Hand gesture recognition for Human-computer interaction using computer vision

Kavin Chandar Arthanari Eswaran[1], Akshat Prakash Srivastava[2] and Gayathri M[3]

[1]SRM Institute of Science and Technology, SRM Nagar, Kattankulathur-603203, India. E-mail: ka7290@srmist.edu.in,
[2]SRM Institute of Science and Technology, SRM Nagar, Kattankulathur-603203, India.E-mail: as9075@srmist.edu.in,
[3]SRM Institute of Science and Technology, SRM Nagar, Kattankulathur-603203, India. E-mail: gayathrm2@srmist.edu.in

Abstract: We use gestures to communicate with friends, family, and colleagues every day. Gestures have always been a natural and intuitive form of interaction and communication, from waving our hands across the hallway to signal someone to keep the elevator doors open or greet friends from afar. Gestures are the universally understood extension of our body language and comprise a core part of everyday interaction. When we consider this idea for vision-based interaction with computers, we get hand gesture recognition for human-computer interaction using computer vision. We perform a predefined set of gestures in front of a camera and assign individual actions to them. Hence, a computer can recognize the said gestures and complete the appropriate actions. We use this exact method to develop a way to interact with computers with little to no contact. The advent of computer vision and machine learning data libraries makes this dream of contactless interactive technology possible. In light of the pandemic, We wish to create an AI virtual computer mouse. We recognize predefined gestures with a camera and assign appropriate actions that a mouse commonly performs to them. Since the limitations of a physical mouse no longer bind us, we can redefine the conventions of operating a computer with a new set of convenient and easy to grasp rules. We are starting this by developing a virtual gesture-based volume controller. We can perform these feats due to the marvelous works done by academics and researchers that have come before us. The main goal is to use the advantages of computer vision and machine learning to combat the pandemic’s challenges and push forward the future of interactive technology. Thus, we get Human-computer interaction using hand recognition technology.

Keywords: Computer vision; Human-computer interaction; Hand gesture recognition; Contactless interactive technology; vision-based techniques

1. Introduction

For the longest time, Contactless human-computer interaction has been at the forefront of making life more accessible and convenient for all. From voice-controlled automotive systems on cars and gesture-based entertainment systems in the living room. Video Games, Virtual Reality Simulations, Voice assistants, etc. These are a few of the many applications built for comfort and entertainment. However, there are plenty of scenarios where contactless human-computer interaction can be the essential and more optimal approach relative to current methods, in the case of factory workers or medical professionals whose hands are preoccupied. Industry Professionals can immensely benefit from a system that can be purely operated with little to no contact. In light of the covid-19 pandemic and living in the post-pandemic world, It has become a necessity for contactless human-computer interaction systems to carry the burden of slowing transmission rates moving forward. Therefore, We aim to contribute to this cause of developing a low cost and highly accessible system that could be used for all. This deal that sounds too good to be true is now a reality due to the significant leaps in computer vision and machine learning fields.

1.1 Hand Tracking Module

The first step was to design a hand tracking module to recognize hand gestures accurately. This would be the foundation for other applications going forward. We can achieve this by using the computer vision and machine learning libraries in python to get optimal results. We also aim to learn and increase hand detection confidence to get even more accurate results.

1.2 AI Virtual Mouse

The best place to start was to develop a virtual mouse that was already a multifunctional device in terms of operating a computer system. Thus, we can create the bare minimum functionality needed to manage a computer system fully. Using the Hand tracking module mentioned above, we can accurately track the positions of the 21 points on an average human hand. We can then use this data to assign standard mouse functions such as cursor movement and click automation to our hand gestures. Thus, we can now fully simulate a physical working mouse using our hands and a camera to operate a computer system adeptly.

1.3 Gesture-based Volume Controller

Now that we have a hand tracking module, the physical limitations of a mouse no longer bind us. We can simulate a volume slider using the index finger and thumb finger position data since this uses the same data already acquired for the virtual mouse. There is no extra cost in obtaining data. Furthermore, Since the physical limitations of a mouse no longer bind us. We can venture into the realm of discovering new rules to replace the old conventions of operating a mouse. We can look forward to many such innovations in the future.

1.4 Software Requirements

Python is the programming language we use for building our appliances. **OpenCV** is often used as a python library of methods primarily for real-time computer vision and an exquisite tool for image processing. It's an open-source library that may perform face detection, objection tracking, landmark detection, and much more. It supports multiple languages, including python, java, C++.

On the other hand, **Mediapipe** is popularly used as an ML solutions library for all major platforms developed by Google that provide out-of-the-box solutions for computer vision tasks, etc. **win32api** is employed for assigning cursor movement. It's a library of python extensions for windows that allows the user to use the Win32 application programming interface features.

PyAutoGui is employed primarily for click automation purposes. It’s a Python automation library accustomed to clicking, dragging, scrolling, moving, etc. **Pycaw** is utilized for acquiring admin access to the quantity slider in windows. It's a python module developed by Stanford students used for Python Core Audio Windows Library.

