Information  
Technology

Mohamed Sathak AJ College of  
Engineering, Siruseri IT park, OMR,  
Chennai - 603 103

Submitted for the university practical examination held at **MOHAMED SATHAK A.J COLLEGE OF ENGINEERING, CHENNAI** on .....

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

## ACKNOWLEDGEMENT

Firstly, we express our gratitude to the almighty for facilitating the successful completion of this project under all circumstances.

We extend our sincere and heartfelt appreciation to **ALHAJ JANAB YOUSUF S.M., Chairman, Mr. MOHAMED SATHAK, Director, JANABA S.H.SHARMILA, Secretary, and JANAB P.R.L HAMID IBRAHIM, Executive Director,** Mohamed Sathak AJ College of Engineering, Chennai.

We record our immense pleasure in expressing sincere gratitude to our Principal **Dr. K.S. SRINIVASAN, M.E., Ph.D., Mohamed Sathak A J College of Engineering,** for granting permission to undertake the project in our college.

We express our sincere thanks to **Dr. D. PRAKASH, M.E., Ph.D., Head and Associate Professor,** Mohamed Sathak A J College of Engineering, our Project and Supervisor **Dr. D. WESLIN, M.E., Ph.D., HEAD OF THE RESEARCH and IRP,** Mohamed Sathak A J College of Engineering for their constant encouragement and direction for this project.

We wish to express our thanks to all the **Faculty Members and Non-teaching staff** of the Department of Information Technology for their valuable support.

Finally, we are thankful to ChatGPT who encouraged us and helped us in doing this project

## **ABSTRACT**

The Virtual Mouse system revolutionizes Human-Computer Interaction (HCI) by leveraging real-time camera technology to offer an innovative alternative to traditional input methods. By capturing and processing real-time images through a camera, the system extracts precise coordinates for cursor movement, allowing users to control actions through hand gestures. This approach introduces a seamless and accessible interface that transcends the limitations of physical mouse or button presses.

Integrating hand movement further enhances the system's convenience and accessibility, particularly benefiting individuals facing challenges related to hand mobility. By mimicking all the functions of a physical mouse, the Virtual Mouse seamlessly integrates with existing technologies, thereby marking a significant advancement in HCI.

The system's ability to interpret hand gestures provides specially-abled individuals with intuitive control, fostering digital independence and enhancing productivity. This empowerment opens new avenues for inclusivity in the digital realm, ensuring that individuals of diverse abilities can engage with technology effectively.

Overall, the Virtual Mouse represents a pivotal innovation in advancing human-computer interactions across a wide spectrum of user demographics. Its ability to offer intuitive control, accessibility, and seamless integration with existing technologies makes it a game-changer in HCI, promising to redefine how users interact with digital interfaces in both personal and professional settings. This innovation not only enhances the user experience but also contributes to a more inclusive and equitable digital landscape, where technology serves as a tool for empowerment and connectivity for all.

## **TABLE OF CONTENTS**

<b>CHAPTER NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>
	<b>BONAFIDE CERTIFICATE</b>	<b>ii</b>
	<b>ACKNOWLEDGEMENT</b>	<b>iii</b>
	<b>ABSTRACT</b>	<b>iv</b>
	<b>LIST OF FIGURES</b>	<b>vii</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>01</b>
	<b>1.1 HUMAN COMPUTER INTERACTION</b>	<b>03</b>
	<b>1.2 VIRTUAL MOUSE SYSTEMS</b>	<b>03</b>
	<b>1.3 OVERVIEW</b>	<b>04</b>
	<b>1.4 INTEGRATION</b>	<b>05</b>
	<b>1.5 EXPERIMENTAL SETUP</b>	<b>05</b>
<b>2</b>	<b>SYSTEM STUDY</b>	<b>07</b>
	<b>2.1 EXISTING SYSTEM</b>	<b>07</b>
	<b>2.1.1 MECHANICAL MOUSE</b>	<b>08</b>
	<b>2.1.2 OPTICAL AND LASER MOUSE</b>	<b>09</b>
	<b>2.2 LITERATURE REVIEW</b>	<b>10</b>
	<b>2.3 PROPOSED SYSTEM</b>	<b>13</b>
	<b>2.3.1 USAGE OF CAMERA</b>	<b>14</b>
	<b>2.3.2 VIDEO PROCESSING</b>	<b>14</b>
<b>3</b>	<b>SYSTEM ANALYSIS AND DESIGN</b>	<b>16</b>
	<b>3.1 SPECIFICATIONS</b>	<b>16</b>

	<b>3.1.1 VISUSAL STUDIO</b>	<b>16</b>
	<b>3.1.2 PYTHON</b>	<b>17</b>
	<b>3.1.3 MODULES USED</b>	<b>19</b>
	<b>3.1.4 ANACONDA ENVIRONMENT</b>	<b>29</b>
	<b>3.2 ARCHITECTURE</b>	<b>31</b>
	<b>3.2.1 PROCESS OF KYAAT</b>	<b>32</b>
	<b>3.3 SEQUENCE DIAGRAM</b>	<b>35</b>
	<b>3.3.2 FLOW OF SEQUENCE DIAGRAM</b>	<b>36</b>
<b>4</b>	<b>SAMPLE OUTPUT</b>	<b>39</b>
<b>5</b>	<b>FUTURE IMPROVEMENTS NEEDED</b>	<b>51</b>
	<b>5.1 IMPLEMENTATIONS</b>	<b>51</b>
<b>6</b>	<b>CONCLUSION</b>	<b>53</b>
	<b>APPENDIX</b>	<b>54</b>
	<b>SAMPLE SOURCE CODE</b>	<b>54</b>
	<b>REFERENCES</b>	<b>58</b>

## **LIST OF FIGURES**

<b>FIGURE NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>
2.1	MECHANICAL MOUSE	08
2.2	OPTICAL MOUSE	10
3.1	HAND TRACKING	24
3.2	HAND TRACKING WITH MEDIAPIPE	25
3.3	ARCHITECTURE	31
3.4	SEQUENCE DIAGRAM	35
4.1	NEUTRAL GESTURE	39
4.2	MOVE CURSOR	40
4.3	LEFT CLICK	41
4.4	RIGHT CLICK	42
4.5	DOUBLE CLICK	43
4.6	DRAG AND DROP	44
4.7	MULTIPLE ITEM SELECTION	46
4.8	VOLUME CONTROL	49
4.9	BRIGHTNESS CONTROL	50

# CHAPTER 1

## 1. INTRODUCTION

Human-computer interaction (HCI) has evolved significantly over the years, becoming an integral aspect of our daily lives as we engage with various technological devices and interfaces. Central to this interaction is the input device, which enables users to control and navigate graphical user interfaces (GUIs) efficiently. While traditional physical mouse have long been the standard input method, advancements in technology have paved the way for innovative alternatives, such as virtual mouse systems, offering new possibilities and convenience in how we interact with technology.

A virtual mouse system represents a software-based rendition of a physical mouse, eliminating the need for a tangible pointing device and enabling users to manipulate on-screen elements through hand gestures and speech commands. This innovation holds particular relevance in contexts where traditional mouse are impractical or unavailable, such as touch-based devices like smartphones and tablets, virtual reality environments, and accessibility solutions for individuals with disabilities.

In this report, we delve into the concept of virtual mouse systems, exploring their functionalities, applications, advantages, and challenges. By understanding the capabilities and implications of virtual mouse technology, we aim to gain insights into the evolving landscape of human-computer interaction and the pivotal role of innovative input technologies in shaping our digital experiences.

The Virtual Mouse system proposed in this work, known as 'KYAAT' (Kinetic Yeomanry for Adaptive Accessibility in Technology), aims to revolutionize the way people interact with computers by leveraging advanced



technology to interpret hand movements accurately. Instead of relying on traditional input devices like mouse or touchpads, users can control the cursor on the screen simply by moving their hands. This approach not only enhances accessibility but also adds a layer of interactivity and convenience to computing experiences.

One of the key features of the Virtual Mouse system is its integration of hand gestures and speech recognition, two of the most efficient and expressive forms of human communication. These modes of communication are universally understood, making the system accessible to individuals with diverse abilities and preferences. The experimental setup of the Virtual Mouse system involves a cost-effective webcam positioned atop a computer monitor or a fixed laptop camera, which captures hand gestures for processing.

For gesture recognition, the system utilizes the Python programming language and the OpenCV library for computer vision tasks, enabling the accurate capture and interpretation of hand movements. Hand tracking is facilitated by the MediaPipe package within the Virtual Mouse system's model, ensuring precise cursor control based on the user's hand gestures.

By combining innovative technology with user-centric design principles, the Virtual Mouse system aims to democratize access to computing resources and empower individuals with disabilities to engage more effectively with digital interfaces. Through ongoing research and development efforts, we are committed to refining the system to ensure ease of use and accessibility for all users, regardless of their physical abilities or technological proficiency.

## **1.1 Introduction to Human-Computer Interaction (HCI):**

Human-computer interaction (HCI) has undergone significant evolution, becoming an indispensable aspect of modern daily life as individuals engage with various technological devices and interfaces. At the heart of this interaction lies the input device, crucial for controlling and navigating graphical user interfaces (GUIs) efficiently.

While the traditional physical mouse has long been the standard input method, technological advancements have paved the way for innovative alternatives, such as virtual mouse systems, offering novel possibilities and convenience in how we interact with technology.

These systems, represented as software-based renditions of physical mice, eliminate the need for tangible pointing devices, enabling users to manipulate on-screen elements through hand gestures and speech commands.

This evolution in HCI is particularly relevant in contexts where traditional mouse devices are impractical or unavailable, such as touch-based devices like smartphones and tablets, virtual reality environments, and accessibility solutions for individuals with disabilities.

## **1.2 Introduction to Virtual Mouse Systems:**

Virtual mouse systems represent a paradigm shift in human-computer interaction, offering users a new way to interact with digital interfaces without relying on conventional input devices. The concept of virtual mouse systems encompasses the idea of using software algorithms to interpret hand movements and gestures as inputs for controlling the cursor on a computer screen.

Unlike physical mice or touchpads, which require users to manipulate tangible objects, virtual mouse systems harness advanced technology to detect and analyze hand

gestures, providing a more intuitive and natural method of interaction. These systems leverage computer vision algorithms and machine learning techniques to interpret hand movements captured by cameras, enabling precise control of on-screen elements through gestures.

By eliminating the need for physical input devices, virtual mouse systems enhance accessibility and usability, opening up new possibilities for interaction in diverse computing environments.

### **1.3 Overview of the Proposed Virtual Mouse System - 'KYAAT':**

The proposed Virtual Mouse system, known as 'KYAAT' (Kinetic Yeomanry for Adaptive Accessibility in Technology), represents a groundbreaking approach to human-computer interaction. The primary objective of the 'KYAAT' system is to revolutionize how people interact with computers by leveraging advanced hand gesture recognition technology.

Unlike traditional input devices like mice or touchpads, which require physical manipulation, the 'KYAAT' system enables users to control the cursor on the screen simply by moving their hands. This approach not only enhances accessibility but also adds a layer of interactivity and convenience to computing experiences.

Central to the 'KYAAT' system is the integration of hand gestures and speech recognition, two of the most efficient and expressive forms of human communication. These modes of communication are universally understood, making the system accessible to individuals with diverse abilities and preferences.

The experimental setup of the 'KYAAT' system involves a cost-effective webcam positioned atop a computer monitor or a fixed laptop camera, which captures hand gestures for processing. For gesture recognition, the system utilizes the Python programming language and the OpenCV library for computer vision tasks, enabling the accurate capture and interpretation of hand movements.

Hand tracking is facilitated by the MediaPipe package within the 'KYAAT' system's model, ensuring precise cursor control based on the user's hand gestures.

#### **1.4 Integration of Hand Gestures and Speech Recognition:**

The integration of hand gestures and speech recognition in the 'KYAAT' system is a pivotal aspect of its design, enhancing user interaction and accessibility. Hand gestures and speech commands serve as intuitive and natural modes of communication, enabling users to interact with the system effortlessly.

By combining these modalities, the 'KYAAT' system caters to a wide range of users, including those with physical disabilities or limitations. Hand gestures allow for precise cursor control and manipulation of on-screen elements, while speech recognition enables users to execute commands and perform actions using voice commands.

This multimodal approach ensures flexibility and adaptability in user interaction, accommodating individual preferences and accessibility needs. Moreover, the integration of hand gestures and speech recognition enhances the overall user experience, making interaction with the system more intuitive and immersive.

#### **1.5 Experimental Setup and Technical Framework:**

The experimental setup and technical framework of the 'KYAAT' system lay the foundation for its functionality and performance. The system utilizes a cost-effective webcam positioned atop a computer monitor or a fixed laptop camera to capture hand gestures in real-time.

These gestures are then processed using the Python programming language and the OpenCV library for computer vision tasks, including gesture recognition and hand tracking. The MediaPipe package within the system's model facilitates precise cursor

control based on the user's hand movements, ensuring seamless interaction with on-screen elements.

The integration of advanced technologies and algorithms enables the 'KYAAT' system to interpret hand gestures accurately and respond in real-time, providing users with a fluid and intuitive interaction experience.

Through ongoing research and development efforts, the system aims to refine its functionality and usability, further enhancing accessibility and user engagement.

## **CHAPTER 2**

### **2. SYSTEM STUDY**

#### **2.1 EXISTING SYSTEM**

Nowadays computers mainly use things like mouse and touchpads for you to control them. But these can be hard to use, especially if you can't move around easily. We're trying to make a system where you can control the computer without needing to touch anything. This will help people who have trouble moving or using regular input devices. We want to make the system easier to use for everyone.

Overall, we hope this project will make using computers easier and more inclusive for everyone. The mouse is a key computer input device used to control the graphical user interface (GUI) through two-dimensional movements on a surface. Initially, mechanical mouse with rubber balls were used, later replaced by optical mouse using LED sensors, and then laser mouse for more accuracy.

