

REAL TIME AI VIRTUAL MOUSE SYSTEM USING DEEP LEARNING

*Report submitted to the SASTRA Deemed to be University
in partial fulfillment of the requirements
for the award of the degree of*

Bachelor of Technology

Submitted by

KUSAMPUDI HARSHA ABHINAV (Reg. No.: 12415802, CSE IOT)
NALUSANI SHIVA NANDU REDDY (Reg. No.: 124158092, CSE IOT)
BANDARU PAVAN KUMAR (Reg. No.: 124003052, CSE)

May 2024



SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)



THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

T H A N J A V U R | K U M B A K O N A M | C H E N N A I

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401



SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)



THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

T H A N J A V U R | K U M B A K O N A M | C H E N N A I

SCHOOL OF COMPUTING
THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the project report titled “**Real Time AI Virtual Mouse System Using Deep Learning**” submitted in partial fulfillment of the requirements for the award of the degree of B. Tech. Computer Science and Engineering to the SASTRA Deemed to be University, is a bona-fide record of the work done by **Mr Kusampudi Harsha Abhinav (RegNo: 124158027)**, **Mr Nalusani Shiva Nandu Reddy (RegNo: 124158092)**, **Mr Bandaru Pavan Kumar (RegNo: 124003052)** during the final semester of the academic year 2023- 24, in the School of Computing, under my supervision. This report has not formed the basis for the award of any degree, diploma, associateship, fellowship or other similar title to any candidate of any University.

Signature of Project Supervisor :

Name with Affiliation : Dr. Venkatesh V, Asst. Professor, SoC

Date : 18-04-2024

Project Based Work *Viva voce* held on _____

Examiner 1

Examiner 2



SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)



THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

T H A N J A V U R | K U M B A K O N A M | C H E N N A I

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Declaration

We declare that the project report titled “**Real Time AI Virtual Mouse System Using Deep Learning**” submitted by us is an original work done by us under the guidance of Dr. Venkatesh V, Asst. Professor, School of Computing, SASTRA Deemed to be University during the final semester of the academic year 2023-24, in the School of Computing. The work is original and wherever We have used materials from other sources, we have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associate-ship, fellowship or other similar title to any candidate of any University.

Signature of the candidate(s):

Name of the candidate(s): Kusampudi Harsha Abhinav
Nalusani Shiva Nandu Reddy
Bandaru Pavan Kumar

Date : 18-04-2023

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K. Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Algeswaran**, Associate Dean, Students Welfare.

Our guide **Dr. Venkatesh V**, Asst. Professor, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped me in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me an opportunity to showcase my skills through project.

Table of Contents

Title	Page No.
Bona-fide Certificate	i
Declaration	ii
Acknowledgements	iii
List of figures	vi
Abbreviations	vii
Abstract	viii
1. Introduction	1
1.1 What is Artificial Intelligence?	2
1.2 Applications	2
2. Objectives	4
3. Experimental Work/ Methodology	
3.1 Programming languages and tools	5
3.1.1 Python	5
3.1.2 Numpy Library	7
3.1.3 Open- CV	7
3.1.4 Mediapipe	10
3.1.5 ML – Pipeline	10
3.1.6 Autopy	12
3.2 Models	12
3.2.1 Palm Identification Model	12
3.2.2 Hand Landmark Model	13
3.3 Methodology	14

4. Results and Discussion	
4.1 Hand Gestures & their Respective Functions	16
4.1.1 AI Virtual Mouse	16
4.1.2 AI Virtual Keyboard	20
5. Conclusions & Future Scope	23
6. References	24
7. Appendix	
7.1 Sample Source Code	25

List Of Figures

Figure No.	Title	Page No
3.1	Requirements	6
3.2	Python IDLE with code Highlighting	6
3.3	OpenCV Block Diagram	8
3.4	OpenCV Demo	9
3.5	ML Pipeline	11
3.6	Hand Landmark Model	13
3.7	Methodology Flowchart	15
4.1	To Move the Cursor	16
4.2	Left Click	17
4.3	Right Click	17
4.4	Scroll Up Function	18
4.5	Scroll Down Function	18
4.6	Drag and Drop	19
4.7	Zoom in function	19
4.7	Keyboard Interface	20
4.8	Key Selection	21
4.9	Results of hand gesture of mouse control	22
4.10	Results of hand gestures of keyboard	22

Abbreviations

AI	Artificial Intelligence
AR	Augmented Reality
GUI	Graphical User Interface
HCI	Human Computer Interaction
IOT	Internet Of Things
ML	Machine Learning
OPEN CV	Opensource Computer Vision
OS	Operating System
OSI	Open Systems Interconnections
VR	Virtual Reality

Abstract

One of the advancements, in Human Computer Interaction (HCI) technology that we're all familiar with is the mouse and keyboard. Currently wireless or Bluetooth mouse and keyboard setups involve needing devices for connection and power such as a dongle for laptop connectivity and batteries for operation. The primary goal of the AI system is to create a substitute for the mouse and keyboard by capturing hand gestures and movements via a webcam. These inputs can then be processed to carry out functions like moving the cursor clicking, scrolling and selecting keys, on a keyboard. The OpenCV computer vision library and the Python programming language are used in the development of the artificial intelligence virtual mouse system.

An AI virtual keyboard and mouse can be used in situations where a physical mouse is not an option, improving human-computer interaction. In lectures and classrooms, it can be used to present on large screens without requiring physical devices. Additionally, it can be used to play games that rely on augmented reality and virtual reality without requiring a wired or wireless keyboard, mouse, or other input devices.

KEY WORDS: Artificial Intelligence, Deep Learning, Autopy, Mediapipe, Opencv, Pyautogui, HandTracking Module, AI virtual mouse, AI virtual Keyword

CHAPTER 1

SUMMARY OF THE BASE PAPER

Base paper Title: Real Time AI Virtual Mouse System Using Learning.

Author List: S. Shriram, B. Nagaraj, J. Jaya, S. Shankar and P. Ajay

Year: October 2021

URL: “<https://www.hindawi.com/journals/jhe/2021/8133076/>”

Base paper is taken from Hindawi.com

Introduction

Our project will aim to design and develop a control computer hands-free application that allows the user to communicate with these machines more efficiently and effectively. These goals can be achieved by understanding the basics of Artificial Intelligence, Machine Learning & using the concepts and algorithms used in Deep Learning.