We use those modules together to form a **computer-vision-based hand gesture recognition module** to simulate human-computer interaction at a coffee cost and in a highly accessible fashion.

2. Literature Survey

The literature in this field comprises three core aspects- Hand gesture recognition, Human-computer interaction, and computer vision.

[1,4] - We see researchers use algorithms to identify particular gestures. To name a few, a novel finger skin pixel algorithm which they claim has a hand recognition rate of 94.3%. The K-nearest neighbor algorithm is also used for recognition based on Kanade's Pyramidal optical flow algorithm. There are also mentions of a full-fledged multi-participant visual environment (VE), Which is essentially a 3D environment to move objects with only our hands.

[5,6] - We see references to multimodal human-computer interaction research for facial expression recognition(MMHCI). We also see methods to improve the precision of gestures by factoring user fatigue into account. There are even mentions of autonomous gesture commands.

[7,8] - Here, we see advances in using perceptive user interfaces for head tracking, facial expression recognition, eye tracking, and other forms of gestures. There are also models and algorithms for detecting sign language at a recognition rate of 92% and in 2.76s. They also claim that there is no need for training sample data for their algorithm to provide said results.

[9,10]- These papers refer to using Microsoft’s Kinect sensors to accurately recognize hand insensitive variations and other distortions using RGB and depth data.

[11,16]- The researchers here look for methods to move on from standard mouse and keyboard setups to other forms of gesture-based systems to push forward means of human-computer interaction. Computer vision-based gesture recognition is done for Augmented reality Interfaces. They use algorithms for detection, tracking, and recognition to build 3D hand models for more accurate results. This particular survey examines 37 papers on depth-based gesture recognition and provides comparisons on optimal methods for optimal use cases. Thus, We have an overview of similar research work done in hand gesture recognition, human-computer interaction, and computer vision.

[17,19] - Here, we discover fields such as Fuzzy systems towards human-centric computing which showcase user-friendly nature of systems with a tradeoff between accuracy and transparency. Next up, we see QA2MN, a question-aware memory network to update the attention on questions during the reasoning process. This helps in performing intelligent human–robot interactions. Finally, we observe an interesting study that aims to learn how colors affect a person’s product preferences by using an eye-tracking device intended to identify the role of their eye movements.

3. System Architecture and Design

3.1 Architecture Block Diagram

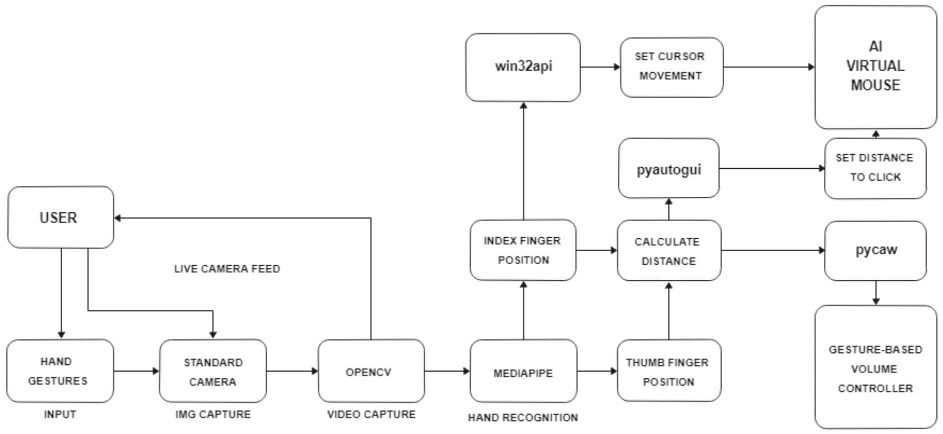


Fig. 1 Block Diagram

The above block diagram clearly illustrates the workings of the modules in tandem to make the AI Virtual Mouse and Gesture-based volume controller a reality

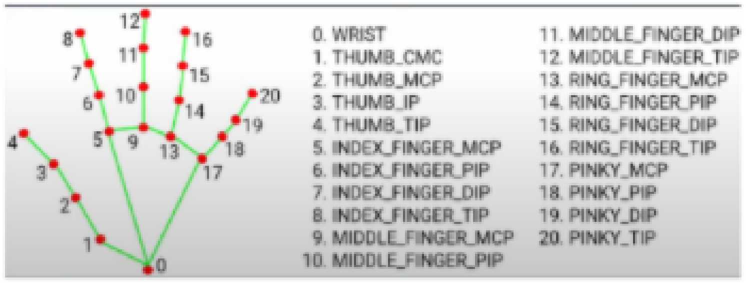


Fig. 2 Hand Landmarks

The 21 Hand landmarks displayed above are crucial in deploying the aforementioned architecture diagram into a functioning prototype. They serve as the rope tying all these modules together to function as one.