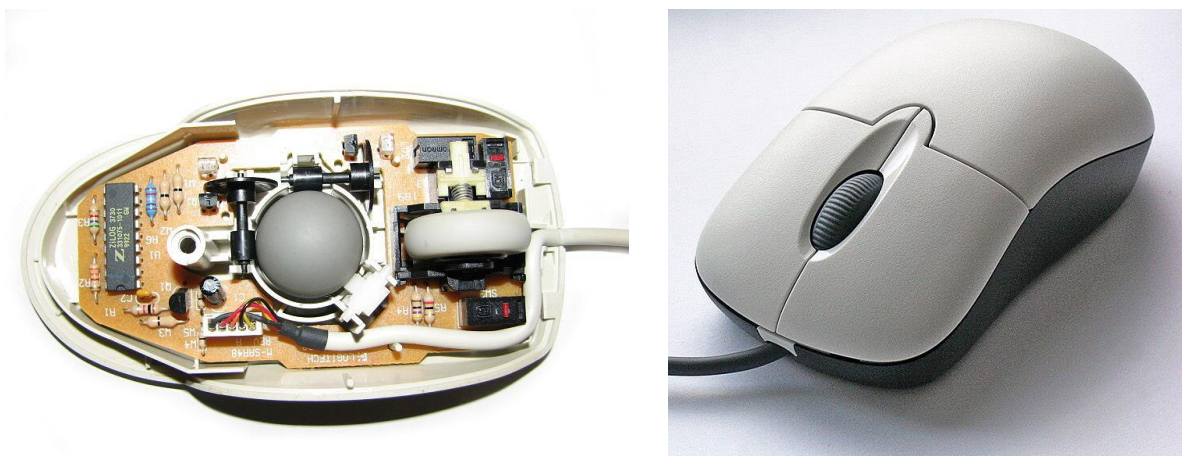
However, mouse have limitations like wear and tear leading to replacements. With the rise of touchscreens, there's a demand for similar technology in desktop systems at a high cost. A potential alternative is virtual human-computer interaction using webcams or image-capturing devices, where software translates user gestures into cursor movements, mimicking a physical mouse.

### 2.1.1 Mechanical Mouse:

Known as the trackball mouse that is commonly used in the 1990s, the ball within the mouse are supported by two rotating rollers in order to detect the movement made by the ball itself. One roller detects the forward/backward motion while the other detects the left/right motion.

The ball within the mouse are steel made that was covered with a layer of hard rubber, so that the detection are more precise. The common functions included are the left/right buttons and a scroll-wheel. However, due to the constant friction made between the mouse ball and the rollers itself, the mouse are prone to degradation, as overtime usage may cause the rollers to degrade, thus causing it to unable to detect the motion properly, rendering it useless.

Furthermore, the switches in the mouse buttons are no different as well, as long term usage may cause the mechanics within to be loosed and will no longer detect any mouse clicks till it was disassembled and repaired.



**Figure 2.1-** Mechanical mouse, with top cover removed.

Advantage	Disadvantage
Allows the users to control the computer system by moving the mouse. Provides precise mouse tracking Movements.	Prone to degradation of the mouse rollers and button switches, causing to be faulty. Requires a flat surface to operate

### **2.1.2 Optical And Laser Mouse:**

A mouse that commonly used in these days, the motions of optical mouse rely on the Light Emitting Diodes (LEDs) to detect movements relative to the underlying surface, while the laser mouse is an optical mouse that uses coherent laser lights.

Comparing to its predecessor, which is the mechanical mouse, the optical mouse no longer rely on the rollers to determine its movement, instead it uses an imaging array of photodiodes. The purpose of implementing this is to eliminate the limitations of degradation that plagues the current predecessor, giving it more durability while offers better resolution and precision.

Furthermore, long term usage without a proper cleaning or maintenance may leads to dust particles trap between the LEDs, which will cause both optical and laser mouse having surface detection difficulties. Other than that, it's still prone to degradation of the button switches, which again will cause the mouse to function improperly unless it was disassembled and repaired.





**Figure 2.2** -Optical mouse, with top cover removed

Advantages	Disadvantages
<p>Allows better precision with lesser hand movements.</p> <p>Longer life-span.</p>	<p>Prone to button switches degradation.</p> <p>Does not function properly while on a polished surface.</p>

## 2.2 LITERATURE REVIEW

A diverse range of research has explored innovative approaches to virtual mouse technology:

Chen-Chiung Hsieh et al. introduced a real-time hand gesture recognition system utilizing motion history images and adaptive skin color detection, implemented in C++ with OpenCV.

Kamran Niyazi et al. developed a mouse simulation system using two-colored tapes and methods like background subtraction and skin detection, employing Java for implementation.

Abhik Banerjee et al. proposed mouse control through a web camera based on color detection, with different colors representing cursor control and clicks, implemented in MATLAB.

Sandeep Thakur et al. focused on vision-based mouse control using hand gestures and color caps, implemented in MATLAB. Horatiu-stefan Grif et al. presented mouse cursor control based on hand gestures with an external camera and color strips, implemented in C with OpenCV.

Pooja Kumari et al. introduced cursor control using camera-captured hand gestures and color tips, implemented in MATLAB and OpenCV.

Danling Lu et al. utilized a data glove for gesture recognition and mouse control, implementing an Extreme Learning Machine method.

Alisha Pradhana et al. designed an intangible interface for mouseless handling with hand gestures, using Microsoft Visual Studio.

Aashni Hariiaa et al. focused on hand gesture recognition with background extraction and contours detection.

This review examines various approaches to hand gesture recognition systems, highlighting their strengths and limitations. One hardware-based system discussed in achieves high accuracy but is hindered by the restrictive nature of wearing gloves, which limits the user's hand movements and can lead to discomfort and skin issues, particularly for those with sensitive skin.

In contrast, presents a machine-user interface that relies on computer vision and multimedia techniques for hand gesture detection. However, a drawback is the need for skin pixel identification and hand segmentation before gesture comparison, adding complexity to the process.

Another innovative approach described in utilizes a mobile phone camera and projector for visual feedback, enabling easy integration with other mobile applications for gesture recognition. This architecture emphasizes user engagement with content rather than device interaction.

Introduces a method for mouse functions using only a webcam, bypassing the

need for additional sensors or electrical equipment. While simple, this approach lacks accuracy and comprehensive mouse functionality.

In the focus shifts to gesture-controlled robots, employing optical flow estimation and face detection for user-centric representation and classification.

Utilizes the convex hull technique to detect fingertip points for mouse control, but the requirement to save frames before processing hampers real-time performance.

Explores a vision-based technique using a webcam for gesture recognition, leveraging the YOLOv5 algorithm and artificial intelligence to enhance human-computer interaction (HCI).

Lastly, implements colored masks for gesture recognition, followed by mouse functions performed through hand gestures, although the approach's complexity presents implementation challenges.

These studies underscore the diverse methodologies and technologies employed to enhance human-computer interactions through virtual mouse control systems.

## **2.3 PROPOSED SYSTEM**

The 'KYAAT' system is a game-changer in how we interact with computers. It uses special technology to understand intricate hand movements, eliminating the need for traditional input devices like mice or touchpads. With 'KYAAT', users can effortlessly control the cursor on-screen by simply gesturing with their hands. This breakthrough is especially significant for people with mobility challenges or difficulty using standard input methods.

By breaking down these traditional barriers, 'KYAAT' not only makes computer usage easier but also adds an element of enjoyment for users of all backgrounds and abilities. We're committed to ensuring that 'KYAAT' is accessible to everyone, reflecting our dedication to inclusivity in technological innovation.

The Virtual Mouse system, a core component of 'KYAAT', harnesses the intuitive and expressive nature of hand gestures for computer interaction. Hand gestures are a universally understood form of communication, transcending language and cultural differences, making them ideal for widespread use.

The system's setup involves positioning a cost-effective webcam on a computer monitor or fixed laptop camera, which captures and interprets hand gestures to perform various mouse functions.

To recognize gestures, the Virtual Mouse system utilizes the Python programming language and the powerful capabilities of the OpenCV library for complex computer vision tasks such as gesture capture and interpretation. Hand tracking, an essential feature of the system, is seamlessly enabled by the integration of the MediaPipe package within the Virtual Mouse system's architecture.

This comprehensive approach ensures precise and responsive cursor control, enhancing the overall user experience.

In summary, the 'KYAAT' system and its Virtual Mouse component mark a significant advancement in human-computer interaction, driven by a commitment to accessibility, innovation, and user-centric design. By leveraging the natural capabilities of hand gestures and cutting-edge technology, we aim to empower users of all abilities to interact seamlessly with digital interfaces.

Our ongoing efforts to refine and optimize the Virtual Mouse system are guided by our mission to democratize access to technology and create a more inclusive digital landscape.

### **2.3.1 The Camera Used in the KYAAT:**

The proposed system uses web camera for capturing images or video based on the frames. For capturing we are using CV library Opencv which is belongs to python web camera will start capturing the video and Opencv will create a object of video capture. To virtual system the frames are passed from the captured web camera.

### **2.3.2 Capturing the Video and Processing:**

The capturing of the frame was done with the AI virtual mouse system until the program termination. Then the video captured has to be processed to find the hands in the frame in each set.

The processing takes places is it converts the BRG images into RGB images, which can be performed with the below code,

```
image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)  
image.flags.writeable = False  
results = hands.process(image)
```

This code is used to flip the image in the horizontal direction then the resultant image is converted from the BRG scale to RGB scaled image.

### **2.3.3 Rectangular Region for Moving through the Window:**

The windows display is marked with the rectangular region for capturing the hand gesture to perform mouse action based on the gesture. when the hands are find under those rectangular area the detection begins to detect the action based on that the mouse cursor functions will be performed.

The rectangular region is drawn for the purpose of capturing the hand gestures through the web camera which are used for mouse cursor operations.

## CHAPTER 3

### 3. SYSTEM ANALYSIS AND DESIGN

#### SPECIFICATIONS

##### HARDWARE SPECIFICATIONS:

Laptop with webcam

CPU: intel i3 or high


RAM: 4GB or Above


Memory: 8GB or Above

##### SOFTWARE SPECIFICATIONS:

Operating System : Windows 7 or above 

Code Editor : Visual studio 

Language : Python 

Environment : Anaconda 

#### 3.1.1 VISUAL STUDIO:

Visual Studio is an integrated development environment (IDE) created by Microsoft for developing a wide range of software applications.

**Code Editing:** Visual Studio provides robust code editors with features such as syntax highlighting, IntelliSense (code completion and suggestion), code navigation, and automatic formatting. It supports various programming languages including C#, Visual Basic, C++, F#, Python, JavaScript, TypeScript, and more.

**Debugging Tools:** Debugging is made easier with Visual Studio's powerful debugging tools. Developers can set breakpoints, inspect variables, step through

code execution, and analyze application behavior to identify and fix issues efficiently.

**Testing Tools:** Visual Studio includes testing tools for unit testing, performance testing, and automated testing. Developers can create and run tests, analyze test results, and ensure the quality and reliability of their code.

**Cross-Platform Development:** Visual Studio facilitates cross-platform development for desktop, web, mobile, and cloud applications. Developers can create applications targeting Windows, macOS, Linux, Android, iOS, web browsers, Azure cloud services, and more using a single IDE.

**Extensions and Customization:** Visual Studio's extensibility model allows developers to enhance the IDE's functionality with extensions available in the Visual Studio Marketplace. These extensions provide additional tools, templates, code snippets, and integrations with third-party services.

**Performance Profiling:** Developers can use Visual Studio's performance profiling tools to analyze application performance, identify bottlenecks, optimize code, and improve overall application speed and efficiency.

### 3.1.2 PYTHON

Python plays a crucial role in your project, especially in the implementation of the Virtual Mouse system. Here's how Python is utilized:

1. **Gesture Recognition:** Python is used to develop algorithms for gesture recognition. Libraries such as OpenCV are commonly employed for computer vision tasks, allowing Python scripts to analyze real-time images captured by the camera and detect hand gestures accurately.
2. **Integration with MediaPipe:** Within the Virtual Mouse system, Python scripts interface with the MediaPipe package, which provides tools and



pipelines for hand tracking and pose estimation. Python serves as the primary programming language for utilizing MediaPipe functionalities and integrating them into the system.

3. **Hand Tracking:** Python scripts facilitate hand tracking by processing images captured by the camera to identify and track the movement of the hand region across consecutive frames. This involves utilizing MediaPipe's hand tracking capabilities within the Python environment.
4. **Finger Tip Detection:** Python is used to detect key points, such as fingertip positions, within the hand region. Algorithms implemented in Python analyze hand pose data to identify and extract relevant features, aiding in the accurate detection of finger tips and hand gestures.
5. **Gesture Analysis and Recognition:** Python scripts analyze finger positions and overall hand pose to recognize meaningful hand gestures based on a predefined gesture library. Using Python, the system compares detected hand poses with known gesture patterns to identify the user's intended actions effectively.
6. **Action Triggering:** Based on the recognized hand gestures, Python scripts trigger corresponding actions or commands within the system. This involves executing predefined functions or operations based on the recognized gestures, enabling seamless interaction between the user and the Virtual Mouse system.

Overall, Python serves as the backbone of your project, providing a versatile and powerful programming environment for implementing gesture recognition algorithms, integrating with external libraries and packages, processing image data, and executing system actions based on user input. Its simplicity, readability, and extensive ecosystem of libraries make Python an ideal choice for developing innovative human-computer interaction solutions like the Virtual Mouse system.

### 3.1.3 MODULES USED:

#### **cv2 (OpenCV):**

OpenCV, a computer vision and machine learning software library, is freely available for download, serving as a valuable resource for programmers developing computer vision applications. It offers a wide range of functionalities, including filtering, feature identification, object recognition, tracking, and other image and video processing operations.

With bindings for programming languages like Python, Java, and MATLAB, OpenCV provides versatility in application development. Written in C++, it finds utility across diverse fields such as robotics, self-driving cars, augmented reality, and medical image analysis.

