WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Implementation of Hand Gesture Recognition using OpenCV

Sasikala Dhamodaran [a]*, Pratyush Pranjal Phukan [b], Mayank Singh [b], Shijin Nandakumar [b]

[a] *Professor, NIET, NIMS University, Dr BS Tomar City, NH-11C, Highway Jaipur - Delhi, Chandwaji, Jaipur, Rajasthan, 303121. India.*
[b] *Scholar, NIET, NIMS University, Dr BS Tomar City, NH-11C, Highway Jaipur - Delhi, Chandwaji, Jaipur, Rajasthan, 303121. India.*
DOI: https://doi.org/10.55248/gengpi.5.0524.1324

A B S T R A C T

This work harnesses the capabilities of OpenCV and MediaPipe libraries providing a comprehensive suite of hand gesture interactions using GUIs. The "Volume Control" feature dynamically adjusts system volume in response to hand gestures, utilizing landmarks detected by the MediaPipe hand tracking model. Meanwhile, the "Virtual Mouse" functionality transforms hand movements into cursor control actions, offering left-click capabilities for intuitive navigation. Another noteworthy feature is "Brightness Control", where screen brightness adapts based on the proximity of the hand to the camera that leverages MediaPipe's hand tracking capabilities to gauge distance and adjust screen brightness accordingly.

For image navigation, the "Arrow Control" feature interprets left- and right-hand gestures as commands to navigate through images streamlining the browsing experience by eliminating the need for conventional input devices. Additionally, the script incorporates a robust "Pattern Recognition" security system. This system captures and authenticates handprints, enhancing security measures by leveraging hand gestures for user authentication thereby upon success, the system displays "Access granted", indicating successful verification.

This Python script also utilizes OpenCV and MediaPipe libraries to implement hand gesture-controlled zoom functionality for a webcam video stream. The resized frame is displayed in real-time, reflecting the zoom level controlled by hand gestures. Users can dynamically adjust the zoom level by moving their thumb and index fingers closer or farther apart. This practical application of hand gesture recognition enhances user experience and accessibility, offering interactive control of digital interfaces. Together, these features showcase the versatility of hand gesture recognition in computing applications. By using the power of these libraries, the script enables seamless interaction with computing devices, spanning from system control functionalities like volume and brightness adjustment to more advanced applications like image navigation, security authentication etc.,.

Keywords: OpenCV, MediaPipe, Volume Control, Brightness Control, Arrow Control

1. Introduction

In recent years, the field of human-computer interaction (HCI) has garnered significant attention as a promising avenue for facilitating communication between humans and computers [1]. Human-machine interaction (HMI) specifically entails the exchange and interaction between an individual and a machine through a user interface. A novel approach within this domain is hand gesture recognition, which has emerged as a transformative technique for HCI due to its inherent automation, naturalness, and ease of use, eliminating the need for traditional input devices such as keyboards and mouse [2]. Through the utilization of hand gestures, users can efficiently interact with computers, accessing and conveying information in a streamlined manner. Consequently, HCI technology holds immense potential for enhancing the interface between individuals and computer systems.

A key advantage of employing hand gestures in HCI lies in their capability to facilitate remote communication [3]. However, the effectiveness of HCI hand gestures hinges upon the accurate measurement and tracking of hand configurations by computer systems. The primary objective of the proposed system is to discern specific human hand gestures, enabling their utilization for information communication or the control of various devices and robots in practical applications [4]. Moreover, the integration of hand gesture recognition technology into HCI systems necessitates robust algorithms and precise data acquisition methods to ensure optimal performance. These systems rely on sophisticated computer vision techniques to capture and interpret hand movements accurately. By employing advanced machine learning (ML) algorithms, such systems can effectively recognize and classify diverse hand gestures, enabling seamless interaction between users and computers. Consequently, the development of reliable hand gesture recognition systems represents a significant advancement in HCI technology, offering new avenues for intuitive and efficient human-computer interaction [5]. Additionally, real-time tracking of hand gestures enables dynamic control of devices and applications, enhancing user experience and productivity [14]. Human–computer collaborations are ominously inclined based on communication used to regulate the computer. Systems centred on eye movement or gesture control hunt down can permit few functions of the keyboard or mice to be initiated or switched. Papers [27] & [28] compared the traditional computer mouse and its handling with a cost-effective hand gesture detection and processing system called Leap Motion.

1.1 Objectives of the Project

Accessibility for Individuals with Disabilities.

Interactive Presentations and Exhibitions.

Hands-Free Operation in Work Environments.

Security and Access Control.

Assistive Technology for Elderly Individuals.

Entertainment and Gaming.

2. Literature Survey

Hand gesture recognition has emerged as a prominent technology in recent years, offering a promising avenue for human-computer interaction (HCI)[1]. This approach allows users to interact with computers and digital devices through natural hand movements, eliminating the need for conventional input devices like keyboards and mouse. Through hand gestures captured via webcam feed, users can perform various tasks such as adjusting volume, emulating a virtual mouse, changing screen brightness, and navigating using arrow keys[2]. Leveraging computer vision algorithms and ML techniques, hand gesture recognition systems can detect and track hand movements in real-time, enabling intuitive control over digital interfaces [3].