.3.2 Design of Modules

OpenCV module is a computer vision library that helps capture and display output. (pip install OpenCV). **Mediapipe** is an open library of ML solutions to recognize hand landmarks. **win32api** is used for cursor movement. (pip install mediapipe). **PyAutoGUI** allows us to use python scripts to control the mouse and keyboard to automate interactions with other applications. (pip install pyautogui). We calculate the distance between the index and the thumb finger in real-time. This distance is used for click automation. **Pycaw** helps us access the windows volume slider and uses the same distance range as a volume slider. **For the Ai Virtual Mouse,** We use the OpenCV module and media pipes open dataset to recognize hands and display the same using OpenCV's custom methods. win32api is used for cursor movement and click automation.

PyAutoGUI allows us to use python scripts to control the mouse and keyboard to automate interactions with other applications. The API is designed to be simple. We use a hand tracking module for the Gesture-based volume controller to calculate the distance between the index and the thumb finger in real-time using OpenCV and mediapipe. Pycaw helps us access the windows volume slider.

4. Methodology

4.1 Hand Tracking Module

The Hand tracking module is developed to provide a foundation for all our upcoming vision-based human-computer interaction projects that rely on hand recognition. It helps us lay the bricks ahead of time since it's made reusable and versatile for various purposes in mind.We use the time module to calculate the full frames per second. On the other hand, cv2 is the OpenCV library. We use it to capture the live camera feed and feed the image output into the mediapipe. Mediapipe is an ML solutions module that allows users to recognize the hands of an image with its highly trained datasets. Next, we implement a class for hand detection recognition. It serves with necessary and respective methods needed for implementing future projects. The default constructor here takes mode, max_Hands, Detection confidence, Tracking confidence into account. These variables are initialized with values that help shape the overall project. For example, First, the max_Hands variable determines the total number of hands recognized on the camera. It allows us to restrict the number of recognized hands based on use-case. Secondly, The detection and tracking confidence help us determine the recognized object to be the intended object. This base constructor serves the critical purpose of setting the base conditions necessary for future endeavors. The first method we have defined is the Find hands method which returns the respective frame from a live camera feed if the module recognizes a hand. We first convert the input image from BGR to RGB. Since mediapipe doesn’t directly recognize BGR images. Then the image is processed for the hand landmarks. The method draws connections to visualize the identified points if landmarks are present and the draw condition is set to True. We then return the updated image in real-time. The second method we have defined is the Find position method which returns a list of positions of all the 21 hand landmarks identified at any given moment. We iterate through the self.results gained from the previous method. The handNo parameter helps us choose a particular hand out of all recognized hands. The height and width of the image are then multiplied with the respective x and y coordinates of each hand landmark. Compiling this information into a list of positions indexed with their respective landmark indices. This constantly updating list of positional landmark coordinates helps us recognize the exact position of the recognized hand at any moment. This data is beneficial in identifying gestures and assigning actions to them in real-time.

4.2 AI Virtual Mouse

The AI virtual mouse we implement here is synonymous with a regular mouse commonly used to operate a computer. The goal is to create a version that works with little to no contact required from the user. To achieve the said goal above, we import the aforementioned modules. Noteworthily, we import the hand tracking module we developed beforehand to serve the function of hand recognition in this project. We introduce two new modules, win32api and pyautogui. These are vital for this model to simulate a working mouse on any windows machine fully. We use the object ‘detector,’ an instance of the hand_detector class. We can access the previously defined methods find_Hands and find_Position. These methods aid us in hand gesture recognition by detecting hand-like features from our camera feed and returning the positions of the 21 hand landmarks relative to the image shape. By checking if poslist exists, we can determine if our hand tracking module recognizes hands at any moment to attain the following data. We only require positional data of the index_tip, thumb_tip, middle_finger_pip, and middle_finger_dip. Using this, we can bring life to the AI virtual mouse and gesture-based volume controller going forward. We check if the x and y coordinates of the index_tip exist given the camera. If detected, we then use win32api's SetCursorPos to set the cursor position relative to the frame of the computer specifications. Since this condition is placed inside a while loop that runs until our camera footage dies, We can constantly update the cursor position in real-time by moving our index_tip part given the camera. Using this method, we can fully simulate the cursor moment of a regular mouse using only a camera and the tip of our index finger. Another neat feature is implementing inverted controls for our virtual mouse by flipping the camera orientation into its mirror counterpart. This will prove beneficial to people accustomed to inverted controls on their devices. Now that we can traverse the computer screen with the cursor freely. We set out to implement the other essential functions that define the standard computer mouse—starting with the most crucial button for interaction. We designed this by keeping in mind the needs and wants of the user. It is created comfortably for the user to left click without any obstructions. The user is meant to adopt a particular hand position to traverse the screen freely. We then use the positional data of the thumb_tip and middle_finger_pip to calculate the euclidean distance between them at any given moment. Now we can use the distance calculated to create a switch. So when we bring our thumb_tip closer to the middle_finger_pip landmark in front of the camera. The euclidean distance between them is being calculated at every passing moment. We check if the space drops below a certain threshold to detect a user's intent to click. Hence when we notice the user's intent, we use pyautogui.click() to automate the left click functionality at the user's beck and call. However, pyautogui.click() performs multiple clicks in the span of a short moment due to the nature of the video camera footage. Therefore, we employ a method to slow down the detection process by checking if the euclidean distance is a multiple of five under the threshold distance of 20. Hence only four out of twenty instances are noticed in a single moment. This gives the user more agency and control over this particular action. Now, To implement the other standard function of the mouse. It helps us access many options depending on the cursor's position and application. To go following the position the user adopts. We employ the same euclidean distance-based switch. We implement it between the thumb_tip and middle_finger_dip. Now by examining this distance relative to a threshold. We can use pyautogui.click(button='right') to provide the right-click functionality when the euclidean distance measure detects the user's intent to click. Same as before to reduce the rate of clicks and keep it similar in feel to the left click automation. We check the euclidean distance values that break the threshold to see if they are multiples of five. This action happens many times in the span of a single moment. Hence, This allows the user more agency and control over this right-click automation.