The operation of OpenCV can be broadly categorized into several steps:

**1. Loading and Preprocessing the Image/Video:** OpenCV facilitates the loading of images or videos from various sources such as files, cameras, or network streams. Following loading, preprocessing tasks like applying filters or transforming color spaces (e.g., converting color images to grayscale) can be performed.

**2. Feature Detection and Description:** OpenCV enables the detection and extraction of features from images or videos, including edges, corners, and blobs. These features play a crucial role in object identification and motion tracking. Additionally, OpenCV provides algorithms for describing these features, facilitating their matching across multiple frames or images.

**3. Object Detection and Recognition:** OpenCV supports object detection and recognition in images or videos through techniques like template matching, Haar cascades, or deep learning-based methods. This capability allows for the identification of specific objects within a scene.

**4. Tracking:** OpenCV enables object tracking in video streams by estimating their position and motion over time. This can be achieved using various algorithms such as optical flow, mean-shift, or Kalman filtering, providing valuable insights into object behavior and movement patterns.

**5. Image and Video Output:** Finally, OpenCV allows for the display or saving of processed images or videos. This can involve showing images in a window, writing video frames to a file, or streaming video over a network, facilitating visualization and further analysis of processed data.

Overall, OpenCV offers a comprehensive toolkit comprising a variety of tools and techniques for working with image and video data, making it an indispensable resource for developers in the field of computer vision.

### **mediapipe:**

Google introduced the open-source MediaPipe framework to facilitate the development of real-time computer vision applications across various platforms. Offering a range of pre-built tools and components for processing and analyzing video and audio streams, MediaPipe includes features like object detection, pose estimation, hand tracking, facial recognition, and more.

Developers can leverage these tools to swiftly construct complex pipelines, combining multiple algorithms and processes, and execute them in real-time on diverse hardware platforms such as CPUs, GPUs, and specialized accelerators like Google's Edge TPU.

Moreover, MediaPipe seamlessly integrates with popular machine learning libraries like TensorFlow and PyTorch, supporting multiple programming languages including C++, Python, and Java.

MediaPipe encompasses a comprehensive set of features tailored for computer vision and machine learning tasks. Here are some key attributes and functionalities of the framework:

**1. Video and Audio Processing:** MediaPipe provides robust tools for processing and analyzing video and audio streams in real-time. This includes functionalities such as video decoding, filtering, segmentation, and synchronization.

**2. Facial Recognition:** MediaPipe offers facial recognition capabilities by detecting and tracking facial landmarks such as eyes, nose, mouth, and eyebrows in real-time. This functionality is essential for applications like facial recognition, emotion detection, and augmented reality.

**3. Hand Tracking:** With MediaPipe, developers can track hand movements in real-time, enabling hand gesture recognition and interaction with virtual objects. This feature enhances applications involving human-computer interaction and virtual reality experiences.

**4. Object Detection:** MediaPipe facilitates real-time object detection and tracking using machine learning models. This functionality is beneficial for applications such as augmented reality, robotics, and surveillance, where the accurate identification and tracking of objects are critical.

**5. Pose Estimation:** MediaPipe enables the estimation of human body poses in real-time, allowing for applications like fitness tracking, sports analysis, and augmented reality experiences. This feature provides valuable insights into human movement and posture, enhancing various interactive applications and experiences.

### **pyautogui:**

PyAutoGUI is a Python library that enables GUI automation by simulating user interactions with the mouse and keyboard. It's platform-independent and can be used

for automating tasks across operating systems. PyAutoGUI is useful for tasks such as automated testing, data entry automation, repetitive task automation, and creating bots for games or applications. It offers functions for mouse movement, clicks, keyboard input, screen capturing, and window management.

### **math:**

The `math` module is part of Python's standard library and provides mathematical functions and constants for numerical computations. It includes functions for basic arithmetic operations (like `add()`, `subtract()`, `multiply()`, `divide()`), trigonometry (like `sin()`, `cos()`, `tan()`), logarithms, exponentiation, rounding, and more.

The module also provides constants such as `pi` and `e`. It's a fundamental tool for mathematical calculations in scientific computing, engineering, data analysis, and various other domains.

### **enum.IntEnum:**

Python's `Enum` module allows developers to create enumerations, which are sets of symbolic names bound to unique values. The `IntEnum` class specifically creates enumerations with integer values.

Enumerations improve code readability by providing meaningful names to constants, making the code more understandable and maintainable. They are commonly used to define states, options, error codes, and configuration settings in a program.

### **ctypes and comtypes:**

Both `ctypes` and `comtypes` are Python libraries used for interfacing with external components. `ctypes` provides a way to call functions from shared libraries

(DLLs/SOs) written in C/C++, allowing Python programs to access C data types, structures, and functions. On the other hand, ``comtypes`` is used for interacting with Component Object Model (COM) objects on Windows systems.

COM is a binary-interface standard for software components, and ``comtypes`` facilitates communication with COM objects, enabling Python to work with Windows-specific technologies like DirectX, Office Automation, and more.

### **pycaw:**

PyCaw is a Python library that provides an interface to the Windows Core Audio API. It allows developers to control audio settings on Windows systems programmatically. With PyCaw, you can manage audio devices, adjust volume levels, mute/unmute audio, set default playback/recording devices, and retrieve information about audio sessions.

It's useful for applications that require audio control, such as media players, voice recognition systems, teleconferencing tools, and automation scripts involving audio settings.

### **google.protobuf.json\_format:**

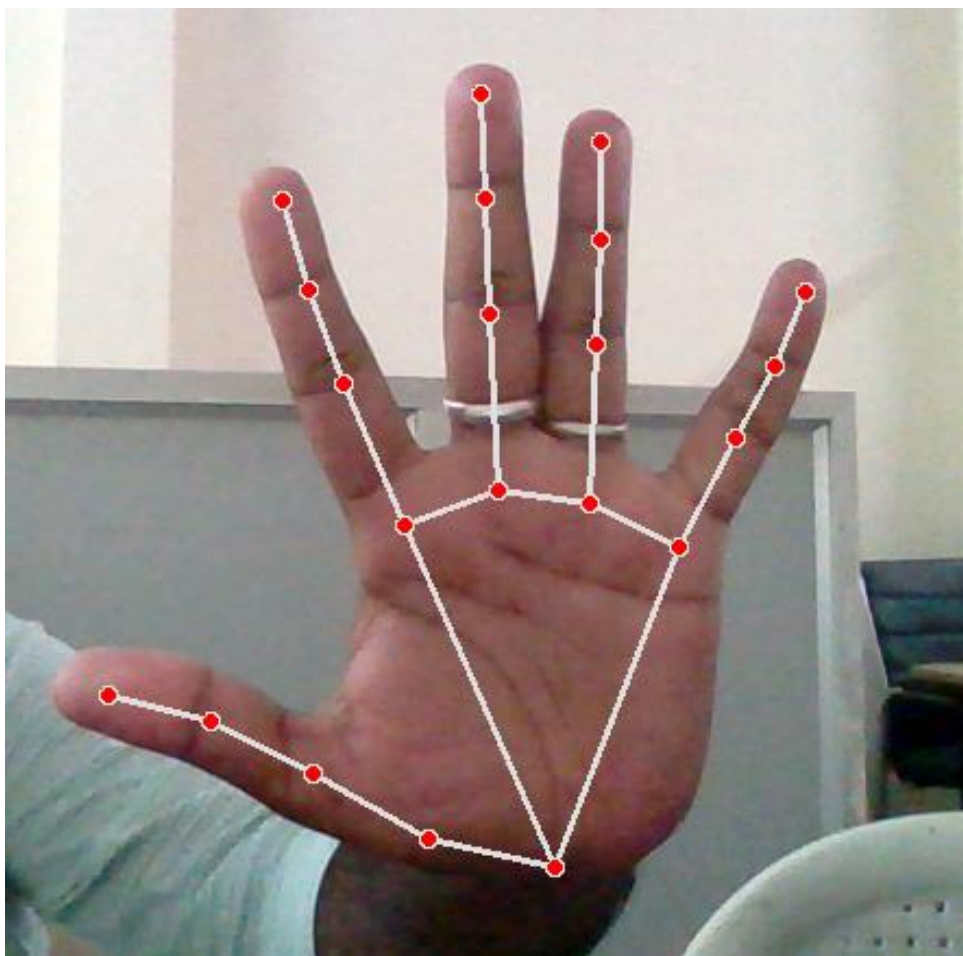
This module is part of Google's Protocol Buffers library (``protobuf``) and provides functions for converting Protocol Buffer messages to and from JSON format. Protocol Buffers (``protobuf``) is a language-agnostic data serialization format developed by Google, and it's commonly used for efficient and structured data exchange between different systems.

The ``json_format`` module simplifies interoperability between systems using Protocol Buffers and those using JSON, allowing seamless communication and data exchange.

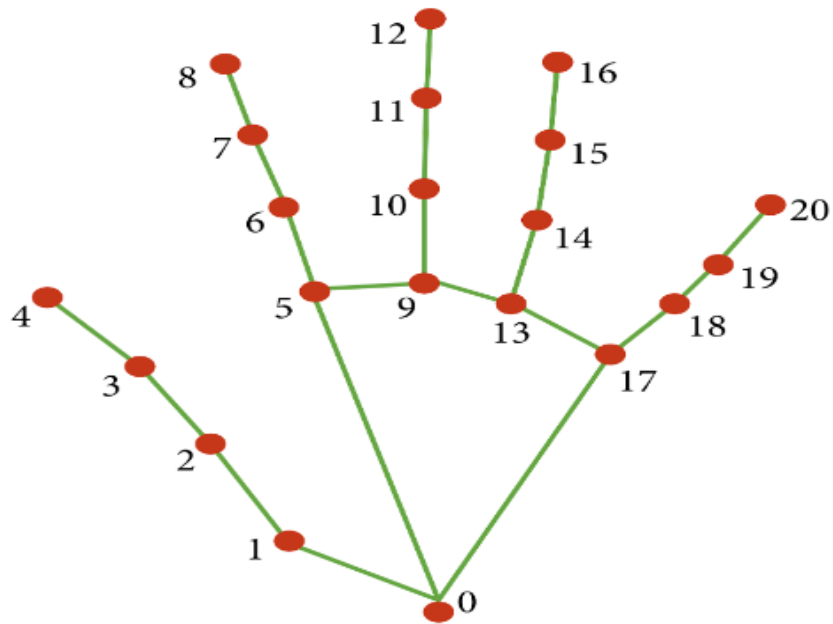
### **Screen\_brightness\_control:**

This Python library offers functionality to adjust screen brightness programmatically on supported platforms such as Windows, macOS, and Linux. It provides an abstraction layer to interact with system-level APIs or commands related to screen brightness control.

Developers can use it to dynamically adjust screen brightness based on ambient light conditions, user preferences, or application-specific requirements. It's beneficial for applications like power-saving utilities, ambient light sensors integration, accessibility tools for users with visual impairments, and display calibration tools.



**Figure 3.1 - Hand Tracking**



**Figure 3.2 - Hand Tracking with MediaPipe**

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



These labels serve as identifiers for specific parts of the hand and fingers within hand tracking systems, particularly those employing computer vision technology to detect and analyze hand gestures. By associating each label with a distinct anatomical feature or joint, these systems can precisely track and recognize hand movements, enabling a range of applications such as virtual reality, augmented reality, and gesture-based user interfaces.

**1. Wrist (WRIST):** This label corresponds to the wrist area of the hand, serving as a foundational reference point for hand tracking and gesture recognition. The wrist provides a stable starting point from which the system can track the movement and orientation of the entire hand.

**2. Thumb Carpometacarpal Joint (THUMB\_CMC):** This label refers to the base joint connecting the thumb to the hand. By tracking the position and movement of the thumb at this joint, the system can accurately detect gestures involving the thumb's base, such as grasping or pinching motions.

**3. Thumb Metacarpophalangeal Joint (THUMB\_MCP):** This label represents the joint between the thumb and the palm. Tracking the thumb's movement at this joint allows the system to capture gestures involving the thumb's alignment with the palm, such as thumb extension or flexion.

**4. Thumb Interphalangeal Joint (THUMB\_IP):** Denoting the joint in the middle of the thumb, this label enables the system to track gestures involving the bending or flexing of the thumb's middle segment.

**5. Thumb Tip (THUMB\_TIP):** This label refers to the tip or end of the thumb. Tracking the thumb's position at this point allows the system to detect precise movements and gestures involving the thumb's tip, such as tapping or pointing.

**6. Index Finger Metacarpophalangeal Joint (INDEX\_FINGER\_MCP):**

Corresponding to the joint between the index finger and the palm, this label enables the system to track movements involving the index finger's alignment with the palm.

**7. Index Finger Proximal Interphalangeal Joint (INDEX\_FINGER\_PIP):**

This label represents the joint closest to the hand on the index finger. Tracking movements at this joint allows the system to capture gestures involving the bending or flexing of the index finger's proximal segment.

**8. Index Finger Distal Interphalangeal Joint (INDEX\_FINGER\_DIP):**

Denoting the joint in the middle of the index finger, this label enables the system to track gestures involving the bending or flexing of the index finger's middle segment.

**9. Index Finger Tip (INDEX\_FINGER\_TIP):**

This label refers to the tip or end of the index finger. Tracking movements at the index finger's tip allows the system to detect precise gestures involving the index finger's pointing or touching actions.

**10. Middle Finger Metacarpophalangeal Joint (MIDDLE\_FINGER\_MCP):**

Corresponding to the joint between the middle finger and the palm, this label enables the system to track movements involving the middle finger's alignment with the palm.

**11. Middle Finger Proximal Interphalangeal Joint (MIDDLE\_FINGER\_PIP):**

This label represents the joint closest to the hand on the middle finger, allowing the system to capture gestures involving the bending or flexing of the middle finger's proximal segment.

**12. Middle Finger Distal Interphalangeal Joint (MIDDLE\_FINGER\_DIP):**

Denoting the joint in the middle of the middle finger, this label enables the system to track gestures involving the bending or flexing of the middle finger's middle segment.