The project utilizes key computer vision libraries, namely OpenCV and MediaPipe, to implement hand tracking and gesture recognition functionalities [7][22]. OpenCV provides a comprehensive set of tools for image and video processing, facilitating tasks such as webcam capture and real-time object detection. MediaPipe offers specialized models and APIs tailored for hand tracking, enabling accurate localization and tracking of hand landmarks, which are essential for gesture recognition.

Gesture-based control mechanisms showcased in the project illustrate the versatility and practical applications of HCI technologies [1][10][22]. These functionalities enhance user experience and accessibility across different domains, from multimedia control to productivity enhancement.

Users have the ability to execute a range of functions, including adjusting volume, simulating a virtual mouse, altering screen brightness, by utilizing hand gestures recorded through a webcam feed [11]. Furthermore, the project demonstrates the integration of hand gesture recognition into security systems, showcasing its potential in biometric authentication and access control. By capturing and comparing hand patterns, the system can authenticate users based on their unique hand gestures, adding an additional layer of security to digital systems [12]. Some applications of hand gesture use sensors to capture data about the movement of the hands. Document [13] is study about Gesture recognition with an information glove, which uses a information glove to keep track of the hand gestures. A real time hand gesture recognition system using motion history image [14] involves exploring advanced techniques which has been used for pattern recognition.

3. Requirements

3.1 Hardware Requirements

1. Webcam: Resolution of at least 640x480 pixels. 2. Processor: Dual-core or higher CPU (e.g., Intel Core i3, AMD Ryzen 3 series). 3. Memory (RAM): Minimum 4GB RAM, preferably 8GB or more for optimal performance. 4. Graphics Processing Unit (GPU): Optional, but expected for improved performance, especially with CUDA (NVIDIA) or OpenCL (AMD) support. 5. Storage: At least 100GB of free disk space, SSD preferred for faster read/write speeds. 6. Operating System: Compatibility with Windows, macOS, or Linux. 7. Optional Accessories: Microphones, external speakers, keyboards, or mice depending on project functionalities.

3.2 Software Requirements

1. Python: Install the latest version of Python that provides a rich ecosystem of libraries and tools for data analysis, ML, and web development. 2. Integrated Development Environment (IDE): Choose an IDE such as PyCharm, or Visual Studio Code to write and execute Python code efficiently.

3.3 Libraries Used

1. Python Libraries: cv2, mediapipe, math, numpy, ctypes, pyautogui, time, screen_brightness_control, autopy, keyboard, subprocess, tkinter. 2. External Libraries: cvzone, pycaw, comtypes.

4. System Modules used in this Project

4.1 OpenCV

for tasks such as image and video processing, object detection and recognition, feature extraction, and more.

4.2 Google MediaPipe

Framework by Google for building ML based pipelines for various perception tasks, primarily focusing on computer vision and audio processing providing a comprehensive set of prebuilt components and tools facilitating the development of real-time applications, and above all those related to gesture recognition, hand tracking, pose estimation, and facial recognition. "MediaPipe graphs." that encapsulate various processing steps such as data ingestion, feature extraction, inference, and visualization. Customization and Extension as well as prebuilt solutions for common tasks. Developers can modify existing components or create new ones to tailor the pipeline to their specific requirements. Integration with TensorFlow Lite, Google's framework for deploying ML models on mobile and embedded devices enables developers to leverage the extensive ecosystem of TensorFlow Lite models within their MediaPipe pipelines.

4.3 "math" module

4.4 NumPy

4.5 "ctypes" module

Supports a foreign function interface (FFI) for calling functions in DLLs (Dynamic Link Libraries) or shared libraries from Python. a. cast(). b. POINTER().

4.6 "comtypes" module

CLSCTX_ALL.

4.7 PyCaw

a. AudioUtilities. b. IAudioEndpointVolume.

4.8 "time" module.

4.9 Autopy

4.10 "sys" module

4.11 "subprocess" module

for process creation, asynchronous execution. inter-process communication, handling output and integration with GUI Application.

4.12 "keyboard" module

for KeyPress and Conditional Key Simulations, Dynamic Input Generation, and integration with Hand Tracking System.

4.13 "cvzone" module

for Hand Detection and Tracking, Finger Tracking, Distance Calculation, Frame Processing, Hand Landmark Visualization, and Hand Gesture Recognition.

4.14 "Screen brightness control" module

for set_brightness(intensity), Dynamic Adjustment, Integration with Hand Tracking, and User Feedback.

4.15 "Tkinter" module

for Widget Creation, Event Handling, Layout Management, Customization, Mainloop and Integration with Other Libraries.

5. Landmarks

5.1 Defining hand landmarks

After confirming the hand axis the next task is to define the full hand land mark capture. The hand land mark are captured by the predefined MediaPipe 21 locating point. These 21 joints or locating point are to be tracked by the MediaPipe. If there is no landmark found then the default value that is empty will be printed. That means no action need to be performed. Otherwise, it returns the coordinate value of X, Y and Z axis. The thumb finger is not included in the hand landmark region. A loop is created in between 21 point and return the tip point value by drawing graph line in between finger. When the finger is folded up or down the finger lines are also up or down and the corresponding graph lines are up or down. The gap in between the finger is not important in this point. The gap involving amidst of fingers may turn into 5 cms. Otherwise it gets back to the initial position of finding. After locating height, width and centre of the hand landmark the decimal coordinate relative to the index is converted according to each image. After capturing the total hand landmarks a green rectangle box appeared around the hand where the mouse operations can be performed.[1][3][6][7][8].