4.3 Gesture-based Volume Controller

We have a functioning vision-based virtual mouse that performs the expected functionalities of a regular mouse. We move on to test the potential of vision-based systems. In doing so, we hope to establish new and improved conventions to overwrite the old and outdated ones. We start by developing a gesture-based volume controller. We chose this because it doesn't require any additional data and can perform well from the data we accumulated for the virtual mouse.The hand-tracking module we established is used here, proving its versatile and reusable nature. Numpy is used here to relate the distance measure with other ranges. We introduce an all-new module here called pycaw. Pycaw is a module developed by Stanford students to gain admin access to the window's audio libraries. By combining them with our code, we can realistically conceive the dream of operating a system without any physical contact. The volume controller function encompasses the whole process. First, we define all the necessary attributes, such as the width and height of the camera used. Then we use cv2.Videocapture(0) to acquire the live camera feed. We also set the image specifications. prev_time is a variable defined here, which will aid us later in calculating the overall frames per second. We also call a detector, an object of the hand tracking module. This allows us to contact the default constructor of the hand_Detector() class. We initialize several member variables required for hand recognition later. Then we use GetSpeakers(), a method of AudioUtilities from the pycaw module. This method acquires all the active audio devices connected to the machine and stores that information in the variable devices. Interface and cast() methods check for the recognized devices' current audio levels and assign those values to the volume variable. We use GetVolumeRange() to get the volume range of an audio device. Then we define the minimum volume and maximum volume and align them to their respective variables and VolRange indices. These stored values gained from the pycaw module will be used to specify the parameters needed for feature-based volume control. We begin our live camera feed by using capture.read() we acquire the frame to be analyzed for one iteration. We pass this frame as input to the hand tracking module methods find_Hands and find_Position. They return an updated picture with visualized hand connections and a list of positions of all the 21 hand landmarks recorded by the module. We only require the positional coordinates of the thumb_tip and index_tip for this particular module. So we store those distinct values alone in variables such as thumb_x, thumb_y, index_x, and index_y. These values are crucial to determining the distance measure needed to detect the user's intent to change the volume. We then calculate the midpoint between the thumb_tip and index_tip. We visualize the distance measure between the thumb_tip and index_tip using cv2.line() and the midpoint using cv2.circle(). We calculate the euclidean distance between the thumb_tip and index_tip using math.hypot(). We can now use this length measure against a threshold of value 20 to detect the user's intent through the gesture of bringing the thumb and index finger closer together. When we notice such an intention, we use cv2.circle() again to change the midpoint color, visualizing the lower bound. Hence, This action loosely translates to the workings of a switch. We used this already for left click and right click automation in the last project. However, in this context, it acts as a mute switch to set the volume to 0. Now that we have established our lower bound, we use the NumPy module's interp() method to relate the length measure between the thumb_tip and index_tip to the volume range between the predefined minimum volume and maximum volume of a recognized audio device. We can then use the volume.SetMasterVolumeLevel() method from pycaw to update the master volume level concerning the changes in the length measure. VolBar and VolPer variables are calculated to simulate the audio changes. We then pass these to the volume bar() function to visualize the process in real-time. The volume bar utility function here displays the changes in volume and the live camera feed in real-time. By combining methods from openCV and values calculated using NumPy, we can show the correlation between the distance measured between the index and thumb and the volume slider in windows.