**13. Middle Finger Tip (MIDDLE\_FINGER\_TIP):** Referring to the tip or end of the middle finger, this label allows the system to detect precise movements and gestures involving the middle finger's pointing or touching actions.

**14. Ring Finger Metacarpophalangeal Joint (RING\_FINGER\_MCP):** Corresponding to the joint between the ring finger and the palm, this label enables the system to track movements involving the ring finger's alignment with the palm.

**15. Ring Finger Proximal Interphalangeal Joint (RING\_FINGER\_PIP):** This label represents the joint closest to the hand on the ring finger, allowing the system to capture gestures involving the bending or flexing of the ring finger's proximal segment.

**16. Ring Finger Distal Interphalangeal Joint (RING\_FINGER\_DIP):** Denoting the joint in the middle of the ring finger, this label enables the system to track gestures involving the bending or flexing of the ring finger's middle segment.

**17. Ring Finger Tip (RING\_FINGER\_TIP):** Referring to the tip or end of the ring finger, this label allows the system to detect precise movements and gestures involving the ring finger's pointing or touching actions.

**18. Pinky Finger Metacarpophalangeal Joint (PINKY\_MCP):** Corresponding to the joint between the pinky finger and the palm, this label enables the system to track movements involving the pinky finger's alignment with the palm.

**19. Pinky Finger Proximal Interphalangeal Joint (PINKY\_PIP):** This label represents the joint closest to the hand on the pinky finger, allowing the system to capture gestures involving the bending or flexing of the pinky finger's proximal segment.

**20. Pinky Finger Distal Interphalangeal Joint (PINKY\_DIP):** Denoting the joint in the middle of the pinky finger, this label enables the system to track gestures involving the bending or flexing of the pinky finger's middle segment.

**21. Pinky Finger Tip (PINKY\_TIP):** Referring to the tip or end of the pinky finger, this label allows the system to detect precise movements and gestures involving the pinky finger's pointing or touching actions.

By utilizing these labels in hand tracking systems, developers can achieve accurate and detailed tracking of hand movements and gestures, enabling a wide range of interactive applications and interfaces.

These labels play a crucial role in facilitating natural and intuitive interaction between users and digital environments, ultimately enhancing user experiences across various domains, from virtual reality gaming to assistive technology for individuals with disabilities.

### **3.1.4 ANACONDA ENVIRONMENT:**

Anaconda is a popular open-source distribution of the Python and R programming languages, widely used for data science, machine learning, and scientific computing tasks. It provides a comprehensive suite of tools and packages, making it easy for users to manage their development environments and dependencies.

One of the key advantages of Anaconda is its package management system, which allows users to easily install, update, and uninstall packages with a simple command. This simplifies the process of setting up and maintaining development environments, ensuring consistency across different systems and projects.

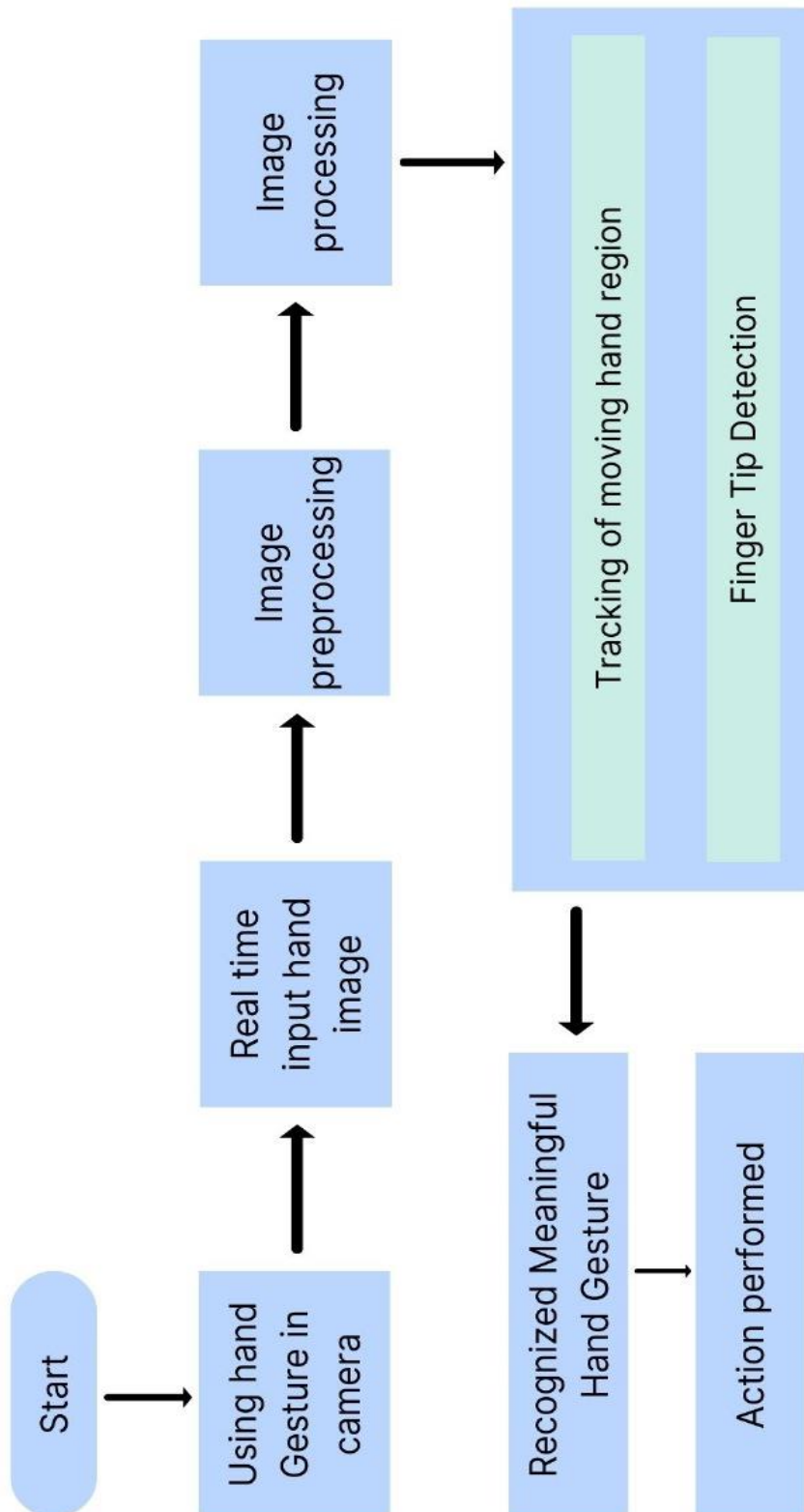
Additionally, Anaconda comes with its own package manager called conda, which enables users to create isolated environments with specific sets of

packages and dependencies. These environments can be easily replicated and shared, making it straightforward to collaborate on projects or deploy applications across different platforms.

Anaconda also includes a range of pre-installed libraries and tools commonly used in data science and scientific computing, such as NumPy, pandas, SciPy, Matplotlib, and Jupyter Notebook.

This out-of-the-box functionality accelerates the development process and eliminates the need for users to manually install and configure these libraries.

### 3.2 ARCHITECTURE:



**Figure 3.3** – Architecture of KYAAT

### **3.2.1 PROCESS OF KYAAT:**

#### **1. Start:**

- The interaction process begins with the user's intention to engage with a system or device using hand gestures.
- This intuitive approach to interaction offers a hands-free and natural way to convey commands or instructions to the system.

#### **2. Using Hand Gestures in front of Camera:**

- Within the camera's field of view, the user performs hand gestures to communicate with the system.
- These gestures serve as the primary mode of input, allowing users to convey their intentions without physical contact with the device.

#### **3. Real-Time Input Hand Image:**

- The camera captures real-time images of the user's hand gestures, providing a continuous stream of input data for gesture recognition.
- This real-time feedback enables the system to respond dynamically to the user's gestures as they occur.

#### **4. Image Preprocessing:**

- Captured images undergo preprocessing techniques to enhance image quality, remove noise, and optimize data for subsequent analysis.
- Preprocessing ensures that the input data is clean and suitable for accurate gesture recognition, improving the overall reliability of the system.

## **5. Image Processing:**

- Preprocessed images are analyzed to identify the hand region and extract relevant features for gesture recognition.
- This step involves detecting hand contours, key points, and other distinctive characteristics to facilitate accurate gesture interpretation.

## **6. Tracking of Moving Hand Region:**

- The system tracks the movement of the hand region across consecutive frames, allowing for the analysis of dynamic gestures performed over time.
- This tracking capability enables the recognition of gestures that involve motion or changes in hand position.

## **7. Finger Tip Detection:**

- Key points, such as fingertip positions, are detected within the hand region to accurately determine the hand's spatial configuration.
- Finger tip detection enhances the system's ability to interpret complex hand gestures and distinguish between different gestures based on subtle variations in hand posture.

## **8. Recognized Meaningful Hand Gesture:**

- The positions of fingers and the overall hand pose are analyzed to recognize meaningful hand gestures from a predefined gesture library.
- By comparing the detected hand pose with known gesture patterns, the system identifies the user's intended actions with a high degree of accuracy.



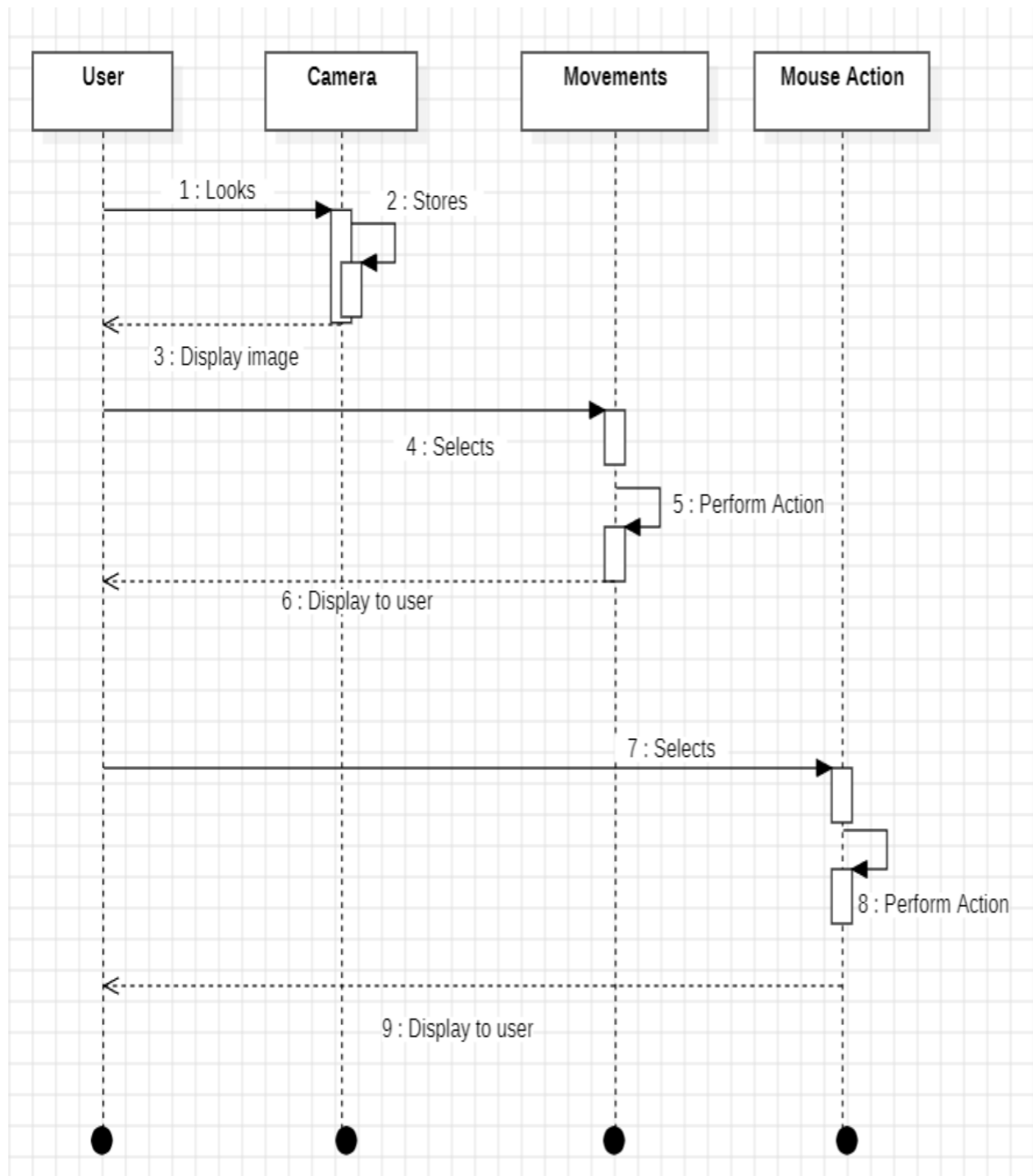
## **9. Action Performed:**

- Leveraging hand gestures for interaction offers numerous benefits, including Based on the recognized hand gesture, the system performs the corresponding action or triggers a specific operation.
- This could involve controlling a device, navigating a user interface, or executing a command, enabling seamless interaction between the user and the system.

Natural and intuitive control, hands-free operation, and accessibility for users with mobility impairments.

By understanding the process of hand gesture recognition and its integration into human-system interaction, developers can design systems that enhance user experience and facilitate more efficient and enjoyable interactions.

### 3.3 SEQUENCE DIAGRAM:



**Figure 3.4 - Sequence diagram**

### **3.3.2 FLOW OF SEQUENCE DIAGRAM:**

#### **1. Introduction:**

- User engagement in camera-equipped interfaces marks the initiation of interaction with the system.
- This process involves a series of steps, from image capture to action viewing, culminating in a seamless user experience.

#### **2. User Engagement:**

- The interaction begins when the user directs their attention towards the camera-equipped interface.
- This signifies the user's intent to engage with the system and initiate an action.