Fig. 1 - Co-ordinates of hand landmark.

Fig. 2 - Capturing the hand landmarks.

5.2 Explanation of Hand Tracking Module

The Hand Tracking module sets up a real-time hand detection and tracking system using OpenCV and MediaPipe libraries[3][6][7]. It continuously processes frames from a video stream, detects hands and landmarks, and displays the annotated frames with FPS information [14]. **Functioning of the module**: 1. Importing libraries: The script imports the necessary libraries for image processing (‘cv2’), hand tracking (‘mediapipe’), time manipulation (‘time’), mathematical operations (‘math’), and numerical computing (‘numpy’). 2. ‘handDetector’ class: a. This class is defined to encapsulate the functionality related to hand detection and tracking. b. It has an ‘__init__’ method to initialize the hand detector with specified parameters. c. Methods like ‘findHands’, ‘findPosition’, ‘fingersUp’, and ‘findDistance’ are defined within this class to perform various tasks related to hand detection, landmark tracking, and finger position detection. 3. ‘main’ function: a. This function serves as the entry point of the script. b. It initializes video capture using ‘cv2.VideoCapture’. c. Creates an instance of the ‘handDetector’ class. d. Starts an infinite loop to continuously process frames from the video stream. e. Within the loop: i. Reads a frame from the video stream. ii. Passes the frame to the ‘findHands’ method to detect hands and landmarks. iv. Passes the detected landmarks to the ‘findPosition’ method to find the positions and bounding box of the hand. v. Prints the position of a specific landmark (in this case, the 4th landmark). vi. Calculates and displays the frames per second (FPS) on the image. vii. Displays the annotated image with hands and landmarks using ‘cv2.imshow’. viii. Waits for a key press using ‘cv2.waitKey’. 4. Execution: The ‘main’ function is called if the script is executed directly (‘if __name__ == "__main__": main()’).

5.3 Hand Segmentation Methods

Research work [9] states that segmentation is the process of converting a grayscale image into a binary image with just two objects: the backdrop and the hand. Otsu points out that this approach is utilized for segmentation, converting grayscale photos into binary images with a backdrop or hands on it. To choose a suitable threshold of gray level to separate the hand from the backdrop, for example, such that no portion of the hand has the background and no part of the background has the hand, excellent segmentation is required. Generally speaking, the kind of picture and the application domains have a major influence on the choice of segmentation method. After testing, this algorithm produced good segmentation results for the hand gesture. It is a method of automatic threshold selection that is nonparametric and unsupervised. The basic theory of segmentation of hand is given by using the equation as follows:

$p_i = \frac{n_i}{MN}, p_i \geq 0, \sum_{i=1}^{L} p_i = 1$ (1)

In a digital image with L gray levels ranging from 1 to L, denoted by L = [1, 2, 3, ..., L], the number of pixels at each level k is represented by N_k, and the total number of pixels in the image is $N = N_1 + N_2 + N_3 + ... + N_L$. The normalized histogram is composed of $\frac{N^k}{N}$. This formulation is based on Rafael's work [17].

To implement this formula (1), each frame of a real-time video is treated as a collection of pixels compared to one another. Since each frame contains identical colour values for pixels, a pixel is considered redundant and thus eliminated. This process applies to all pixels within each frame, and frames are compared over time. If any pixels between two frames differ, the pixel in the first frame retains its value.

Otsu's thresholding method involves iterating through all possible threshold values and measuring the spread of pixel values on either side of the threshold, distinguishing between foreground and background pixels. The goal is to find the threshold value where the sum of foreground and background spreads is minimized. Figure 3 illustrates how a hand image is extracted from real-time video using a grayscale filter and then processed with Otsu's algorithm to generate the thresholded image. As mentioned earlier, Otsu's thresholding method is restricted to processing only bimodal images that are images characterized by histograms containing two distinct peaks. In such cases, Otsu's algorithm selects a threshold value positioned between these peaks, automatically determining it from the image histogram. This approach is effective when converting RGB-format frames from a video into grayscale.

5.4 Track Gesture of Hand Using Haar-Cascade Classifier

To ensure continuous detection of the hand, especially when it's stationary or not in motion, the program relies on ML to learn the hand image. In this project, ML techniques are employed prior to image processing to track hand gestures. As described in documents [18] & [20], cascade classifiers serve as program features enabling the machine to precisely identify the objects requiring classification in each frame of images. One commonly used example in this project is the Haar-cascaded type. In research work [20], Viola further suggests that the Haar-cascaded classifier is a straightforward option for ML systems to detect various objects.

Fig. 3 - Contrast grayscale image and thresholded image.

5.5 Region of Interest

Denotes a specific area within an image that is of interest while disregarding the surrounding background region. As suggested in script [18], in order to accurately detect hand recognition, the detected region must align with the designated ROI. Essentially, as the classifier tracks the target object, such as the bounding rectangle encompassing the hand contour, the hand area must coincide with the ROI. By employing a conditional method to ascertain overlapping regions, the system can determine whether the selected region, namely the rectangle outlining the hand contour, overlaps with other active regions, namely the hand itself.

By translating this theoretical concept of ROI into simulation and hardware implementation, the process of detecting the presence of a hand becomes significantly streamlined compared to alternative methods, as discussed in [21]. Figure 4 illustrates the criteria for ROI overlap, crucial for ensuring the program's ability to identify hand gestures accurately.