4.4 Utility Functions

We write a few functions ahead of time to maintain the readability and cleanliness of our code. These functions tend to require use more than once. So we establish these as utility functions to prevent redefining them from scratch every time. The fps function uses the time() method from the time python module to acquire the current time at any given moment. We then perform the above-shown fps calculation and visualize them using the putText() method from the cv2 module. We update the prev_Time variable at every function call instance to ensure the calculation remains up to date with the live camera feed. We use the display function above to view the recognized 21 hand landmarks and their connections. This function is essential as it showcases the individual working of our heavily customized hand recognition module. We use the fps() utility function here to calculate and display the total frames per second calculated at any given moment in the live camera feed. Here it serves to show the efficiency of the Hand Tracking module on a particular device. This method provided by OpenCV allows us to view our ever updating image in unison with the live camera feed. This allows us to observe the fruits of our labor instantly. This method exists within the while loop and constantly updates with every input. We use matplotlib.pyplot to plot the efficiency of our overall code in a devised graph format with time measured in seconds on the x-axis and fps on the y-axis. We also use it to test and plot the efficiency of our overall code in a colored bar graph format with time measured in minutes on the y -axis and fps on the x-axis. Then, the cv2.waitkey(10) and 0xFF = ord('x') commands use these as exit conditions to break the while loop thereby, breaking the whole live camera feed. So the user can press the key 'X' on his keyboard to terminate the program when it is running.

5. Output

5.1 Hand Tracking module

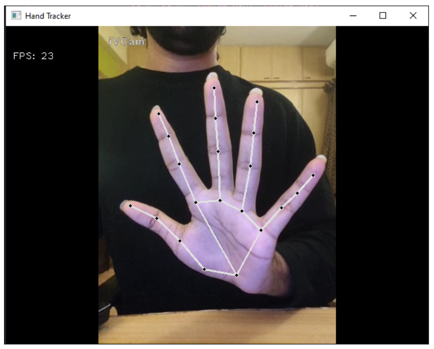


Fig. 3 Front view

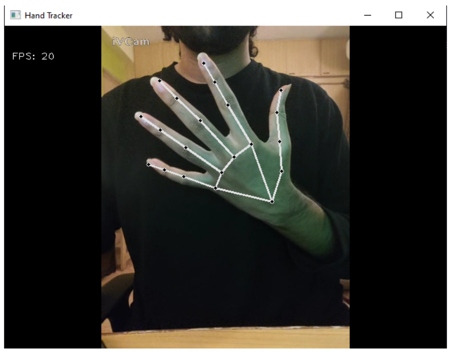


Fig. 4 Back view

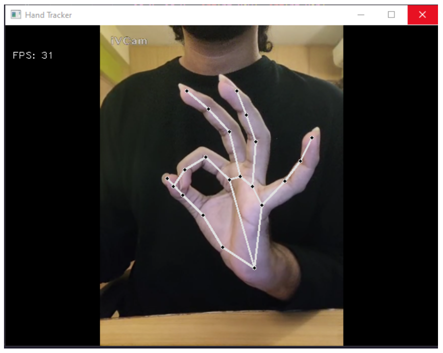
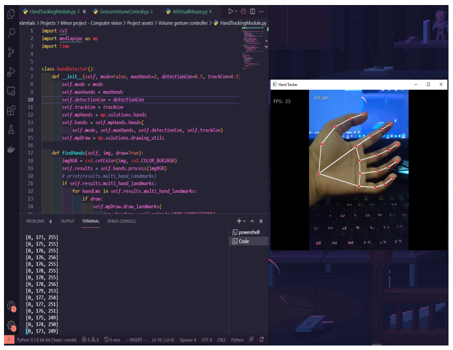


Fig. 5 Random gesture


HandTrackingModule.py 2
GestureVolumeControl.py 2
AiVirtualMouse.py 4
edentials > Projects > Minor project - Computer vision > Project assets > Volume gesture controller > HandTrackingModule.py
PROBLEMS 8
OUTPUT
TERMINAL
DEBUG CONSOLE
powershell
Code
Python 3.7.6 64-bit ('base': conda)
Ln 10, Col 41
Spaces: 4
UTF-8
CRLF
Python
Hand Tracker
FPS: 33

```
import cv2
import mediapipe as mp
import time


class handDetector():
    def __init__(self, mode=False, maxHands=2, detectionCon=0.5, trackCon=0.5)
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.trackCon = trackCon
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(
            self.mode, self.maxHands, self.detectionCon, self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils

    def findHands(self, img, draw=True):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)
        # print(results.multi_hand_landmarks)
        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(
```

```
[0, 173, 255]
[0, 175, 255]
[0, 176, 255]
[0, 176, 256]
[0, 176, 255]
[0, 178, 255]
[0, 178, 255]
[0, 178, 256]
[0, 179, 253]
[0, 177, 254]
[0, 177, 251]
[0, 176, 251]
[0, 175, 249]
[0, 174, 250]
[0, 173, 249]
```