#### **3. Image Capture:**

- Once the user is engaged, the camera captures an image of the user's surroundings.
- This includes the user's gestures or actions within the camera's field of view, enabling the system to interpret user input.

#### **4. Image Storage:**

- The captured image is stored within the system's memory, awaiting processing and display.
- This step ensures that the system has access to the necessary visual data to respond to user interactions effectively.

#### **5. Image Display:**

- The stored image is presented to the user on the interface's display screen.

- This provides visual feedback of the captured scene, allowing the user to verify the accuracy of the system's perception.

## **6. Option Selection:**

- Using input devices like a mouse, the user selects an option displayed on the screen.
- This action indicates the user's desired course of action or choice within the system.

## **7. Action Triggering:**

- The selected option triggers a corresponding action or command within the system.
- This initiates the requested operation based on the user's input, driving the interaction forward.

## **8. Action Viewing (First Action):**

- The user observes the result of the triggered action displayed on the screen.
- This step provides feedback to the user, confirming the successful execution of the selected option and guiding further interactions.

## **9. Option Re-selection:**

- Subsequently, the user may choose to select another option presented on the screen.
- This indicates the user's intention to perform a subsequent action or express a new preference within the system.

## **10. Second Action Triggering:**

- The second selected option prompts another action or command within the system.

- This initiates a corresponding operation based on the user's input, continuing the interaction cycle.

### **11. Action Viewing (Second Action):**

- Finally, the user views the outcome of the second action displayed on the screen.

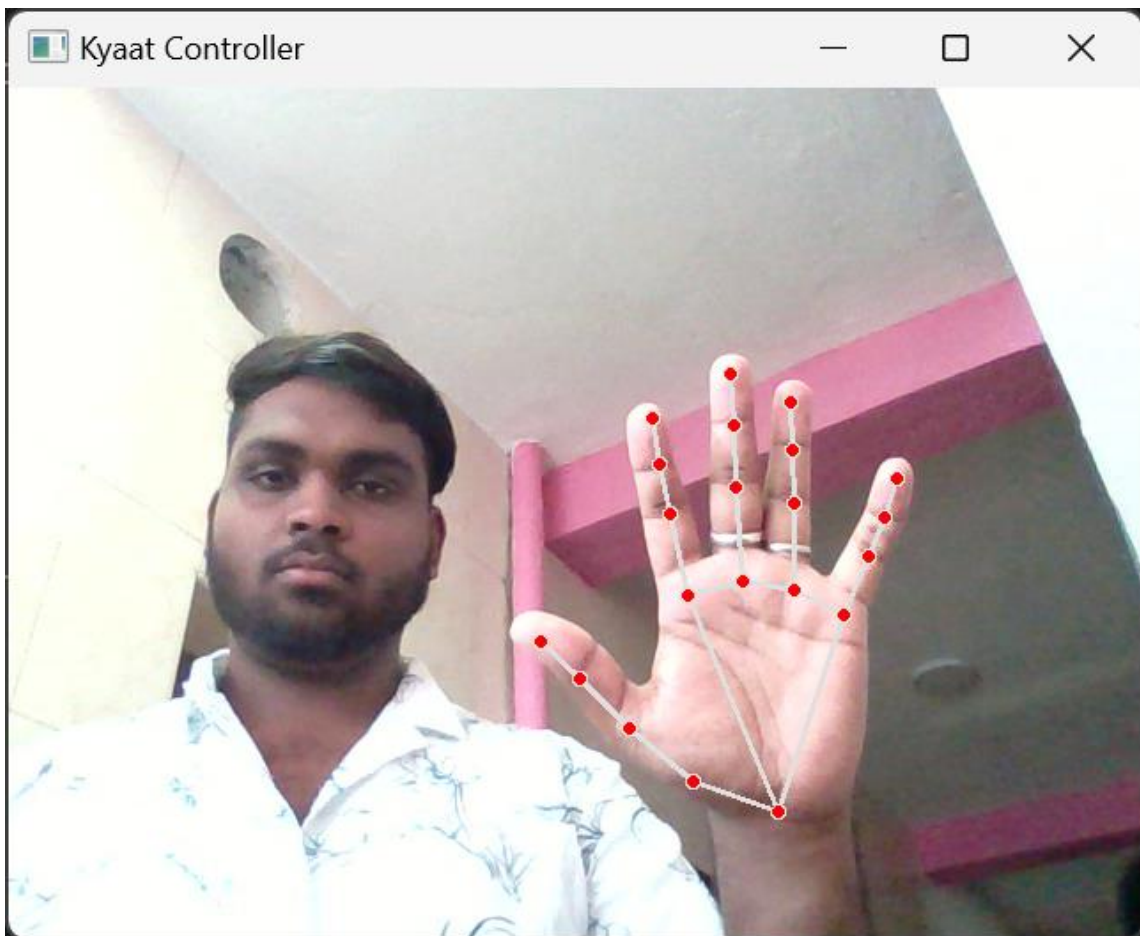
- This completes the interaction cycle, allowing the user to provide feedback on the system's response to their selections and ensuring a satisfying user experience.

Understanding the intricacies of user interaction in camera-equipped interfaces is essential for designing intuitive and efficient systems. By following the outlined steps, developers can create interfaces that facilitate seamless communication between users and technology, enhancing overall user satisfaction and usability.

## CHAPTER 4

### 4.SAMPLE OUTPUT

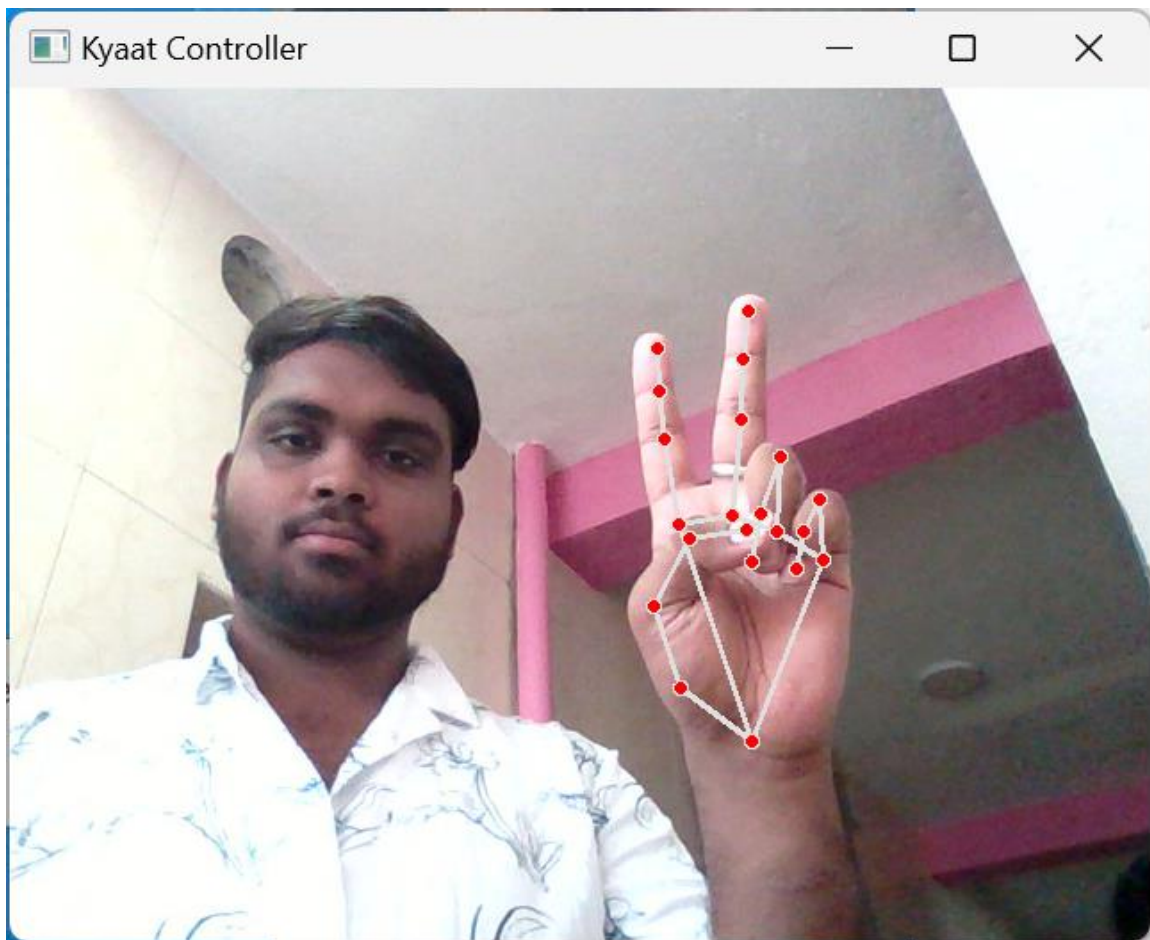
#### 4.1 Neutral Gesture:



#### Neutral Gesture:

- The neutral gesture serves as the default state of the system, requiring no explicit activation from the user.
- This gesture ensures that users can interact with the system effortlessly, without the need for continuous engagement or input.

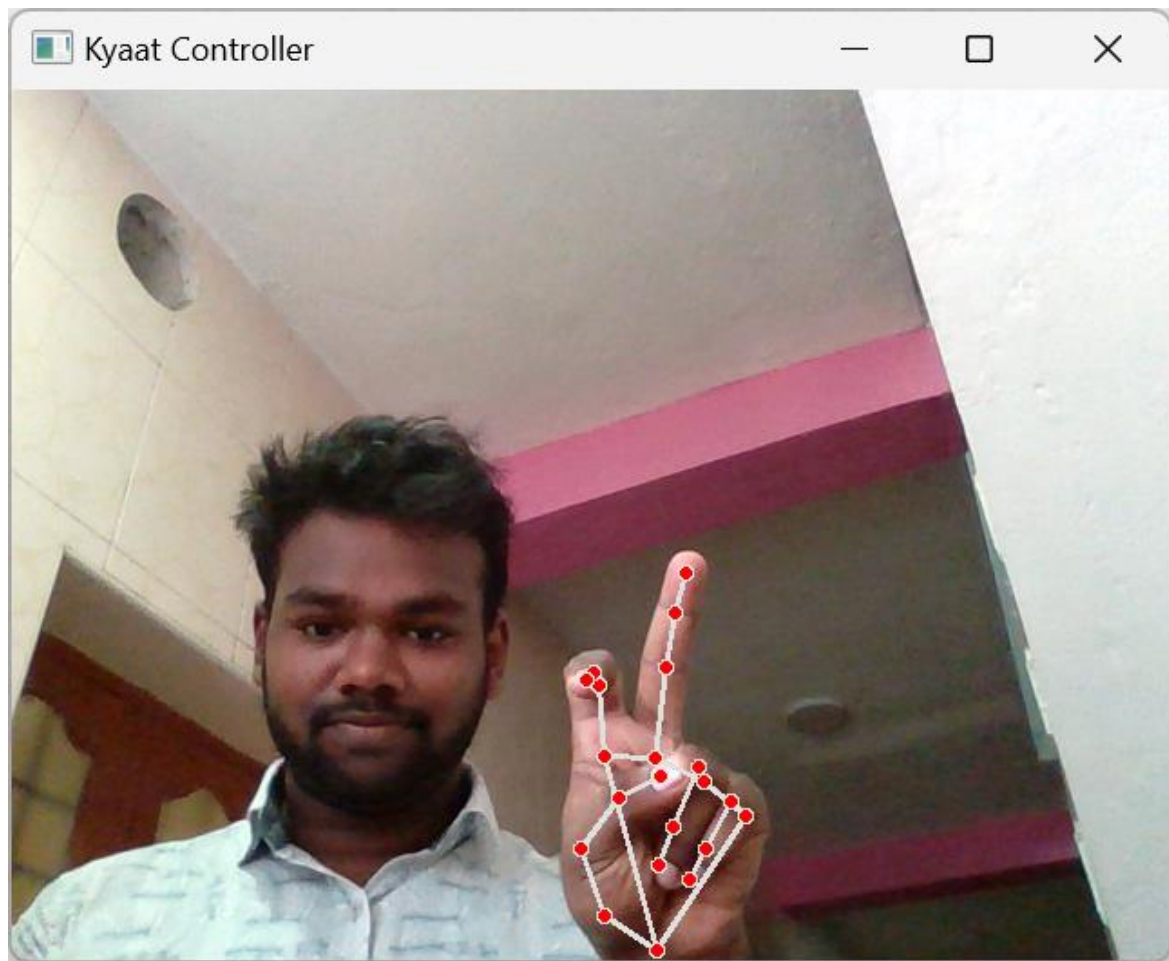
## 4.2 Move Cursor:



### Cursor Movement:

- Users can manipulate the on-screen cursor using natural hand movements, providing a seamless and intuitive method of navigation.
- By tracking the position and orientation of the user's hand, the system translates these gestures into corresponding cursor movements on the display.