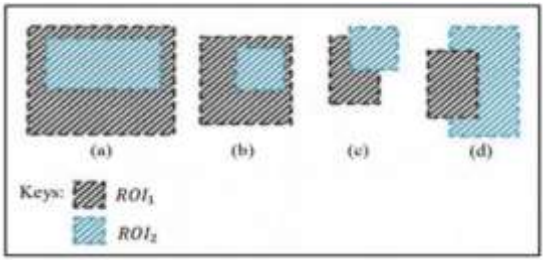

Fig. 4 - Overlapping Region of Interest.

In this project, the Region of Interest (ROI) [22] is employed to observe hand gestures, primarily due to its capability to detect the overlapping of two regions within it. Furthermore, the system is designed to detect only two outputs for each hand gesture recognition. This is elaborated upon in Table 1 below.

Table 1 - Application of ROI.

Application	**Column A (*t*)**
ROI	Region implemented for
ROI_1	An active region, in this case the hand itself
ROI_2	The rectangle sub-window on the screen. The region is static or not movable outside the frame of the real-time video and it highlights the area that involves

5.6 Implementation on Simulation and Hardware prototype

Hand gesture detection in this project relies on measuring the spatial occupancy within the area bounded by the convex hull and the hand contour. The convex hull is utilized to delineate the hand's outline by generating a descriptor that encompasses the smallest convex set enclosing its edges. For this purpose, the project utilizes appropriate files from the OpenCV library, which offers a wide array of image processing techniques. Referring to the flowchart depicted in Figure, the creation of a Python file for both simulating and hardware prototyping can be executed. The primary function of the main program is to detect hand appearance and gestures utilizing the ROI theory. To track the rectangle surrounding the hand contour, the program needs to ascertain whether the selected area of interest, represented by the rectangle in this project, overlaps with the hand detection. If an overlap occurs, it signifies a gesture type, such as a number or sign, within the ROI. If there's no overlap (outside the ROI), the program prompts "Show your hand in a box". Planning the ROI for this project involves identifying a set of coordinates to define the bounding rectangle area around the hand contour. Since there's an area of the bounding rectangle, there exists an ROI to be implemented for each type of gesture.

After generating the Python main file, the simulation commences to test hand detection and recognize the type of gestures based on the selected ROI. To run the Python file, the OpenCV environment must be linked and executed through the command window.

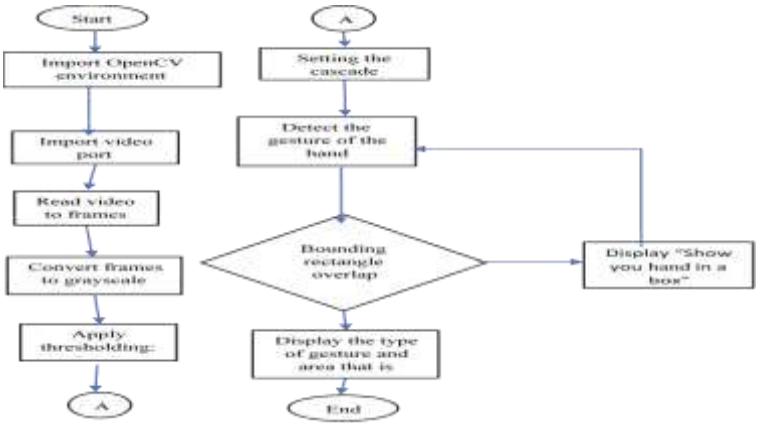

Fig. 5 - Flowchart of Python file for hand gesture recognition.

Subsequently, similar procedures are applied for the hardware prototype. In this prototype, a USB video webcam, as used by Nayana [25], will be employed. Modifications are required in the Python core source code, particularly in the video capture section, to switch the camera from the laptop webcam to the USB video webcam. The procedures for detecting and analyzing hand gestures using the USB video webcam remain consistent with those during the simulation phase. It's worth noting that the OpenCV environment is once again utilized to detect the hand via the Haar-cascade classifier.

6. Methodology

6.1 Volume Control using Hand Gesture

The methodology for Volume Control involves the following steps:

1. **Import necessary libraries**. OpenCV (cv2), Mediapipe (mp), NumPy (np), ctypes and comtypes, and pycaw functionalities.

2. **Initialize volume control**: Next, the code initializes the volume control functionality using the PyCaw library. It retrieves the available audio output devices (speakers) and activates the audio endpoint volume interface. This interface allows the code to interact with the system's audio controls programmatically [3].

3. **Setup webcam**: The code configures the webcam for capturing video input. It sets the desired dimensions (width and height) for the video stream using the set() method of the VideoCapture object. This step ensures that the webcam captures video frames with the specified resolution (wCam x hCam).

4. **Initialize Mediapipe hand landmark model**: The code initializes the Mediapipe hand landmark model for hand detection and tracking. It sets parameters such as model complexity, minimum detection confidence, and minimum tracking confidence to control the performance and accuracy of the hand tracking process.

5. **Main loop**: The code enters a loop to continuously capture frames from the webcam and process them for hand gesture recognition. This loop ensures real-time processing of video input and enables the system to respond dynamically to hand movements.