Fig. 6 Position data

5.2 AI Virtual Mouse

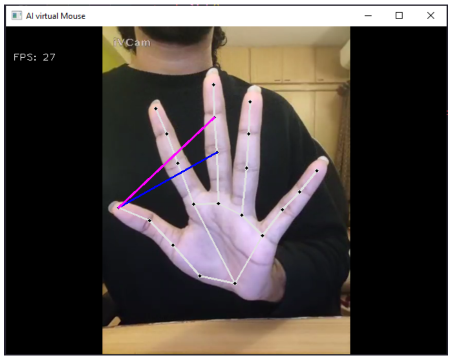


Fig. 7 Base virtual mouse

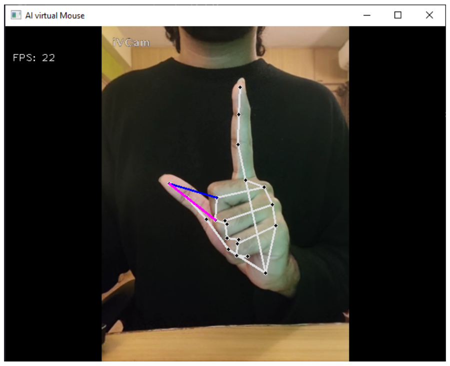
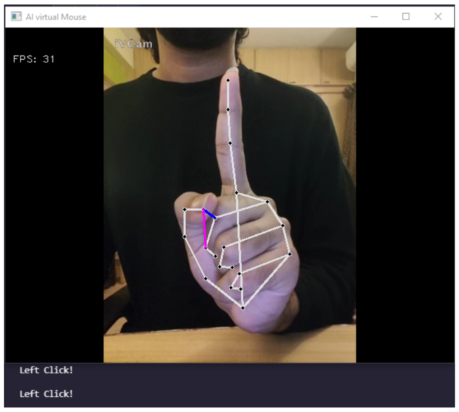
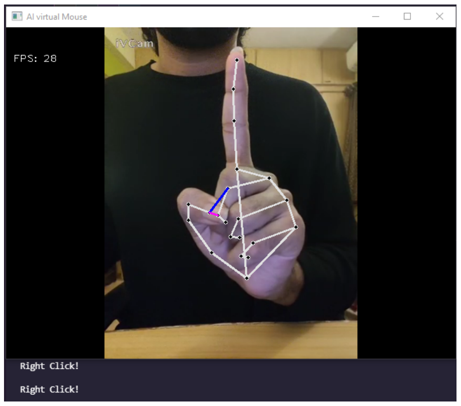


Fig. 8 Cursor gesture



```
Left Click!
Left Click!
```

Fig. 9 Left-click gesture



```
Right Click!
Right Click!
```

Fig. 10 Right-click gesture

5.3 Gesture-based Volume Controller

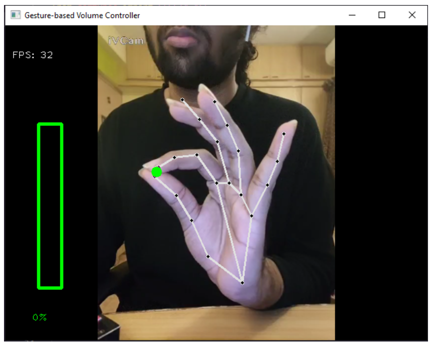


Fig. 11 Volume at 0%

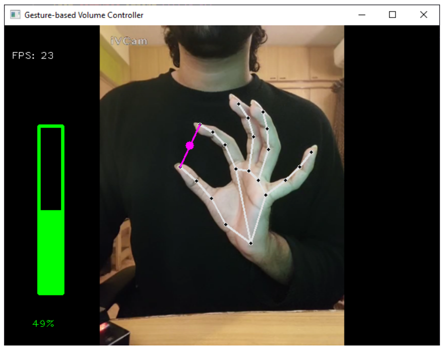


Fig. 12 Volume at 50%

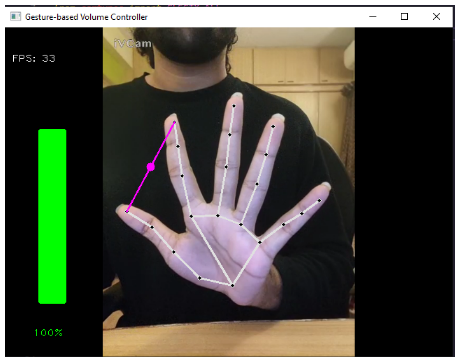


Fig. 13 Volume at 100%

6. Results and Discussions

Our algorithm amalgamates triumphs in computer vision, machine learning, hand-recognition technology, and human-computer interaction. It works with all the necessary advanced technological models developed in said fields. We measured the total frames per second to judge the efficiency of our applications.