The AI virtual mouse is a software-based pointing device that allows users to manipulate the movement of the pointer on a computer screen through hand gestures, a touchscreen, or other input devices. It can be used to reduce the space required for a real mouse and in situations where employing one is not feasible. The method eliminates the need for devices while improving human-computer connection. The Virtual Mouse creates an infrastructure between the user and the system using merely a camera. It allows users to interact with machines and control mouse tasks without requiring any hardware.

AI virtual mouse can be controlled using hand gestures and motion-tracking technology. These types of virtual mouse are often used in gaming and other interactive applications, allowing users to control the cursor using natural hand movements. Virtual mouse software can be useful for people who are unable to use a physical mouse due to a physical disability or injury, or for those who prefer to use the keyboard or touchpad instead of a physical mouse. It can also be useful in situations where a physical mouse is not available.

1.1 WHAT IS ARTIFICIAL INTELLIGENCE?

Artificial intelligence (AI) is an area of computer science that focuses on developing intelligent robots that can perform activities that normally require human intellect, such as interpreting natural language, recognizing pictures, making decisions, and learning from experience.

AI algorithms are meant to analyze massive volumes of data, detect patterns, and utilize that knowledge to make choices or take an action. This enables robots to accomplish complicated tasks with greater speed and precision, frequently outperforming human skills.

Numerous industries, including healthcare, banking, transportation, entertainment, and more, have used AI in various ways. As AI develops, it has the ability to transform many facets of society and foster more innovation, efficiency, and problem-solving skills.

The many subfields that comprise AI research are concentrated on certain goals and methods. Reasoning, knowledge representation, planning, learning, natural language processing, sensing, object manipulation, and mobility are a few of the conventional objectives of AI study. The ability to solve any problem is known as general intelligence, and it is one of the field's long-term goals. To tackle these problems, researchers in artificial intelligence (AI) have combined and modified techniques from the fields of formal logic, artificial neural networks, search, statistical optimization, probability, and economics. Artificial intelligence has an impact on numerous academic fields, such as computer science, psychology, linguistics, philosophy, and many more.

Deep learning, machine learning, and rule-based systems are the three categories of artificial intelligence. While machine learning algorithms learn from data and become more effective over time, rule-based systems base their decisions on pre-established rules. Artificial neural networks that are fashioned after the human brain are used in deep learning to analyse and interpret data.

1.2 APPLICATIONS:

Virtual mouse applications are programs or systems that offer substitutes for using a real mouse to operate and interact with a computer. Following are a few typical uses for virtual mouse technology:

Touchscreen Devices: Devices with touch screens but no actual mouse can benefit from using virtual mouse programs. They provide users a virtual cursor that can be moved around the screen using touch gestures, making it easier for them to utilize the device.

Virtual Reality: Virtual mouse apps let users engage in a computer-generated world when they are immersed in virtual reality surroundings. A more natural and immersive experience is produced when users can point, click, and manipulate items using hand gestures or portable devices.

Remote Desktop: Virtual mouse programs are frequently used in remote desktop software or apps, which let users manage a computer from a distance. With the help of these programs, users may easily manage a distant computer thanks to a virtual mouse interface that mimics the capabilities of a real mouse.

Gaming: Gaming apps often make use of virtual mice, particularly on mobile devices and consoles without hardware mice. Players can use touch or motion controls to interact with the game environment, traverse menus, and operate the in-game cursor.

Accessibility: For people with physical limitations or illnesses that make it difficult to operate a regular mouse, virtual mouse apps are crucial. These programs offer many means of input that enable users to interact with the computer, such as eye-tracking technologies, head motions, or gestures.

Presentation and Collaborative Tools: When several people need to control the cursor at once during presentations or group work sessions, virtual mouse programs might be useful. They let users engage with shared displays, papers, or whiteboards by using their own devices as virtual mouse.

Assistive Technology: Applications for a virtual mouse are also used in assistive technology for people with motor or cognitive disabilities. Users may explore and operate computers with the help of these app's streamlined and programmable interfaces by employing natural gestures or switches.

CHAPTER 2

OBJECTIVES:

The main objective of the AI virtual system is to develop a replacement for the traditional mouse system so that it can carry out and manage keyboard and mouse operations. A web camera can be used for this purpose to capture hand gestures and tips, which can subsequently be processed to carry out particular action such as scrolling, keypad selections, and left and right clicks. Our work involves the use of deep learning algorithms and a variety of machine learning approaches. By providing hands-free solutions, the program can help users in situations where they are unable to use tangible devices.

An AI virtual mouse's main goal is to give those with physical impairments or circumstances that make it challenging to use a regular mouse an alternate input method. The virtual mouse intends to make computer control and interaction possible for those with restricted movement or dexterity. The goal of the virtual mouse should be to give people a simple, intuitive way to move about a computer or other digital environment. Users should be able to point, click, drag, and carry out other typical mouse activities without any problems because it should mimic the capabilities of a real mouse. An AI virtual mouse's ability to be customized to meet the unique requirements and preferences of users is one of its key goals. This may involve adjusting the sensitivity, gesture recognition, button layouts, and other settings to suit the needs of different users. The goal of the virtual mouse should be to give accurate and precise pointer control so that users may do activities with great accuracy. For interactions to be seamless and responsive, this goal entails minimizing latency, jitter, or mistakes in cursor movement. To achieve universal use, the virtual mouse should work with a variety of operating systems, software programs, and gadgets. Collaboration and interoperability are made possible by integration with current assistive technology frameworks and accessibility features. The goal of the virtual mouse should be to increase productivity by enabling quick and effective interaction. This goal entails improving the virtual mouse's functionality, responsiveness, and usability so that users can navigate and carry out activities swiftly and efficiently. The continual development and updating of an AI virtual mouse might be advantageous. The goal is to employ machine learning and AI approaches to improve the performance of the virtual mouse over time, adapt to user preferences and behavior.

CHAPTER 3

EXPERIMENTAL WORK / METHODOLOGY

Design and development of algorithms used for Hand Tracking

There are many different tools and languages present for programming applications in the market. However, the tools we will use for the development of this project are pretty-famous and extensively accepted by professional developers.

3.1 Programming languages and tools

Various programming languages are being used nowadays. A few of them are listed below.

3.1.1 Python

So, first thing first, we used Python as a language to implement the code because of its vast feature of having many libraries to use, which can make our life easier to implement the application we want.