6. **Hand detection and landmark extraction**: Within the loop, each captured frame is converted to RGB format to facilitate processing with the Mediapipe hand landmark model. The model is applied to the frame to detect and track hand landmarks, such as key points corresponding to fingers and palm regions. These landmarks' coordinates are extracted to analyse hand gestures and perform further actions.

7. **Hand gesture analysis**: Based on the extracted hand landmarks' coordinates, the code analyses hand gestures to identify specific actions, such as pinching or spreading fingers. For example, it calculates the distance between the thumb and index finger landmarks to determine if a pinching gesture is detected.

8. **Volume control**: If a pinching gesture is detected, the code maps the distance between the thumb and index finger to a corresponding volume level. It adjusts the system's master volume using the volume control interface obtained earlier, ensuring that the user's hand gestures directly control the audio output volume.

9. **Visual feedback**: To provide visual feedback to the user, the code visualizes the detected hand landmarks, gesture recognition results, and volume control feedback on the captured video frames. It uses OpenCV drawing functions to overlay graphical elements, such as circles, lines, rectangles, and text, on the video stream, making it easier for the user to understand the system's behaviour.

10. **Display and exit**: Finally, the code displays the processed video frames in a window named "handDetector" using the imshow() function of OpenCV. It continues looping until the user presses the 'q' key, at which point the webcam is released, and the program exits gracefully. This ensures that the system can be easily stopped when the user no longer requires volume control through hand gestures.

6.2 Virtual Mouse

The methodology for Virtual Mouse involves the following steps [15][22]:

1. **Import necessary libraries**: The code imports required libraries, including OpenCV (cv2) for image processing, NumPy (np) for numerical computations, time for time related operations, Hand Tracking module for hand detection and tracking functionalities, and autopy for controlling the mouse cursor[16] and simulating mouse clicks.

2. **Variables Declaration**: Key variables such as pTime (previous time), width, height, frameR (frame rate), smoothening factor, and coordinate variables are initialized to appropriate values.

3. **Webcam Setup**: The code initializes the webcam (cap) to capture video feed. It adjusts the webcam's width and height parameters using the set() method to ensure compatibility with subsequent processing.

4. **Hand Detection and Tracking**: Using the Hand Tracking module, the code detects and tracks the user's hand within the captured video frames. It identifies the position of landmarks (finger joints) on the hand and stores the results in lmlist and bbox variables.

5. **Gesture Recognition**: Based on the detected finger positions, the code analyses the hand gestures to determine the user's intended actions. It checks if specific fingers (forefinger and middle finger) are raised or lowered to differentiate between cursor movement and mouse click actions.

6. **Cursor Movement**: If the forefinger is raised while the middle finger is lowered, the code maps the hand's position to the screen's coordinates. It smoothes the cursor movement using interpolation and moves the mouse cursor accordingly using autopy's mouse.move() function[16].

7. **Mouse Click**: If both the forefinger and middle finger are raised, indicating a click gesture, the code calculates the distance between the fingers. If the distance is below a certain threshold, it simulates a mouse click using autopy's mouse.click() function.

8. **Frame Rate Calculation**: The code calculates the frame rate (fps) based on the time taken to process each frame and displays it on the output image using the cv2.putText() function.

9. **Display and User Interaction**: The processed video frames are displayed in a window named "Image" using cv2.imshow(). The code waits for a key press event using cv2.waitKey(1), allowing the user to exit the program by pressing any key.

10. **Continual Execution**: The code continuously loops through the above steps, ensuring real-time hand gesture recognition and mouse control functionalities until the user exits the program.

6.3 Brightness Controls

The methodology of Brightness Control involves the following steps[11]:

1. **Hand Tracking Model Initialization:** The hand tracking model from MediaPipe is initialized with specific parameters such as model complexity, detection and tracking confidence thresholds, and the maximum number of hands to detect.

2. **Video Capture**: The code starts capturing video frames from the webcam using OpenCV's VideoCapture class.

3. **Frame Processing**: The captured frame is flipped horizontally to prevent mirror inversion. The frame's color space is converted from BGR to RGB, as MediaPipe requires RGB input.

4. **Hand Landmark Detection**: The processed RGB frame is fed into the hand tracking model to detect hand landmarks. If hand landmarks are detected, the x and y coordinates of each landmark are extracted and stored in a list.

5. **Gesture Recognition**: The code identifies the positions of the thumb and index finger landmarks. Circles are drawn at the tips[23] of the thumb and index finger, and a line is drawn between them. The Euclidean distance between the thumb and index finger is calculated.

6. **Brightness Control**: The Euclidean distance is used to interpolate the brightness level based on a predefined range of distances. The interpolated brightness level is mapped to a range of screen brightness values (0 to 100). The screen brightness is adjusted accordingly using the screen_brightness_control library.

7. **Display**: The current brightness percentage is displayed on the frame. The modified frame with drawn landmarks and brightness information is displayed in a window using OpenCV's imshow function.

8. **Termination**: The loop continues until the 'q' key is pressed, upon which the program releases the video capture resources and closes all windows.

6.4 Arrow-key Controls

The methodology of Arrow-key Controls involves the following steps[8][15]:

1. **Hand Detection Initialization**: The MediaPipe Hands model is initialized with specific parameters for detecting hand landmarks, including confidence thresholds and the maximum number of hands to detect.

2. **Video Capture**: The code captures a video stream from the webcam using OpenCV's VideoCapture functionality.