Hand Tracker module

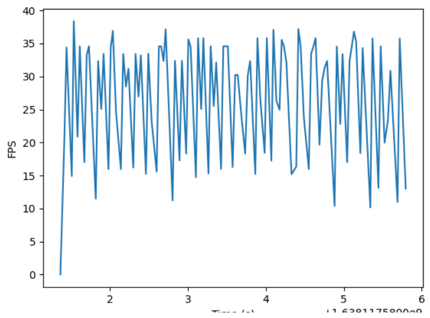


Fig. 14 Fps drop rate graph

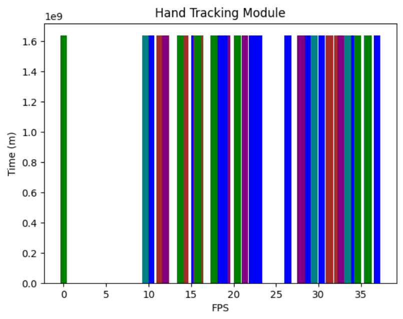


Fig. 15 Fps consistency graph

AI Virtual Mouse

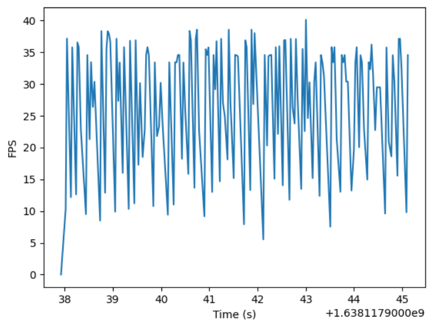


Fig. 16 Fps drop rate graph

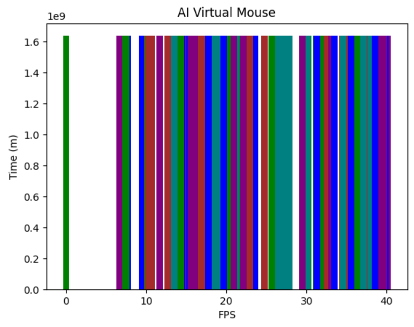


Fig. 17 Fps consistency graph

Gesture-based Volume Controller

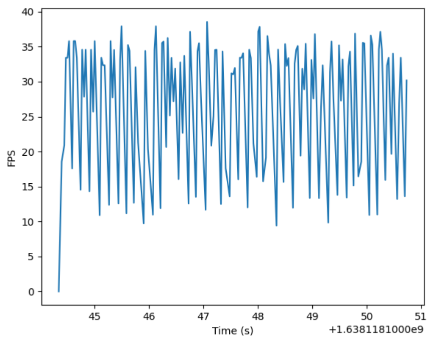


Fig. 18 Fps drop rate graph

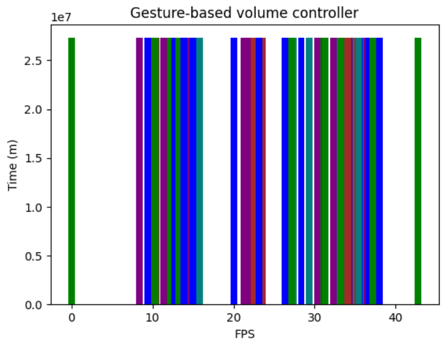


Fig. 19 Fps consistency graph

S.No	**Modules**	**Average Fps**
1.	Hand Tracking Module	26.0 Frames per second
2.	AI Virtual Mouse	25.0 Frames per second
3.	Gesture-based Volume Controller	26.0 Frames per second

Fig. 20 Synopsis of proposed models

The frames per second in each module were more or less identical due to using mediapipe advanced models. However, The frames per second were observed to drop depending on low lighting, distorted camera vision, too much movement, etc. These actions caused the frames per second to fluctuate instead of the expected steady 30 frames per second. However, These flaws can be overcome by installing a better graphics processing unit (GPU) in the computer.

7. Challenges

We are raising Hand Recognition Confidence using any standard camera on any traditional system. To make this widely accessible, we aim to create a model that works best for all available current devices without needing much investment from the user in terms of camera specifications. We are looking for ways to improve consistency in the detection of gestures. Currently, there are inconsistencies due to low light settings and other distractions that occur in the camera frame. We are looking for methods to improve this as much as possible while keeping the cost as low as possible. We are assigning an action to gestures. Administrative access to certain functionalities is not readily available at a coffee cost. This has proven to be quite a challenge to overcome on our own. We wish to develop a dynamic Software system to work on any compatible system with a camera and computer. Currently, Our model only works on PC machines with windows operating systems. We aim to expand this forward to all sorts of devices. We are improving the user interface and experience. We only have working prototypes, but we hope to develop full-fledged applications convenient to use shortly. We are improving accuracy and quality while maintaining feasibility and accessibility. We aim to optimize as much as we can to keep costs low without compromising on the integrity of the application. Thus, We have successfully implemented a hand tracking module to create a functioning model of an AI virtual mouse and gesture-based volume controller using computer vision and machine learning solutions. This is the first of many steps yet to be taken to use hand recognition technology using computer vision commercially for human-computer interaction at a low cost and highly accessible fashion.