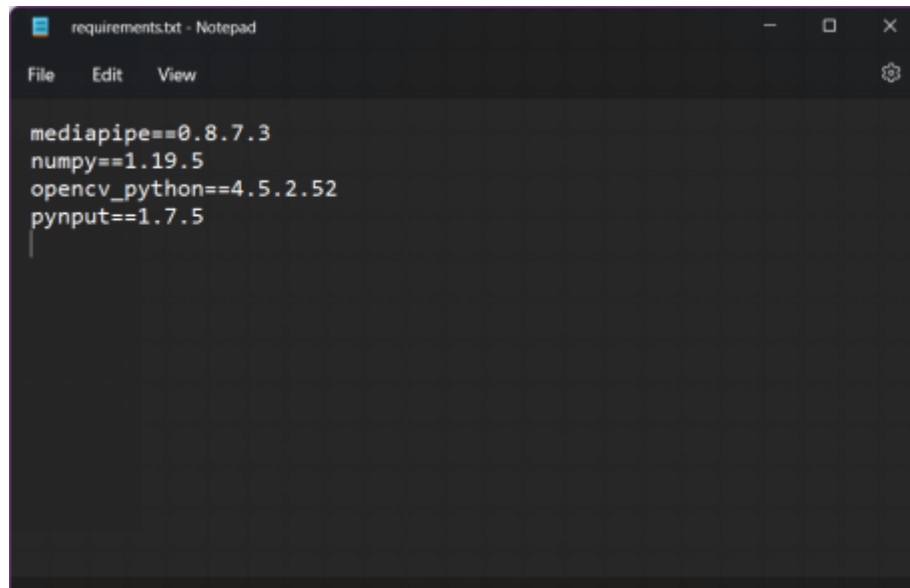
In the late 1980s, Python was designed to replace the Netherlands ABC programming language based on the SETL system that could handle and communicate with the Amoeba operating system by Guido van Rossum in Centrum Wiskunde & Informatics (CWI). It began to be implemented in December 1989.

Python is a high-level, English-like programming language. It does read and understanding the code simpler. Python is easy to collect and learn; thus, many people recommend Python to novices. You require fewer code lines than other prominent languages like C/C++ and Java to do the same thing.

Python is an incredibly productive language. Thanks to Python's simplicity, developers can concentrate on resolving the problems. You do not have to spend much time knowing the programming language syntax or behavior. In a nutshell, you write less code, and you do more. This indicates that Python executes the line code by line directly. Python is an interpreted language.

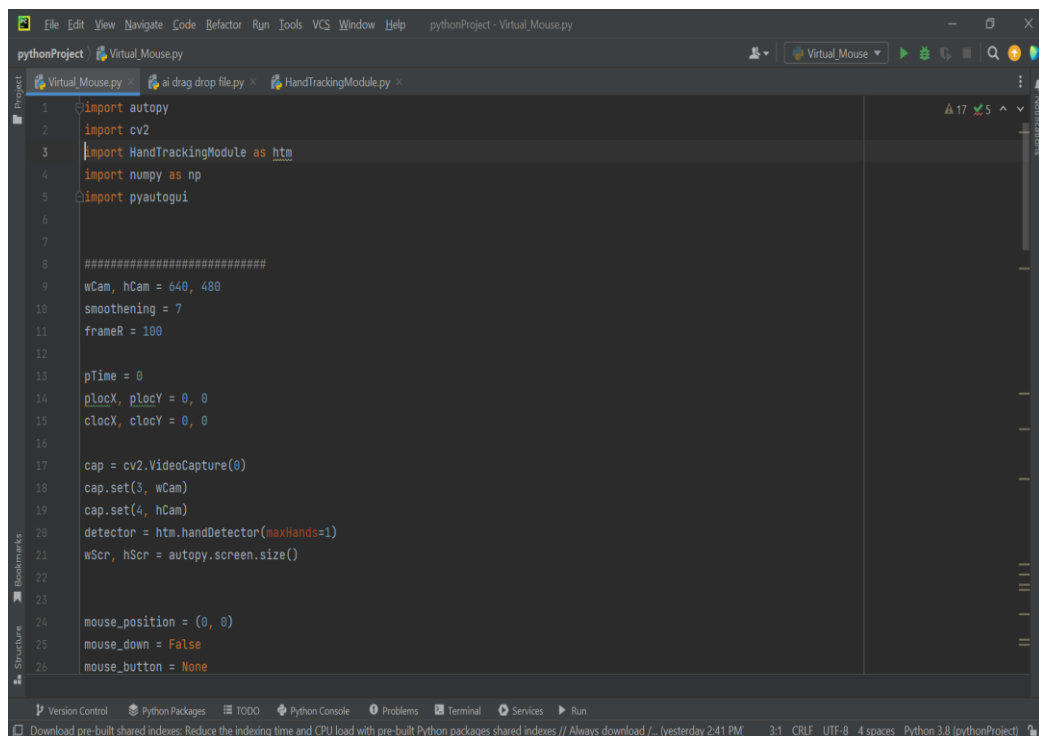
In the event of an error, the execution stops and reports the mistake. Even if the program has several mistakes, Python only shows one error. This facilitates debugging. Until we execute the code, Python does not know the variable type. During execution, it automatically allocates the data type. The programmer need not concern himself with defining variables and their data types

Python is subject to the open-source license authorized by OSI. It can be used and distributed free of charge. Download, change and even share a version of the source code.



```
requirements.txt - Notepad
File Edit View
mediapipe==0.8.7.3
numpy==1.19.5
opencv_python==4.5.2.52
pynput==1.7.5
```

Fig 3.1: Requirements



```
pythonProject - Virtual_Mouse.py
pythonProject Virtual_Mouse.py
Virtual_Mouse.py ai drag drop file.py HandTrackingModule.py
1 import autopy
2 import cv2
3 import HandTrackingModule as htm
4 import numpy as np
5 import pyautogui
6
7
8 #####
9 wCam, hCam = 640, 480
10 smoothing = 7
11 frameR = 100
12
13 pTime = 0
14 plocX, plocY = 0, 0
15 clocX, clocY = 0, 0
16
17 cap = cv2.VideoCapture(0)
18 cap.set(3, wCam)
19 cap.set(4, hCam)
20 detector = htm.handDetector(maxHands=1)
21 wScr, hScr = autopy.screen.size()
22
23
24 mouse_position = (0, 0)
25 mouse_down = False
26 mouse_button = None
```

Fig. 3.2 Python IDE with code Highlighting

3.1.2 NumPy library

NumPy is an essential library of Python's superficial level for complex math. With multi-dimensional array objects, NumPy highly anticipates slower implementation. It features built-in array handling methods. We can transform many algorithms into an array of application functions.