3. **Hand Gesture Recognition Loop**: Within a loop, each frame from the video stream is processed. The frame is converted to RGB color space to match the requirements of the MediaPipe model. The model is used to detect hands in the frame. If hands are detected, the code checks the landmarks to determine if it's the left or right hand. Based on the detected hand, a simulated key press is triggered (left arrow for right hand, right arrow for left hand) using the Keyboard library. A flag is set to prevent multiple key presses during the same hand gesture.

4. **Display**: The processed frame with hand landmarks and recognition information is displayed in a window using OpenCV's imshow function.

5. **Termination**: The loop continues until the 'q' key is pressed, upon which the program releases the video capture resources and closes all windows:

6.2 Pattern Recognition (Biometric)

The methodology of Pattern Recognition involves the following steps [26]:

1. **Library Import**: The necessary libraries, including OpenCV, MediaPipe, and NumPy, are imported.

2. **Class Definition**: The HandprintSecuritySystem class is defined to encapsulate the functionality for capturing and authenticating handprints.

3. **Initialization**: In the constructor (__init__) of the above mentioned class: The MediaPipe Hands model is initialized with specific parameters for hand detection. An empty list (saved_handprints) is created to store saved handprints.

4. **Handprint Capture**: Hand landmarks are detected in the provided image using the MediaPipe Hands model. If a hand is detected: The landmarks are appended to the list of saved handprints. A message indicating successful handprint capture is printed, and the captured handprint is displayed in a window. 5. **Handprint Authentication**: Hand landmarks are detected in the provided image using the MediaPipe Hands model. If a hand is detected: The detected landmarks are compared with the saved handprints for authentication using Euclidean distance. If a match is found, a message indicating successful authentication is printed; otherwise, an authentication failure message is printed. If no hand is detected, a corresponding message is printed.

6. **Landmark Detection**: The provided image is converted to RGB color space. The MediaPipe Hands model processes the image to detect hand landmarks. If one or more hands are detected: i. The normalized coordinates of the landmarks are extracted and stored. ii. The landmarks and the corresponding MediaPipe hand object are returned.

7. **Landmark Comparison**: Euclidean distances between landmarks of two handprints are calculated using NumPy. The mean distance is computed, and if it falls below a predefined threshold, the handprints are considered similar, indicating authentication success.

8. **Main Functionality**: An instance of that class is created. Video capture from the webcam is initiated. Within a loop: i. Frames are continuously captured from the webcam. ii. If the 'c' key is pressed, the current handprint is captured and saved. iii. Handprint authentication is performed on each frame, and appropriate actions are taken based on the result. iv. The frame with overlays is displayed in a window.

9. **Termination**: The loop continues until the 'q' key is pressed, upon which the program releases the video capture resources and closes all windows.

6.5 Zoom-In and Zoom-Out Controls

The methodology used in Zoom-in and Zoom-out controls [19][24] involves the following steps:

1. **Video Capture Setup**. It adjusts the resolution of the captured video to 1280x720 pixels using the set method, ensuring high-quality input for hand gesture detection.

2. **Hand Detection Initialization**: An instance of the HandDetector class from the cvzone.HandTrackingModule is created with a specified detection confidence threshold of 0.7. This detector will be used to locate and track hands in each frame of the video feed.

3. **Continuous Frame Processing Loop**: Within a continuous loop, the code reads frames from the video capture object using the read method. Each frame is processed sequentially to detect and track hand gestures.

4. **Hand Gesture Detection and Image Loading**: The findHands method of the HandDetector object is applied to each frame to detect hands. Additionally, an image (img1) is loaded from a specified file path, which will be manipulated based on the hand gestures detected.

5. **Hand Gesture Analysis and Image Scaling**: If two hands with two fingers extended on each hand are detected (len(hands)==2), the code calculates the distance between the centers of the hands using the findDistance method of the HandDetector. If it's the first time the distance is calculated (startDist is None), the initial distance between the hands is stored (startDist). Subsequently, the code calculates the difference in distance from the initial distance, representing the scaling factor (scale) for the image.

6. **Image Resizing and Overlay**: The calculated scaling factor is used to resize the loaded image (img1) to a new size based on the difference in distance between the hands. The new size ensures that the image maintains its aspect ratio and is resized to even dimensions. The resized image is then overlaid onto the original frame at the position determined by the center of the hands (cx, cy).

7. **Display and User Interaction**: The resulting frame with the overlaid image is displayed in a window titled "Image" using the imshow method of OpenCV. The loop continues until a key is pressed, where each iteration waits for 1 millisecond for a key event (cv2.waitKey(1)). This allows for real-time interaction with the application.