### 4.3 Left Click:

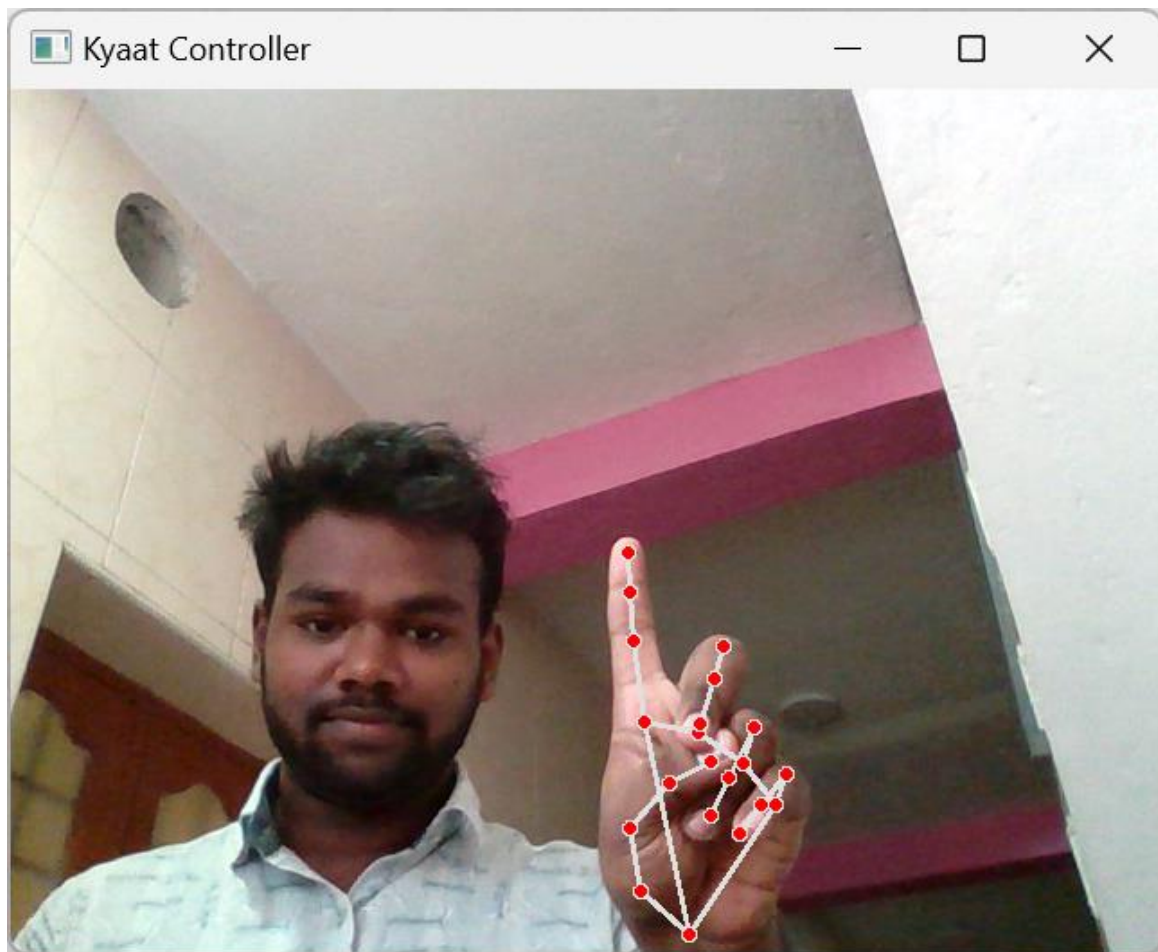


#### Left Click:

- Users can trigger a left-click action by performing a specific hand gesture associated with this action.
- The system recognizes the designated left-click gesture and translates it into a command to interact with interface elements, such as selecting items or activating buttons.
- This feature enables users to perform common interaction tasks without the need for physical mouse devices, enhancing accessibility and convenience.



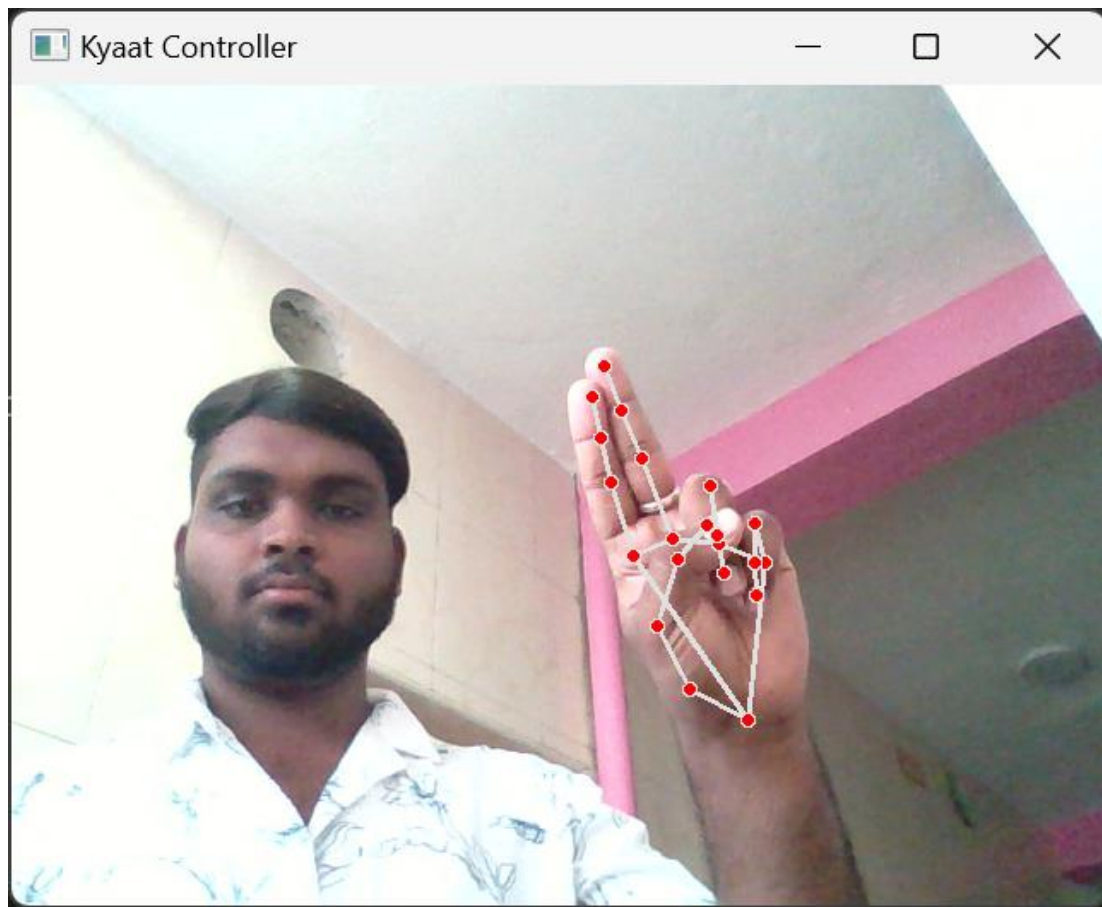
## 4.4 Right Click:



### Right Click:

- Similarly, users can initiate a right-click action by performing a distinct hand gesture tailored to this action.
- The system identifies the right-click gesture and executes corresponding commands, allowing users to access context menus, perform secondary actions, or trigger specific functionalities.
- Like the left-click feature, this capability expands the range of interactions users can perform without traditional input devices, promoting flexibility and user autonomy.

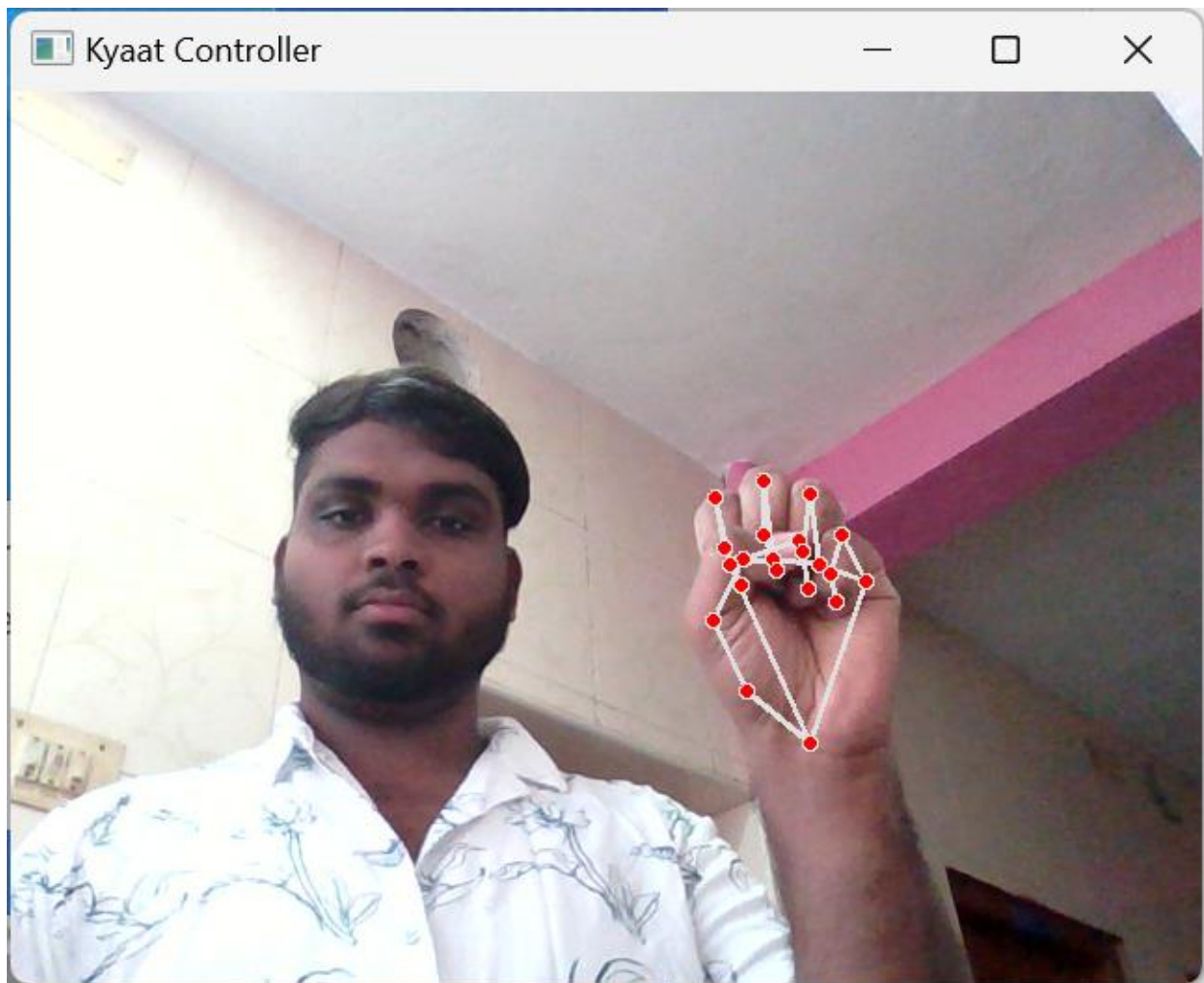
## 4.5 Double Click:



### Double Click:

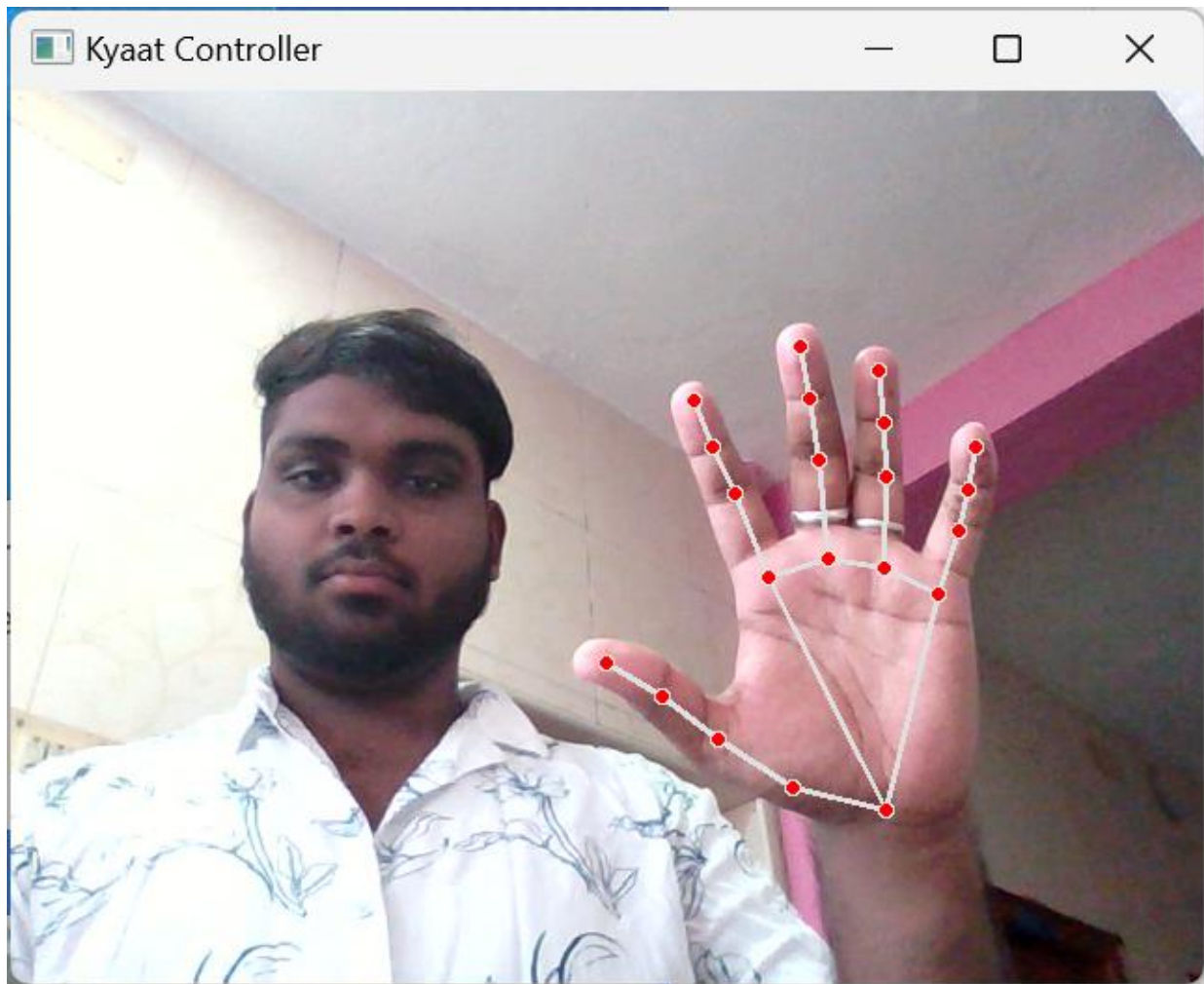
- Double-clicking is facilitated through a specialized hand gesture designed to replicate this action.
- Upon detecting the double-click gesture, the system interprets it as a sequence of two rapid clicks, mimicking the behavior of a physical mouse double-click.
- This functionality enables users to perform actions such as opening files, launching applications, or executing commands that require a rapid succession of clicks, enhancing efficiency and workflow speed.