NumPy does not have solely self-confined applications. This library is quite broad and has many applications in different areas. Together with data science, data analysis, and machine education, NumPy may be used. It also forms the basis for several libraries of pythons. These libraries leverage NumPy's features to expand their capacity.

Here are the top four advantages NumPy can bring:

1. More speed: NumPy employs Co-written algorithms which finish nanoseconds instead of Seconds.
2. Fewer loops: NumPy helps you shorten loops and keep iteration indices under check.
3. Clearer code: Without loops, the equations you want to compute are more similar to our code.
4. Better quality: thousands of contributors work quickly, are friendly, and are free of bugs to preserve NumPy.

Due to these advantages, NumPy is a de-facto standard in Python data science for multidimensional arrays, and many of the most popular modules are built on it.

To install numpy : `$ pip install numpy==1.19.5`

3.1.3 Open-CV

OpenCV [3] is an information center of programming tasks aimed primarily at computer vision in sync.

- The Nizhny Novgorod Intel team has established the library.
- Vadim Pisarevsky – the most crucial library contributor.
- Gary Bradski – opened in 1999 with the hope of speeding up OpenCV Computer vision and artificial insight with a solid infrastructure.
- Rapidity - developed for computer efficiency and focused on real-time applications written in optimized C++ and utilizing a multi-core processor
- Portability
- Linux, Windows, and Mac OS X are running
- Python, Java, MATLAB, and other linguistic interfaces
- Android and iOS portable for mobile apps
- A variety of applications and usages provides a computer vision infrastructure

that is easy to use and allows for a reasonably elaborate vision.

- OCV library has over 500 functions, including factory product inspection, covering several fields of vision, medical pictures, security, UI, camera, Stereo View, and Robotics Calibration.
- Functions are well documented but lack diversity as an example.
- OpenCV is not great when influencing pictures.
- OpenCV is excellent for showing how to see something on the computer

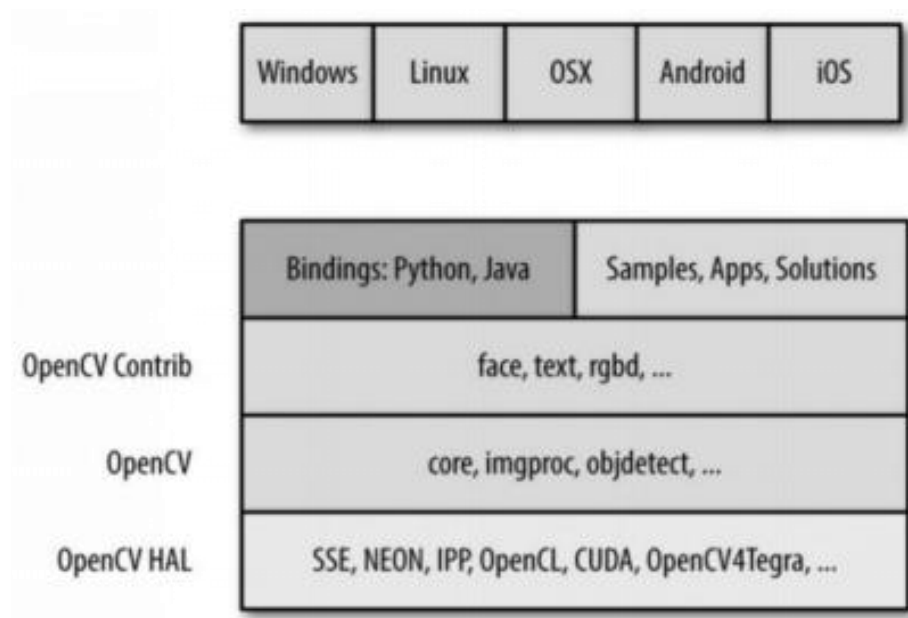


Fig.3.3 OpenCV Block Diagram

OpenCV Demo:

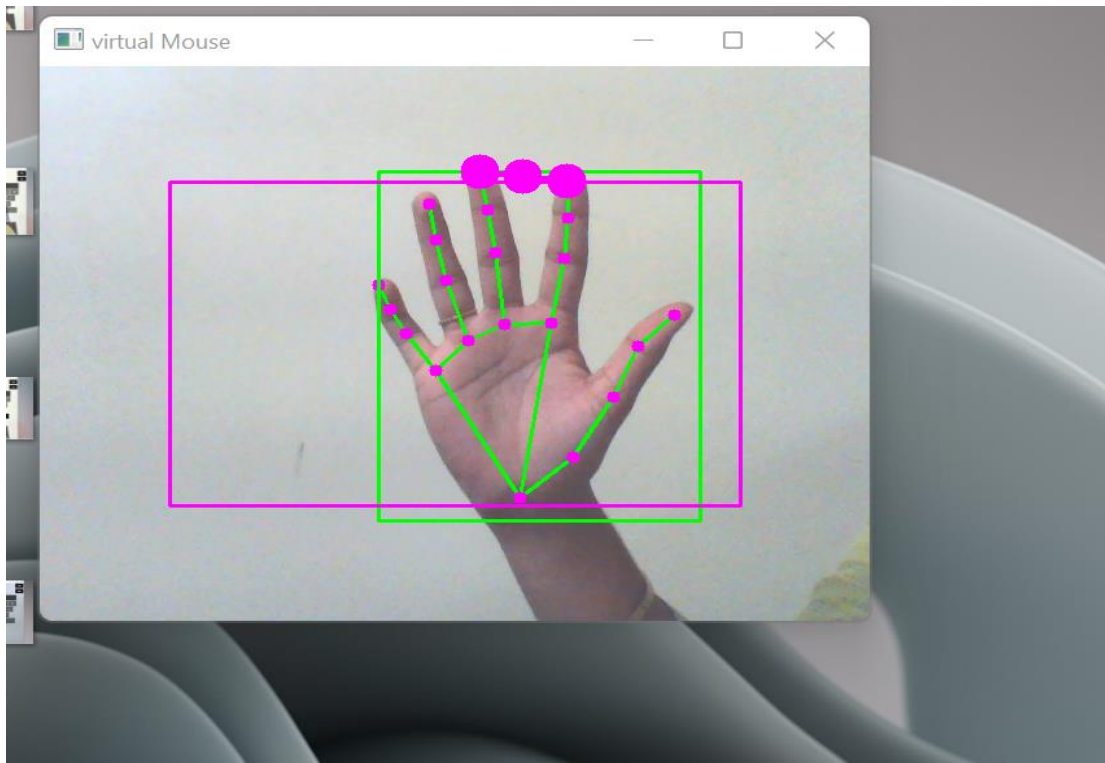


Fig.3.4 OpenCV Demo

Hand detection: Once the hand region is extracted from the camera feed, OpenCV can be used to detect the hand and track its movement. This can be done using techniques such as skin color detection, feature detection, and machine learning-based methods.