7. Output Generation

7.1 The methodology for Volume Control using Hand Gesture: was discussed in detail in point 6.1 of Methodology from 6.

Fig. 6 – (a) Volume 100% [Distance between index and thumb]; (b) Volume 0% [Mute]

7.2 The methodology for Virtual Mouse: was discussed in detail in point 6.2 of Methodology from 6.

Fig. 7 – (a) Movement of the cursor using index finger; (b) Left Click [Tapping the index & the middle finger]

7.3 The methodology for Brightness Control: was discussed in detail in point 6.3 of Methodology from 6.

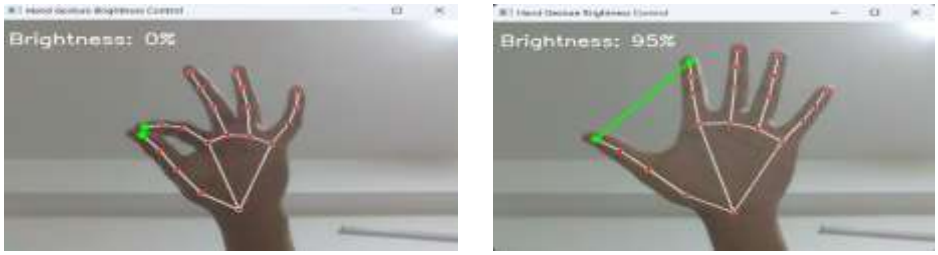

Fig. 8 – (a) Brightness 0% [Distance between index and thumb]; (b) Brightness 95% [Almost Max]

7.4 The methodology for Arrow Key Controls: was discussed in detail in point 6.4 of Methodology from 6.

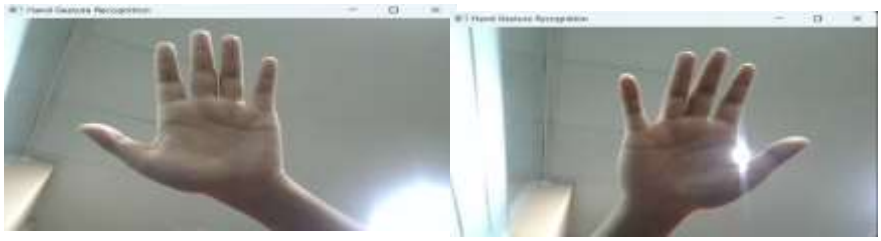

Fig. 9 – (a) Left Arrow [Show left hand]; (b) Right Arrow [Show right hand]

7.5 The methodology for Pattern Recognition: was discussed in detail in point 6.5 of Methodology from 6.

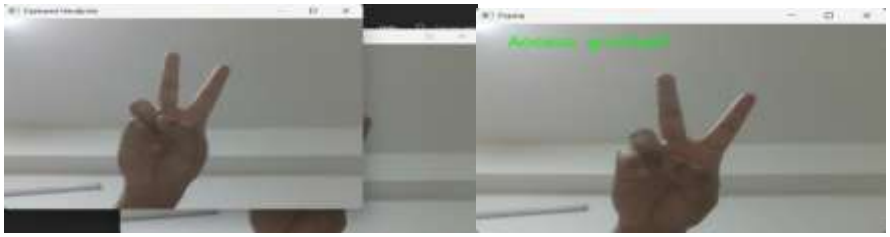

Fig. 10 – (a) Capturing hand pattern for recognition [Press "c" to capture]; (b) Hand-pattern Recognized [Access Granted]

7.6 The methodology for Zoom-In and Zoom-Out: was discussed in detail in point 6.6 of Methodology from 6.

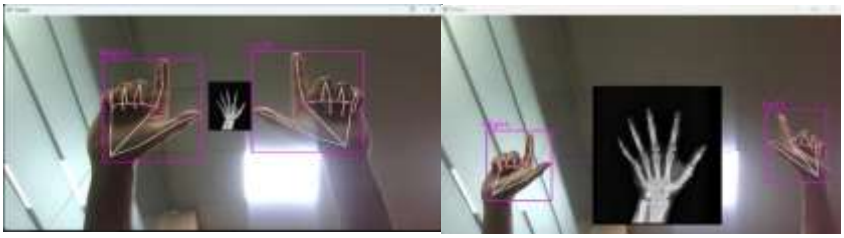

Fig. 11 – (a) Zoom-Out [Hand landmark 5 - 5]; (b) Zoom-In

8. Conclusion

In conclusion, this project showcase a diverse range of functionalities implemented through innovative methodologies and robust programming techniques from controlling volume and mouse movements using hand gestures to adjusting brightness levels and implementing biometric authentication, demonstrating their versatility and power of computer vision and ML technologies. Throughout the development process, careful consideration was given to importing necessary libraries, initializing functionalities, setting up webcams, detecting hand gestures, analyzing gestures, and providing visual feedback to users. These steps were crucial in ensuring the seamless integration of various components and the effective functioning of the projects in real-world scenarios. Moreover, the projects exhibit a user-centric approach by prioritizing ease of use and intuitive interaction. By providing visual feedback and implementing user-friendly interfaces, users can easily understand and interact with the systems, enhancing their overall experience. Moving forward, these projects serve as valuable foundations for further exploration and development in the fields of human-computer interaction, biometric security, and assistive technologies. As technology continues to evolve, this project offers a glimpse into the possibilities of leveraging computer vision and ML to create innovative solutions that improve everyday tasks and enhance user experiences. The project underscores the potential for interdisciplinary collaboration, drawing upon concepts from computer vision, ML, human-computer interaction, and user experience design. By combining expertise from multiple domains, these projects were able to deliver holistic solutions that address complex challenges while remaining accessible to a wide range of users. As technology continues to advance and these projects serve as inspiration, it is essential to consider the ethical implications and societal impact of deploying such systems. Ensuring privacy, fairness, and inclusivity should be integral parts of the development process, fostering trust and acceptance among users.

Acknowledgements

Grateful for the Almighty, loving Parents and other family members, our most respectable Management, Principal, Directors, Professors, Administrating and Supporting faculty members and all our student friends who helped us directly or indirectly in this MCA research.