## 4.6 Drag and Drop:



### Drag and Drop:

- Drag and drop functionality allows users to move digital objects or files within an interface by performing specific hand gestures.
- To initiate a drag action, users typically perform a gesture indicating the selection of an object, followed by a movement gesture to indicate dragging.
- The system detects these gestures and translates them into commands to select and move the desired object.

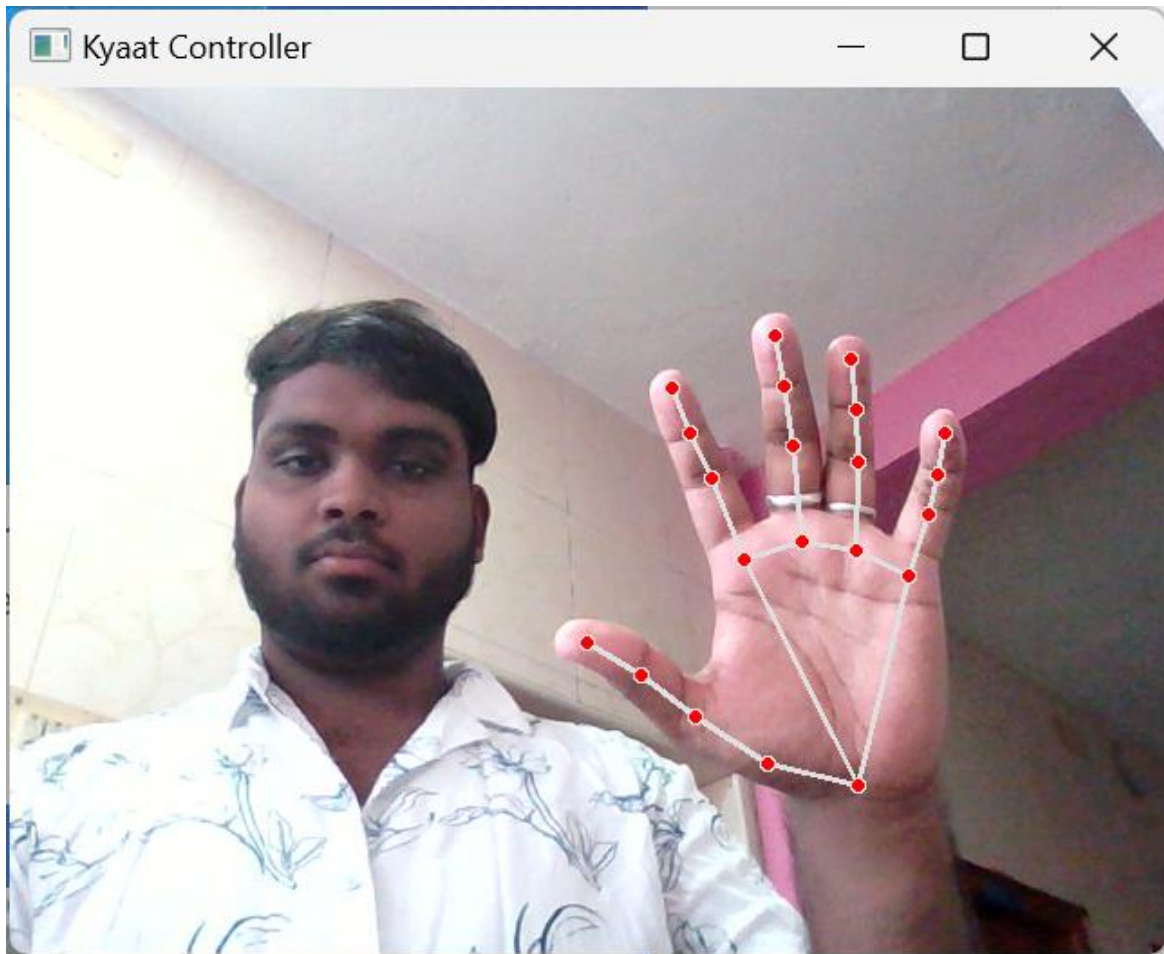


### **Drag and Drop:**

- Once the object is selected, users can then move their hand to simulate dragging the object to a new location on the screen.
- Finally, releasing the gesture indicates dropping the object at the desired destination, completing the drag and drop operation.
- This feature streamlines tasks such as rearranging icons on a desktop, organizing files in a file explorer, or moving elements within a graphical interface.

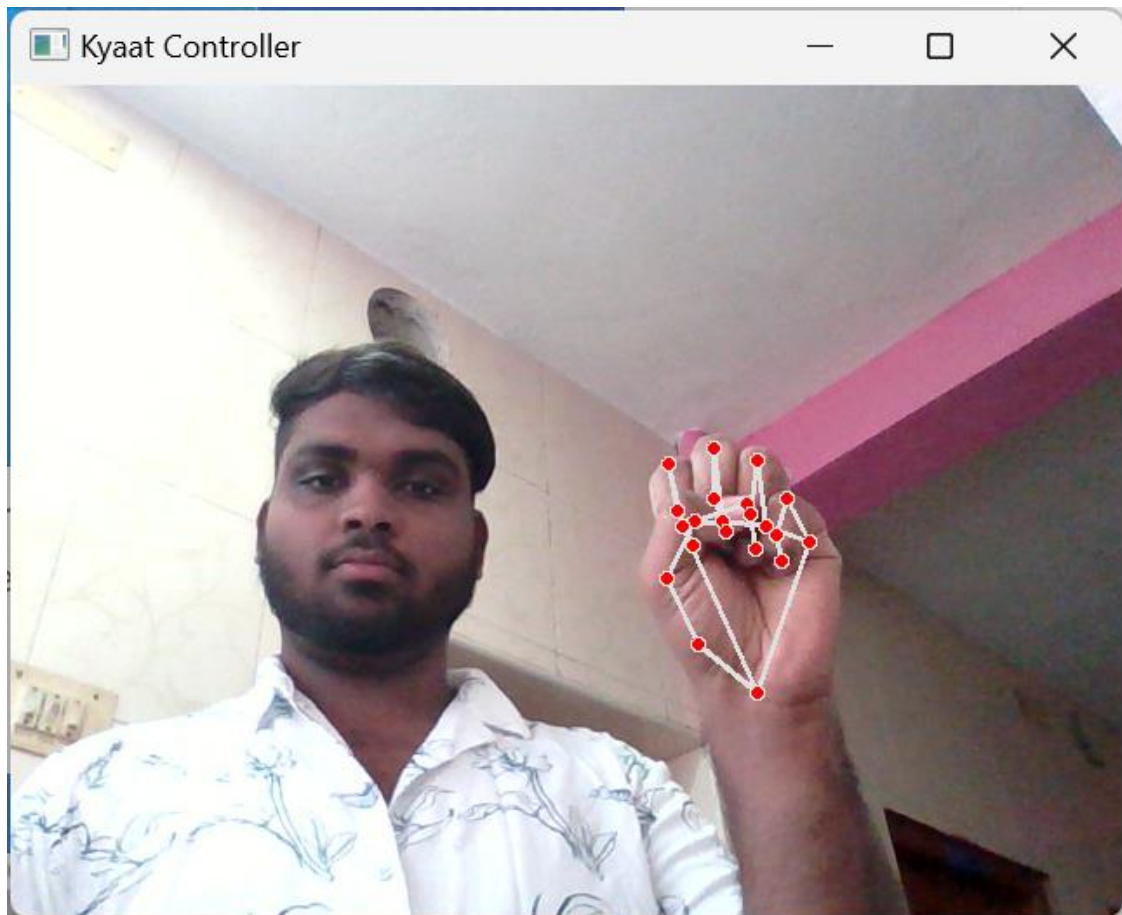


## 4.7 Multiple item Selection:



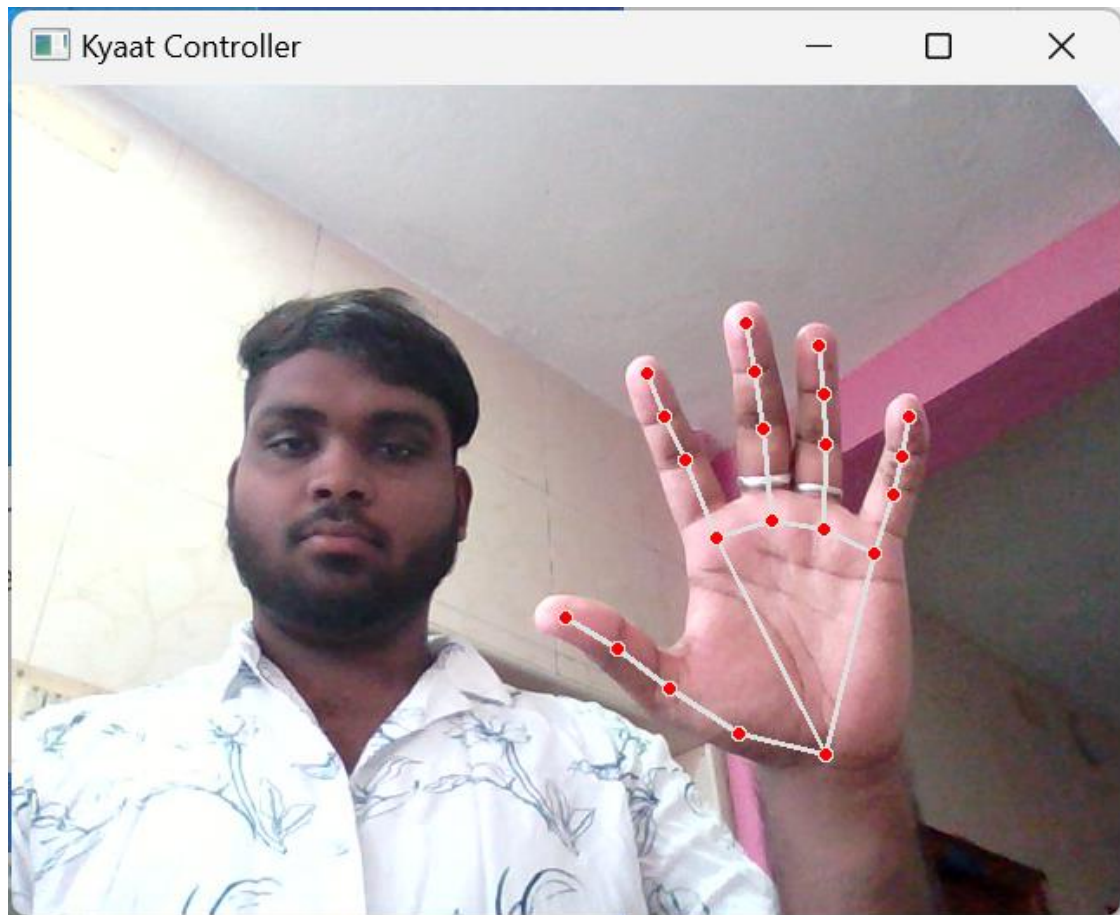
### Multiple Item Selection:

- Multiple item selection allows users to choose more than one item simultaneously within an interface using intuitive hand gestures.
- Users typically perform a gesture that indicates the beginning of the selection process, followed by gestures to indicate the additional items to include in the selection.



### **Multiple Item Selection:**

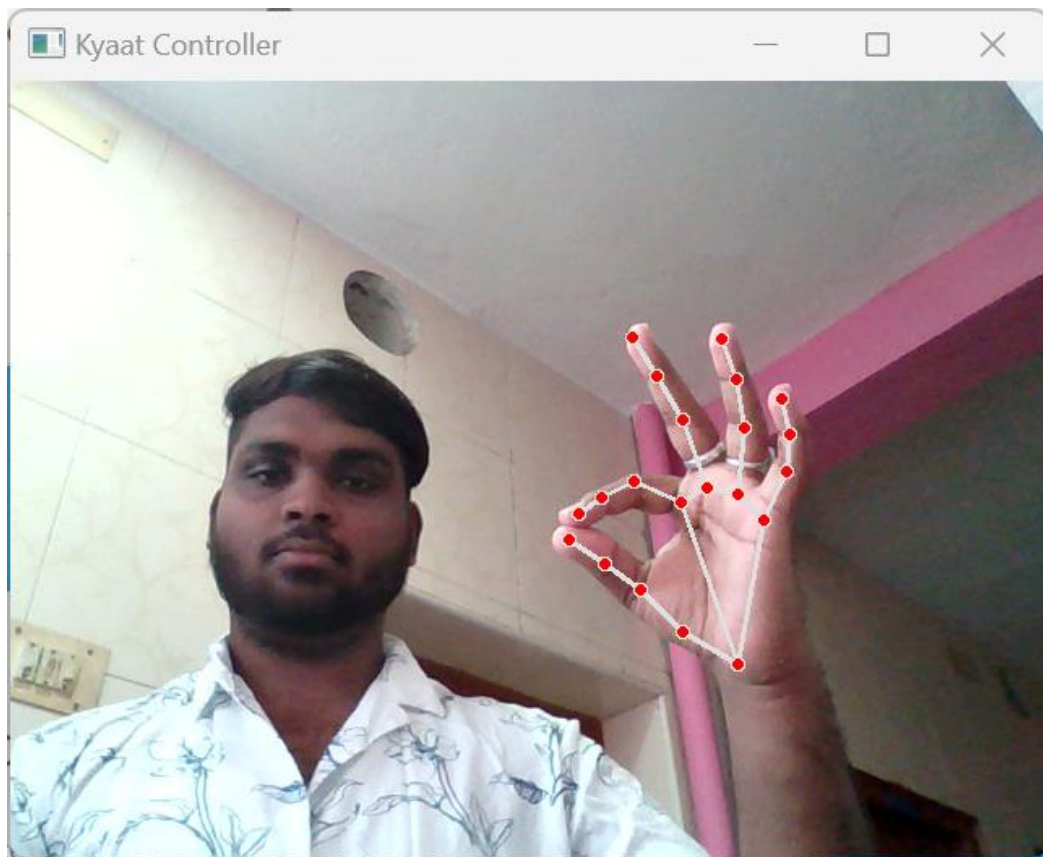
- The system detects these gestures and accumulates the selected items until the user indicates the end of the selection process.
- This feature simplifies tasks that involve selecting multiple items, such as highlighting multiple files for deletion, moving multiple objects at once, or selecting multiple checkboxes in a list.