Gesture recognition: OpenCV can be used to recognize hand gestures such as clicks, scrolls, and drag-and-drop actions. This can be done using techniques such as template matching, machine learning-based methods, and custom algorithms.

Mouse control: Once the hand movements and gestures are detected, OpenCV can be used to control the position of the mouse cursor on the screen. This can be done by mapping the hand movements to the corresponding mouse movements using calibration technique

3.1.4 MediaPipe:

The potential to recognize hand shapes and movements can be crucial in improvising user experience across different technology domains. For example, using these gestures will form a motive for controlling hand gestures and understanding sign language; it also helps to allow the layer of digital ideas and info above the physical world in AR. While considering humans real-time hand perceptions is a demanding OpenCV task.

Media pipe hands is one of the highly trusted finger and hand tracking solutions. Taking only a single frame into consideration, it utilizes Machine learning (ML) to deduce the 2D and 3D landmarks of the hand. While today's methods depend most on high-performance desktops, this method can attain better results even on mobile phones, and it also scales to multiple hands.

Hence, more research may be carried out on this hand perception functionality, which will implement new simulations, approach to new research, and exposure to new creative uses.

3.1.5 ML Pipeline:

MediaPipe Hands uses a machine learning pipeline that includes many models which work conjointly. The palm detection version works throughout the photo and provides the output as an aligned hand bounding box. On the other hand, the hand detection model works only in the cropped image area specified by the palm detection model and provides output as a 3D hand key point [8]. The strategy which has been used above is the same as that of the MediaPipe face recognizing remedy, which utilizes the face identifier along with that of the hand landmark replica.

The need for data augmentation can be decreased by supplying the precisely cropped hand image. This, in turn, permits the grid to bind much of its attention towards coordinating forecast reliability. Furthermore, cropped images can also be created in our pipeline based on the hand indicator recognized in the preceding fixture. The unique model in which presence of the hand cannot be found is the palm detection model, which involves repositioning the hand.

The pipeline execution will be done as a MediaPipe graph that utilizes the hand indicator tracing subgraph. It utilizes a palm hand indicator subgraph from the hand identifier part and the palm identification subgraph from the palm identification model.

3.1.6 AUTOPY:

Autopy is a Python library that provides a simple interface for simulating mouse and keyboard input, as well as for taking screenshots and manipulating windows. In a virtual mouse system, Autopy can be used to simulate mouse movements and clicks in response to user input.

To use Autopy as a virtual mouse, you can import the `autopy.mouse` module and call its various functions to move the mouse pointer, click the mouse buttons, and so on. For example, the `move` function can be used to move the mouse pointer to a specific screen position.

```
import autopy

# Move the mouse pointer to position (100, 100) on the screen
autopy.mouse.move(100, 100)
```

```
import autopy

# Simulate a left mouse button click at the current mouse position
autopy.mouse.click()
```

3.2 MODELS:

3.2.1 Palm identification Model:

As detecting hands is a more complicated process, and to detect both the self-occluded and occluded hands, our model (lite and full) must be able to work on various hand sizes and shapes to the image frame. Also, there is a lack of contract patterns of high-quality like in the face, mouth, and the eye region, but in the hands, as there is no such area, it is challenging to detect alone from their visual features. Apart from that, we can provide extra information such as personal features and the body, which helps in detecting the hand location

The method that we are using helps overcome all the problems mentioned above using different strategies. The first step we follow for this is that we will work more on a palm detector, as palm detection can be much more straightforward. Estimating the bounding box for a rigid object like a palm will be much simpler. Also, as palms can be detected taking the bounding box conditions as squares, we can neglect all the other aspect ratios and reduce the number of anchors by 3 to 5 seconds. By leveraging significant variance, you can eliminate or minimize focus loss during working to pillar a large number of anchors.

3.2.2 Hand Landmark Model:

The hand landmark model carries out the precise key point localization of 21 3D hand knuckle coordinates of the interior of the hand region, which has been detected through regression. It all takes place after the palm detection is completed. This is the Direct, coordinated prediction. As this model gains an idea about the accordant position for representing the hand, it also detects the self-occlusions and hands that are partially visible.

As shown in the image below, to get the ground truth data manually, we have annoyed ~30K images of the natural world with coordinates of 21 3D (If the corresponding coordinate value exists, the Z value can be obtained from the depth map of the shown image). We have also rendered a high-quality artificial hand model on different backdrops to cover all the possibilities in hand poses. We mapped it according to the obtained 3D coordinates.