References

Saurabh Adhikari, Tushar Kanti Gangopadhayay, Souvik Pal, Akila, D. ,Mamoona Humayun, Majed Alfayad & Jhanjhi, N. Z. (2023). Novel Machine Learning–Based Hand Gesture Recognition Using HCI on IoT Assisted Cloud Platform. *Computer Systems Science and Engineering, 46, 2,* 2123-2140.

Djosic, S., Stojanovic, I., Jovanovic, M., Nikolic, T., & Djordjevic, G. L. (2021). Fingerprinting-assisted UWB-based localization technique for complex indoor environments. *Expert Systems with Applications, 167, 1,* 1–14.

Martendra Pratap Singh, Arzoo Poswal, & Eshu Yadav, (2022). Volume Control Using Gestures. *International Journal of Innovative Science and Research Technology, 7, 5,* 203-206.

Google Mediapipe, www.developers.google.com/mediapipe/solutions/guide.

Hamid A. Jalab, (2012). Static Hand Gesture Recognition for Human Computer Interaction. *Information Technology Journal, 11,* 1265-1271.

Cristina Manresa-Yee, Javier Varona, Ramon Mas, & Francisco J. Perales. (2000). Real –Time Hand Tracking and Gesture Recognition forHuman-Computer Interaction. *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, 1-7.

Intel Corp. OpenCV Wiki. OpenCV Library.

Zhang, Z. , Wu, Y. , Shan, Y. & Shafer. S. (2001). Visual Panel: Virtual Mouse, Keyboard and 3D Controller with an Ordinary Piece of Paper. In Proceedings of Perceptual User Interfaces. Orlando, FL USA.

Nobuyuki Otsu. (1979). A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics. 9, 1,* 62-66.

Murthy, G. R. S., & Jadon, R. S. (2009). A Review of Vision Based Hand Gestures Recognition. *International Journal of Information Technology and Knowledge Management. 2, 2,* 405-410.

Mokhtar M. Hasan, & Pramoud K. Misra. (2011). Brightness Factor Matching For Gesture Recognition System Using Scaled Normalization. *International Journal of Computer Science and Information Technology, 3, 2,* 35-46.

Asaari, M.S.M., Rosdi, B.A., & Suandi, S.A. (2014). Intelligent biometric group hand tracking (IBGHT) database for visual hand tracking research and development. *Multimed. Tools Appl*., 70, 1869–1898.

Quam, D.L., (2002). Gesture recognition with a DataGlove, IEEE Conference on Aerospace and Electronics. Dayton, OH, USA.

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. Dalian, China.

Ashwini M. Patil, Sneha U. Dudhane, Monika B. Gandhi, Nilesh J. Uke. Cursor system using hand gesture recognition. International Journal of Advanced Research in Computer and Communication Engineering, 2, 5, 1-4.

Sziladi, G., Ujbanyi, T., Katona, J., Kovari, A. (2017). The Analysis of Hand Gesture Based Cursor Position Control during Solve an IT Related Task. In Proceedings of the 2017 8th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), Debrecen, Hungary.

Gonzalez, R. C., (2016). Digital image processing (Prentice hall)

Gowsikraja P, Thevakumaresh T, Raveena M, Santhiya.J, Vaishali.A.R.R, (2022).Object Detection Using Haar Cascade Machine Learning Algorithm, International Journal of Creative Research Thoughts, 10, 6, 742-745.

Zhuang, H., Xia,Y., Wang N. & Dong, L. (2021). High inclusiveness and accuracy motion blur real-time gesture recognition based on YOLOv4 model combined attention mechanism and DeblurGanv2. Applied Sciences, 11, 21, 1–19.

Viola, P. & Jones M. (2001). Rapid object detection using a boosted cascade of simple features. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognition, 1, Kauai, HI, USA.

Nahar, M. & Ali, M. S. (2014). An Improved Approach for Digital Image Edge Detection. Int. J. Recent Dev. Eng. Technology, 2, 3, 14-20.

Ahmad Puad Ismail, Farah Athirah Abd Aziz, Nazirah Mohamat Kasim, & Kamarulazhar Daud. (2021). Hand gesture recognition on python and opencv. 2021 IOP Conf. Ser.: Mater. Sci. Eng. Sanya, Hainan Province, China.

Mukherjee, S., Ahmed, S., Dogra, D., Kar, S., & Roy, P. (2019) Fingertip detection and tracking for recognition of air-writing in videos. Expert Syst. Appl. 136, 217–229.

Joshua R New, Erion Hasanbelliu, & Mario Aguilar. (2003). Facilitating user interaction with complex systems via hand gesture recognition. In the Proceedings of the 2003 Southeastern ACM Conference, Savannah, GA.

Nayana, P. B. & Kubakaddi, S. (2014). Implentation of Hand Gesture Recognition Technique for HCI Using OpenCV. International Journal of Recent Dev, 2,5, 17-21.

Lin, T., Doll´ar, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 2117–2125.

Katona, J. (2021). A review of human–computer interaction and video game research fields in cognitive Info Communications. Applied Sciences, 11, 6, 2646..

Sziladi, G., Ujbanyi, T., Katona, J., & Kovari, A. (2017). The Analysis of Hand Gesture Based Cursor Position Control during Solve an IT Related Task. In Proceedings of the 2017 8th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), Debrecen, Hungary