### **Multiple Item Selection:**

- Multiple item selection allows users to choose more than one item simultaneously within an interface using intuitive hand gestures.
- Users typically perform a gesture that indicates the beginning of the selection process, followed by gestures to indicate the additional items to include in the selection.
- The system detects these gestures and accumulates the selected items until the user indicates the end of the selection process.
- This feature simplifies tasks that involve selecting multiple items, such as highlighting multiple files for deletion, moving multiple objects at once, or selecting multiple checkboxes in a list.

## 4.8 Volume Control:

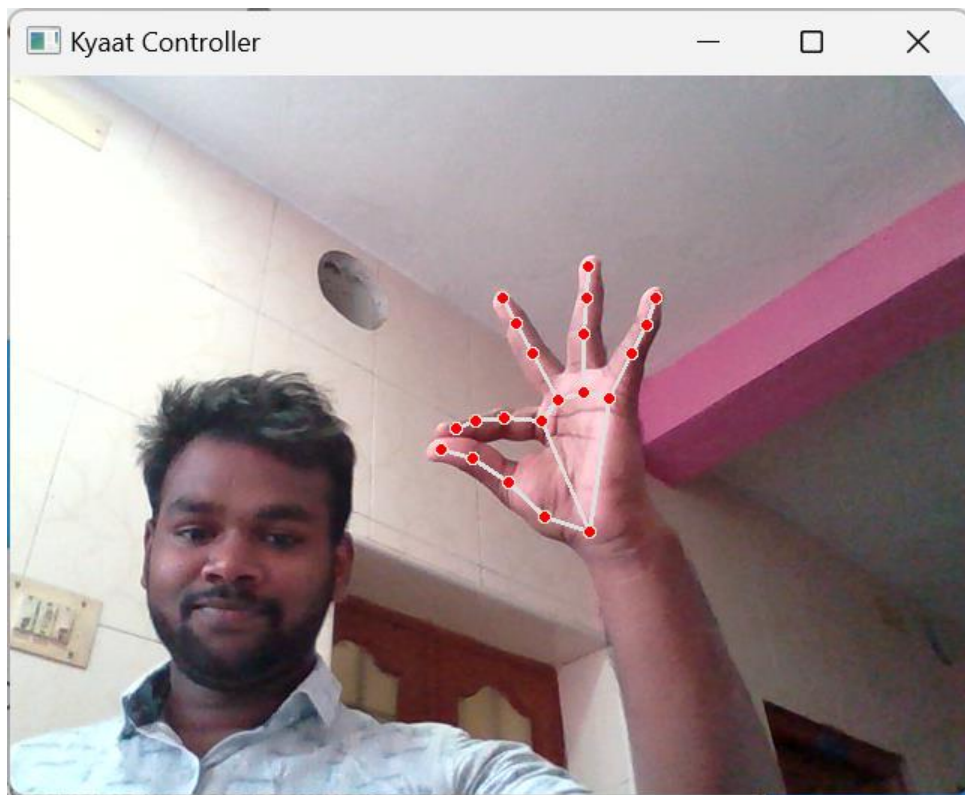


### Volume Control:

- Gesture-based volume control enables users to adjust the volume of audio output devices using hand gestures.
- Users typically perform gestures that represent increasing or decreasing volume levels, such as raising or lowering their hand.
- The system detects these gestures and translates them into commands to adjust the volume accordingly.
- For example, raising the hand may increase the volume, while lowering the hand may decrease it.
- This feature provides a convenient and intuitive way for users to control audio output levels without the need for physical buttons or controls, enhancing accessibility and user experience.



## 4.9 Brightness Control:



### Brightness Control:

- Similar to volume control, brightness control allows users to adjust the brightness of display screens using hand gestures.
- Users perform gestures that signify increasing or decreasing brightness levels, such as spreading their fingers to increase brightness or pinching them together to decrease it.
- The system interprets these gestures and adjusts the screen brightness accordingly.
- This feature is particularly useful in environments with changing lighting conditions, as users can easily adjust the display brightness to optimize visibility and reduce eye strain.

## **CHAPTER 5**

### **5. FUTURE IMPROVEMENTS NEEDED:**

- There are several features and improvements needed in order for the program.
- Smart Movement: Due to the current recognition process are limited within 25cm radius, an adaptive zoom in/out functions are required to improve the covered distance, where it can automatically adjust the focus rate based on the distance between the users and the webcam.
- Better Accuracy & Performance: The response time are heavily relying on the hardware of the machine, this includes the processing speed of the processor, the size of the available RAM, and the available features of webcam.

### **5.1 IMPLEMENTATIONS:**

Many applications benefit from the virtual mouse mechanism. It can be utilised to save space by eliminating the need for a physical mouse, as well as in instances where the physical mouse is not available. This technology reduces the need for a hardware device (mouse) and enhances human-computer interaction.

#### **Major applications:**

- Can alleviate physical stress on the body, which causes back discomfort, poor vision, and poor posture, among other things.
- Because it is not safe to use equipment by touching them during the COVID-19 outbreak because contacting the gadgets could result in the virus spreading, the proposed virtual mouse can be utilised to manage the computer without using the physical mouse.

- Without the need of gadgets, this system can control automation systems and robots.
- Hand movements can be used to draw 2D images on the virtual system.
- Without the usage of a wireless or cable mouse, a virtual mouse can be utilised to play augmented reality and virtual reality games.
- This virtual mouse can be used by those who have difficulty with their hands to handle computer mouse functionalities.
- Using a gesture, you can perform the functions of a traditional mouse quickly and efficiently.

## **CHAPTER 6**

### **6.CONCLUSION:**

- The accuracy and efficiency plays an important role in making the program as useful as an actual physical mouse, a few techniques had to be implemented.
- After implanting such type of application there is big replacement of physical mouse i.e., there is no need of any physical mouse. Each & every movement of physical mouse is done with this Virtual.
- The basic goal of the virtual mouse system is to control the mouse cursor and complete activities without needing a physical mouse by using hand gestures.
- This proposed system is created by using a webcam (or any built-in camera) that recognises hand gestures and hand tip movement and processes these frames to perform the relevant mouse actions.
- The proposed model has been tested for high sophistication, the virtual mouse can be used for real-time applications.
- Because the proposed mouse system may be operated digitally utilising hand gestures rather than the traditional physical mouse, it will be of more value in combating the propagation of viruses like COVID-19 in the current context.

## APPENDIX

### SAMPLE SOURCE CODE

```
import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from google.protobuf.json_format import MessageToDict
import screen_brightness_control as sbcontrol
pyautogui.FAILSAFE = False
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
# Kyaat Encodings
class Gest(IntEnum):
    # Binary Encoded
    """
    Enum for mapping all hand Kyaat to binary number.
    """
    FIST = 0
    PINKY = 1
    RING = 2
    MID = 4
    LAST3 = 7
    INDEX = 8
    FIRST2 = 12
    LAST4 = 15
    THUMB = 16
    PALM = 31
# Extra Mappings
    V_GEST = 33
    TWO_FINGER_CLOSED = 34
    PINCH_MAJOR = 35
    PINCH_MINOR = 36
```

```

# Multi-handedness Labels
class HLabel(IntEnum):
    MINOR = 0
    MAJOR = 1
# Convert Mediapipe Landmarks to recognizable Kyaats
class HandRecog:
    """
    Convert Mediapipe Landmarks to recognizable Kyaats.
    """

    def __init__(self, hand_label):
        """
        Constructs all the necessary attributes for the HandRecog object.
        Parameters
        -----
        finger : int
            Represent Kyaat corresponding to Enum 'Gest',
            stores computed Kyaat for current frame.
        ori_Kyaat : int
            Represent Kyaat corresponding to Enum 'Gest',
            stores Kyaat being used.
        prev_Kyaat : int
            Represent Kyaat corresponding to Enum 'Gest',
            stores Kyaat computed for previous frame.
        frame_count : int
            total no. of frames since 'ori_Kyaat' is updated.
        hand_result : Object
            Landmarks obtained from mediapipe.
        hand_label : int
            Represents multi-handedness corresponding to Enum 'HLabel'.
        """
        self.finger = 0
        self.ori_Kyaat = Gest.PALM
        self.prev_Kyaat = Gest.PALM
        self.frame_count = 0
        self.hand_result = None
        def start(self):
            """

```

Entry point of whole programm, caputres video frame and passes, obtains landmark from mediapipe and passes it to 'handmajor' and 'handminor' for controlling.

"""

```
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 KyaatController.cap.isOpened() and KyaatController.gc_mode:
```

```
        success, image = KyaatController.cap.read()
```

```
        if not success:
```

```
            print("Ignoring empty camera frame.")
```

```
            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:
```

```
            KyaatController.classify_hands(results)
```

```
            handmajor.update_hand_result(KyaatController.hr_major)
```

```
            handminor.update_hand_result(KyaatController.hr_minor)
```

```
            handmajor.set_finger_state()
```

```
            handminor.set_finger_state()
```

```
            gest_name = handminor.get_Kyaat()
```

```
            if gest_name == Gest.PINCH_MINOR:
```

```
                Controller.handle_controls(gest_name, handminor.hand_result)
```

```
            else:
```

```
                gest_name = handmajor.get_Kyaat()
```

```

        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('Kyaat Controller', image)
        if cv2.waitKey(5) & 0xFF == 13:
            break
    KyaatController.cap.release()
    cv2.destroyAllWindows()

gc1 = KyaatController()
gc1.start()

```



## REFERENCES

1. Rao, A.K., Gordon, A.M., 2001. Contribution of tactile information to accuracy in pointing movements. *Exp. Brain Res.* 138, 438–445. [DOI: 10.1007/s002210100717]
2. Masurovsky, A., Chojewski, P., Runde, D., Lafci, M., Przewozny, D., Gaebler, M., 2020. Controller-Free Hand Tracking for Grab-and-Place Tasks in Immersive Virtual Reality: Design Elements and Their Empirical Study. *Multimodal Technol. Interact.* 4, 91. [DOI: 10.3390/mti4040091]
3. Lira, M., Egito, J.H., Dall’Agnol, P.A., Amodio, D.M., Gonçalves, Ó.F., Boggio, P.S., 2017. The influence of skin colour on the experience of ownership in the rubber hand illusion. *Sci. Rep.* 7, 15745. [DOI: 10.1038/s41598-017-16137-3]
4. Inside Facebook Reality Labs: Wrist-based interaction for the next computing platform. (2021). Facebook Technologies. URL: <https://tech.fb.com/inside-facebook-reality-labs-wrist-based-interaction-for-the-next-computing-platform/>
5. Danckert, J., Goodale, M.A., 2001. Superior performance for visually guided pointing in the lower visual field. *Exp. Brain Res.* 137, 303–308. [DOI: 10.1007/s002210000653]
6. Carlton, B. (2021). HaptX Launches True-Contact Haptic Gloves For VR And Robotics. VRScout. URL: <https://vrscout.com/news/haptx-truecontact-haptic-gloves-vr/>
7. Brenton, H., Gillies, M., Ballin, D., Chatting, D., 2005. The uncanny valley: does it exist, in: 19th British HCI Group Annual Conference: Workshop on Human-Animated Character Interaction.
8. Buckingham, G., Michela kakis, E.E., Cole, J., 2016. Perceiving and acting upon weight illusions in the absence of somatosensory information. *J. Neurophysiol.* 115, 1946–1953. [DOI: 10.1152/jn.00587.2015]
9. Katona, J. (2021). A review of human–computer interaction and virtual reality research fields in cognitive Info Communications. *Applied Sciences*, 11(6), 2646.
10. Quam, D.L. (1990). Gesture recognition with a DataGlove. *IEEE Conference on Aerospace and Electronics*, 2, 755–760.

11. Liou, D.-H., Lee, D., & Hsieh, C.-C. (2010). A real-time hand gesture recognition system using motion history image. In Proceedings of the 2010 2nd International Conference on Signal Processing Systems (pp. 1-4). IEEE.
12. Dudhane, S.U. (2013). Cursor control system using hand gesture recognition. IJARCCCE, 2(5).
13. Vinay, K.P. (2016). Cursor control using hand gestures. International Journal of Critical Accounting, 0975–8887.
14. Thomas, L. (2018). Virtual mouse using hand gesture. International Research Journal of Engineering and Technology (IRJET), 5(4).
15. Nandhini, P., Jaya, J., & George, J. (2013). Computer vision system for food quality evaluation—a review. In Proceedings of the 2013 International Conference on Current Trends in Engineering and Technology (ICCTET) (pp. 85–87). Coimbatore, India.
16. Jaya, J., & Thanushkodi, K. (2011). Implementation of certain system for medical image diagnosis. European Journal of Scientific Research, 53(4), 561–567.
17. Nandhini, P., & Jaya, J. (2014). Image segmentation for food quality evaluation using computer vision system. International Journal of Engineering Research and Applications, 4(2), 1–3.
18. Jaya, J., & Thanushkodi, K. (2011). Implementation of classification system for medical images. European Journal of Scientific Research, 53(4), 561–569
19. Google, MP, <https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html>.
20. V. Bazarevsky and G. R. Fan Zhang. On-Device, Media Pipe for Real-Time Hand Tracking.