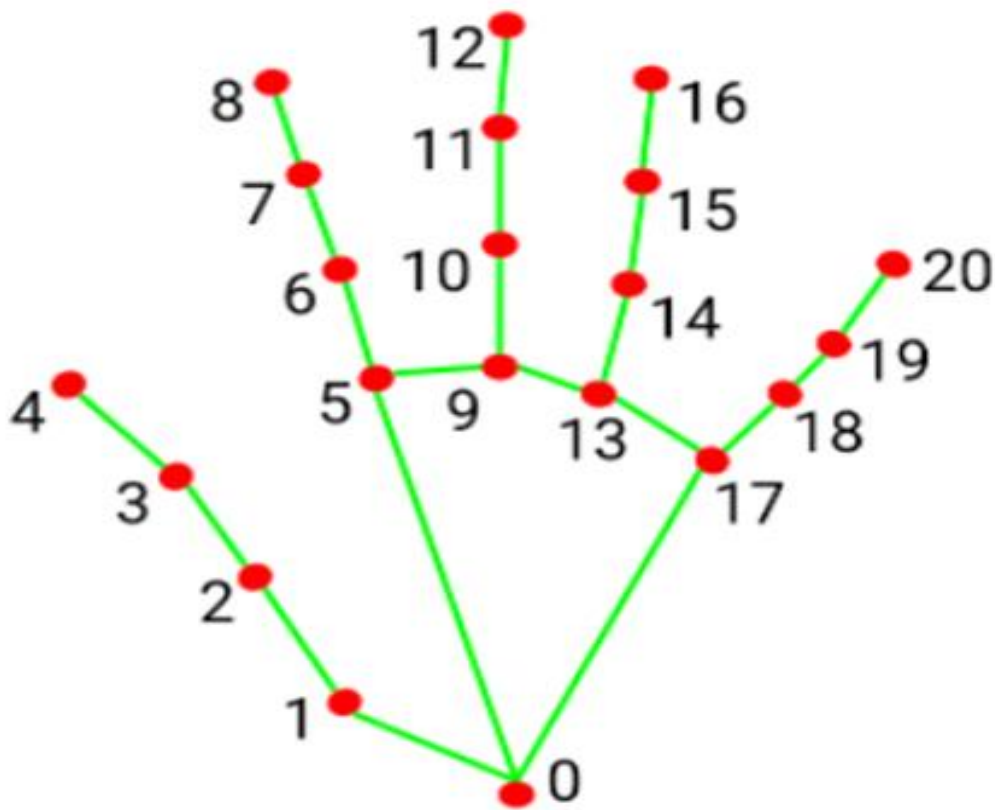


Fig.3.6 Hand Landmark Model

3.3 Methodology:

The procedure followed in the project is explained in the flowchart.

Step involved -

1. video capturing commences.
2. Capturing the frames using Webcam
3. Detecting the tips of the hand using Media Pipe and OpenCV.
4. Draw a rectangle box which is the workspace for using the mouse.
5. Detect which finger is UP in order to perform diction functions.
 - 1) When the forefinger is up then it acts as a mouse cursor
 - 2) When both index and middle finger are up and distance < 39 , it performs a left click.
 - 3) When both index and middle finger are up and distance > 39 , it performs a right click.
 - 4) When the ring finger is up, it performs the scroll up function.
 - 5) When the little finger is up, it performs the scroll down function.
 - 6) When both index and thumb fingers are up and distance < 0.5 threshold, it performs a drag function.
 - 7) When all the five fingers are up zoom in function is executed.
 - 8) When all fingers except thumb should be down, then zoom out function is executed.
6. To terminate the process press stop.

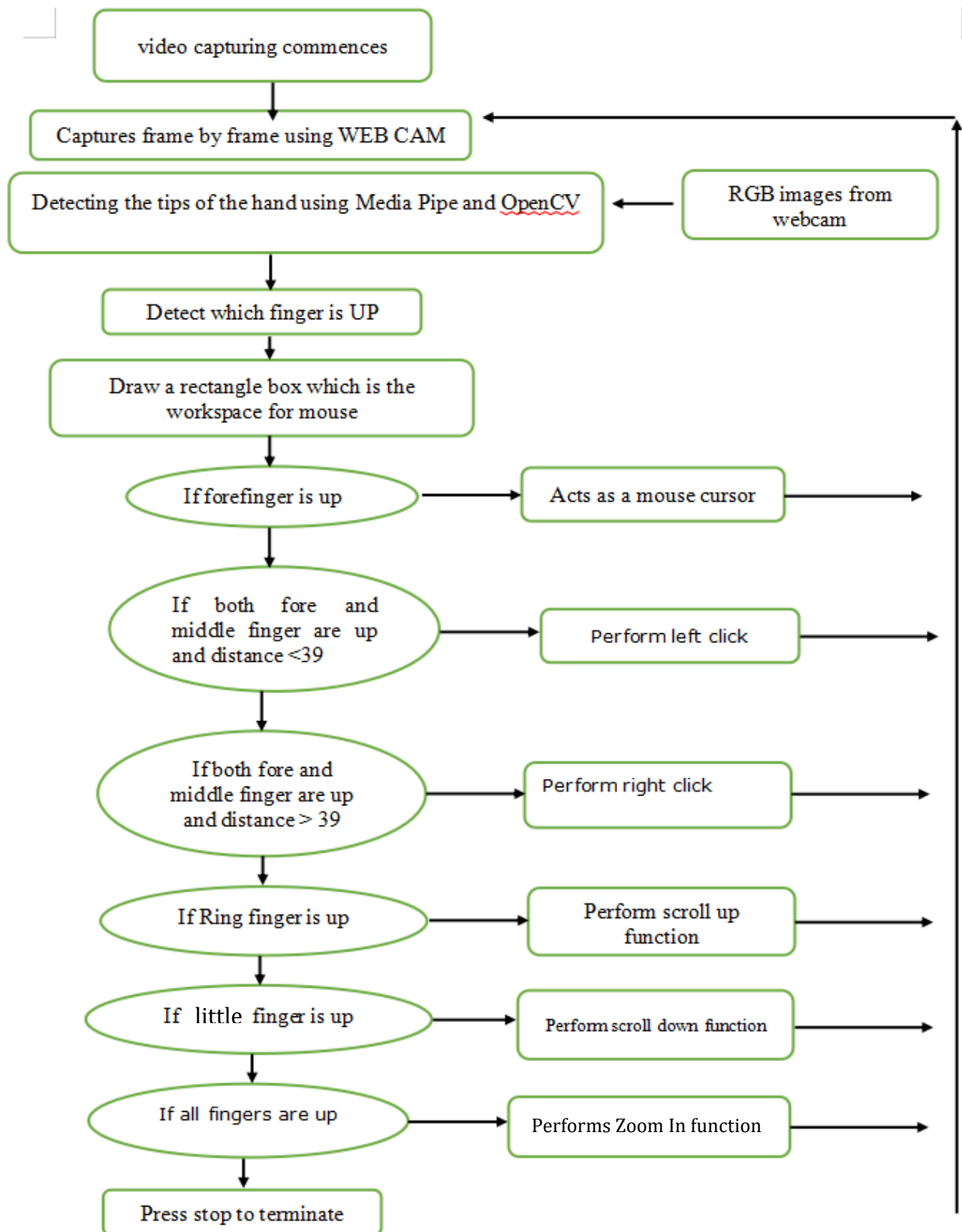


Fig:3.7 Methodology Flowchart

RESULTS & DISCUSSION

4.1 HAND GESTURES & THEIR RESPECTIVE FUNCTIONS:

4.1.1 AI VIRTUAL MOUSE:

- **CURSOR MOVEMENT:**

In order to move the cursor, the forefinger should be up then the mouse cursor is made to move around the window of the computer.