8. Conclusion and Future Enhancement

With the arrival of new and exciting technologies such as computer vision, interactive intelligence, and machine learning, We can create advanced human-computer interaction models due to the advances made in such fields. Our algorithm uses these advances to detect the needed positions, such as the index and the thumb finger. Using these positions, we can calculate the Manhattan distance between them. This relative distance is used for click automation and as a volume slider when appropriate. We also use the index finger position when in the frame and assign it to cursor movement. These two simple actions assigned to advanced miracle models in hand recognition and computer vision give an AI mouse with a gesture-based volume controller. In the future, This technology can be custom-tailored to be of use in a wide variety of areas such as healthcare solutions, consumer electronics, security and entertainment systems, etc. Models then gesture recognized through cameras can potentially control all devices sign or even give birth to fully virtually controlled environments. Predictions show that the market for gesture recognition technologies is growing, and the sky's the limit**.** We evolved from analog-controlled devices to touch-controlled devices. Eventually, the future points to contactless interactive technology through voice and gestures, to name a few. For example, Kinect, developed by Microsoft, was initially intended to track whole-body movements. KinTrans Hands Can Talk is a project that uses AI to learn and process sign the body movements of sign language. GestSure allows surgeons to navigate through MRI and CT scans without touching a screen. Audi and BMW have already implemented a system that will enable drivers to use gestures to control the infotainment system inside the car. There are also numerous open-source projects for hand gesture recognition, like Real-time GesRec based on PyTorch. The Future is immensely bright going forward with Gesture-based technology.

9. References

1. Rios-Soria, D. J., Schaeffer, S. E., & Garza-Villarreal, S. E. (2013). Hand-gesture recognition using computer-vision techniques.
2. Lai, H. Y., Ke, H. Y., & Hsu, Y. C. (2018). Real-time Hand Gesture Recognition System and Application. *Sens. Mater*, *30*, 869-884.
3. Garg, P., Aggarwal, N., & Sofat, S. (2009). Vision-based hand gesture recognition. *World Academy of science, engineering, and technology*, *49*(1), 972-977.
4. Rautaray, S. S., & Agrawal, A. (2015). Vision-based hand gesture recognition for human-computer interaction: a survey. *Artificial intelligence review*, *43*(1), 1-54.
5. Jaimes, A., & Sebe, N. (2007). Multimodal human-computer interaction: A survey. *Computer vision and image understanding*, *108*(1-2), 116-134.
6. Lenman, S., Bretzner, L., & Thuresson, B. (2002). Computer vision-based hand gesture interfaces for human-computer interaction. *Royal Institute of Technology, Sweden*.
7. Zabulis, X., Baltzakis, H., & Argyros, A. A. (2009). Vision-Based Hand Gesture Recognition for Human-Computer Interaction. *The universal access handbook*, *34*, 30.
8. Panwar, M., & Mehra, P. S. (2011, November). Hand gesture recognition for human-computer interaction. In *2011 International Conference on Image Information Processing* (pp. 1-7). IEEE.
9. Ren, Z., Yuan, J., Meng, J., & Zhang, Z. (2013). Robust part-based hand gesture recognition using Kinect sensor. *IEEE transactions on multimedia*, *15*(5), 1110-1120.
10. Tang, M. (2011). Recognizing hand gestures with Microsoft's Kinect. *Palo Alto: Department of Electrical Engineering of Stanford University:[sn]*.
11. 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.
12. Störring, M., Moeslund, T. B., Liu, Y., & Granum, E. (2004, September). Computer vision-based gesture recognition for an augmented reality interface. The 4th *IASTED international conference on visualization, imaging, and image processing* (Vol. 766, p. 771).
13. Pavlovic, V. I., Sharma, R., & Huang, T. S. (1997). Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on pattern analysis and machine intelligence*, *19*(7), 677-695.
14. Davis, J., & Shah, M. (1994, May). Recognizing hand gestures. In the European *Conference on Computer Vision* (pp. 331-340). Springer, Berlin, Heidelberg.
15. Rautaray, S. S., & Agrawal, A. (2010, December). A novel human-computer interface based on hand gesture recognition using computer vision techniques. In *Proceedings of the First International Conference on Intelligent Interactive Technologies and Multimedia* (pp. 292-296)
16. Wu, Y., Lin, J. Y., & Huang, T. S. (2001, July). Capturing natural hand articulation. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001* (Vol. 2, pp. 426-432). IEEE.
17. Pedrycz, W. and Gomide, F., 2007. *Fuzzy systems engineering: toward human-centric computing*. John Wiley & Sons.
18. X. Li, M. Alazab, Q. Li, K. Yu, Q. Yin, "Question-aware memory network for multi-hop question answering in human–robot interaction", *Complex & Intelligent Systems*, 2021
19. B. Wu, Y. Zhu, K. Yu, S. Nishimura and Q. Jin, "The effect of eye movements and cultural factors on product color selection", *Human-centric Computing and Information Sciences*, vol. 10, no. 48, pp.1-14, 2020