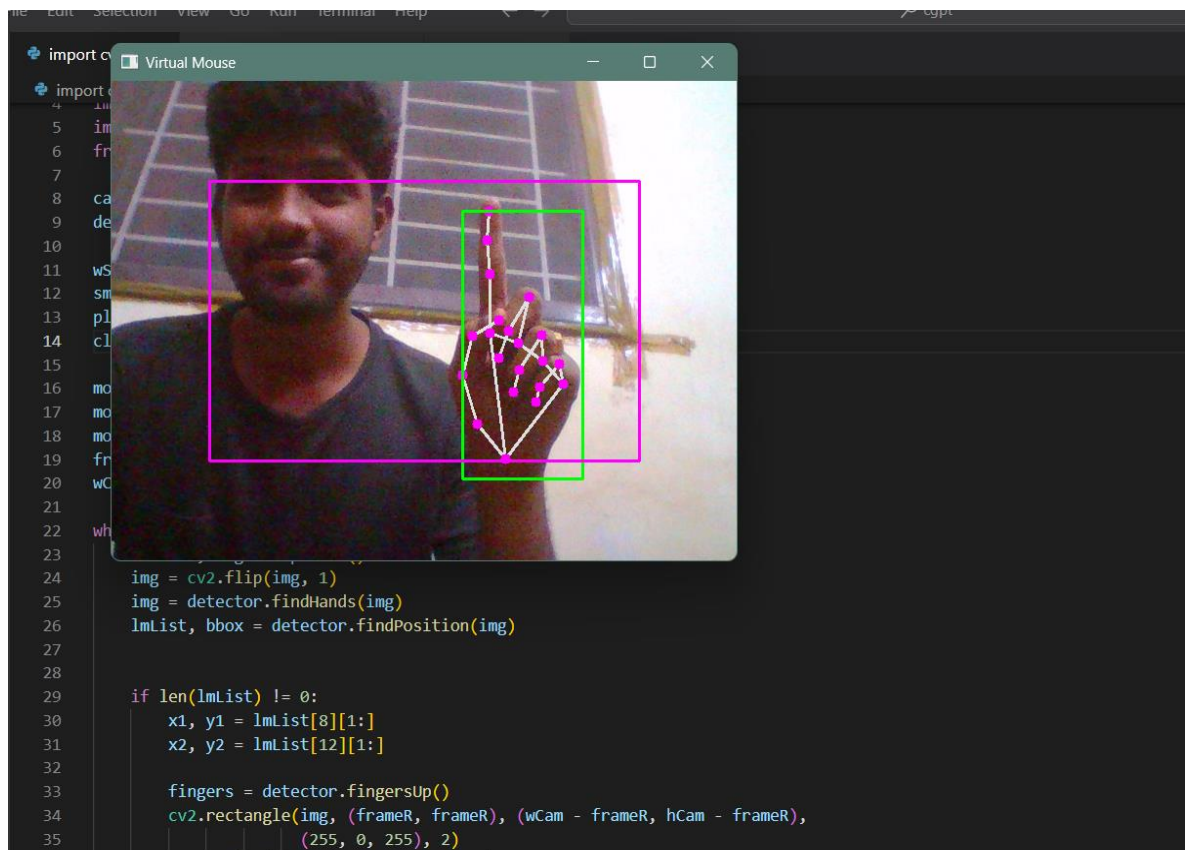


Fig.4.1 TO MOVE THE CURSOR

- **LEFT CLICK:**

In order to execute the left click, both the forefinger and middle finger should be up, and distance should be less than 39, then the system will perform the Left click function.

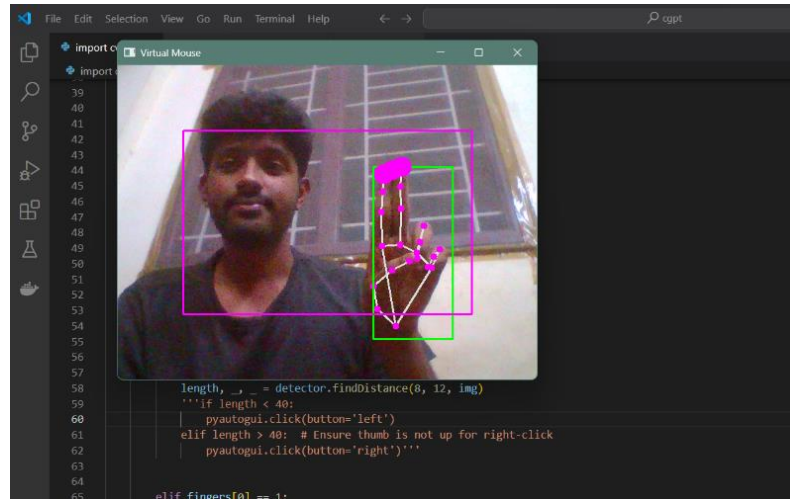


Fig.4.2 LEFT CLICK

- **RIGHT CLICK:**

In order to perform a Double click, both the forefinger and middle finger should be up, and distance should be greater than 39, then the system will recognize it as a Right click function.

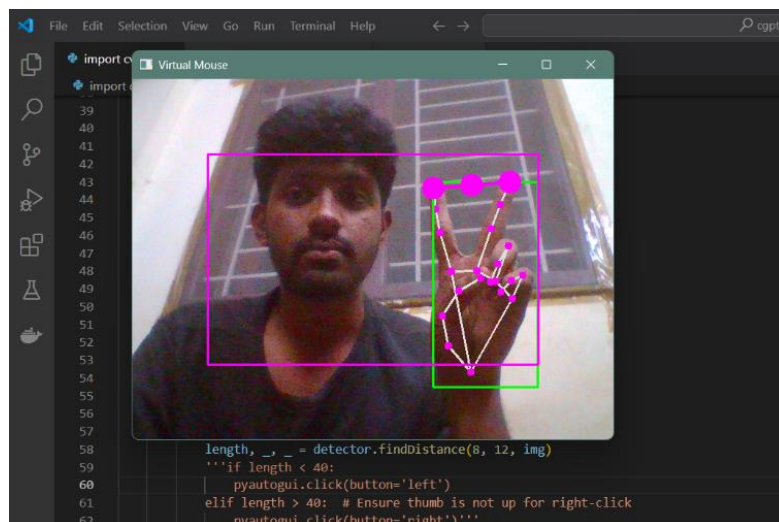


Fig.4.3 RIGHT CLICK

- **SCROLL UP FUNCTION:**

When we hold the ring finger up, then the system will perform a scroll up function according to the movement of the hand.

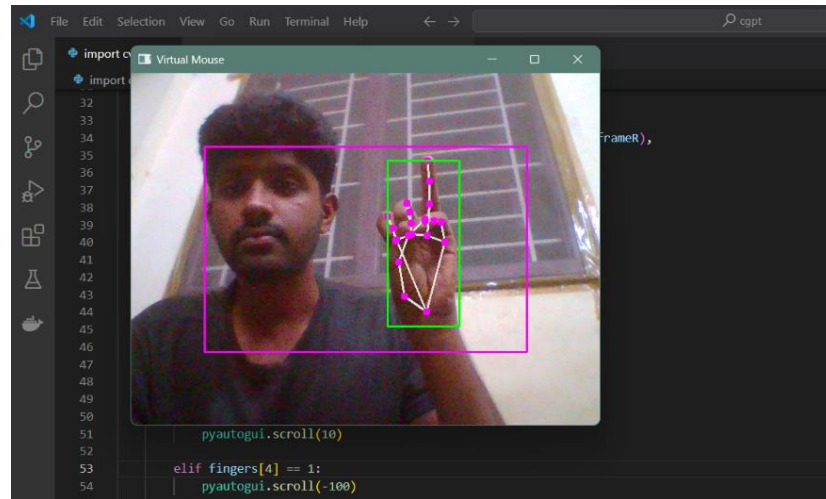


Fig.4.4 SCROLL UP FUNCTION

- **SCROLL DOWN FUNCTION:**

When we hold the little finger up, then the system will perform a scroll down function according to the movement of the hand.

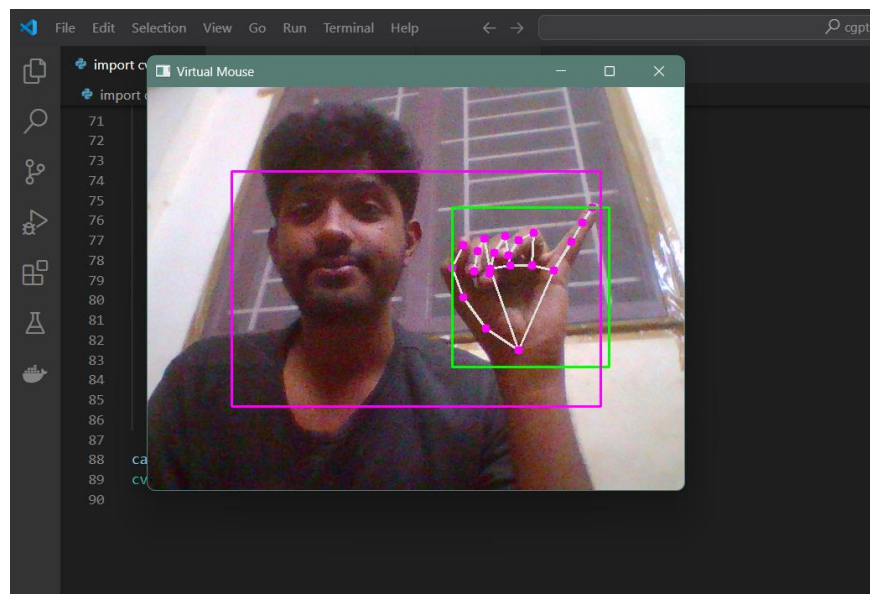


Fig.4.5 SCROLL DOWN FUNCTION

- **DRAG AND DROP FUNCTION:**

To perform the drag and drop function, the index and thumb finger should be up and curl distance

between them is less than 0.5; then, the system will perform the drag function.

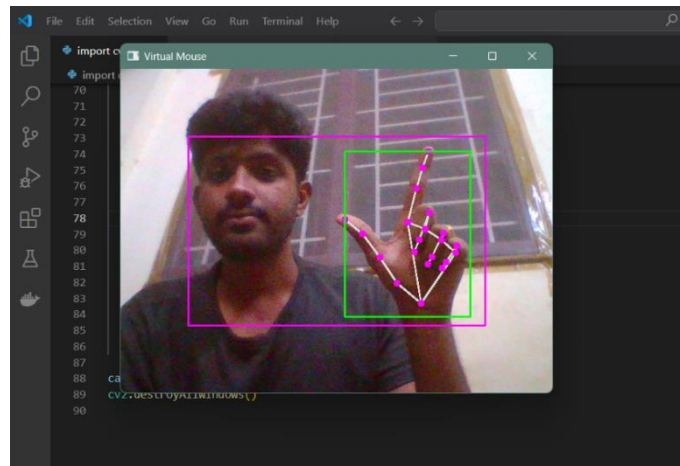


Fig.4.6 DRAG AND DROP FUNCTION

- **ZOOM IN FUNCTION:**

When all fingers except thumb should be down, then zoom out function is executed.

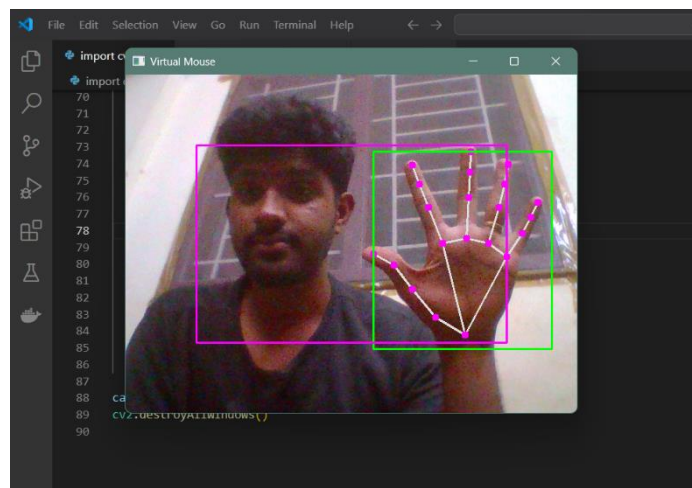


Fig.4.6 ZOOM IN FUNCTION

4.1.2 AI VIRTUAL KEYBOARD:

With the help python Virtual keyboard interface is created when we use the hand tracking module in order to perform different keyboard functions.

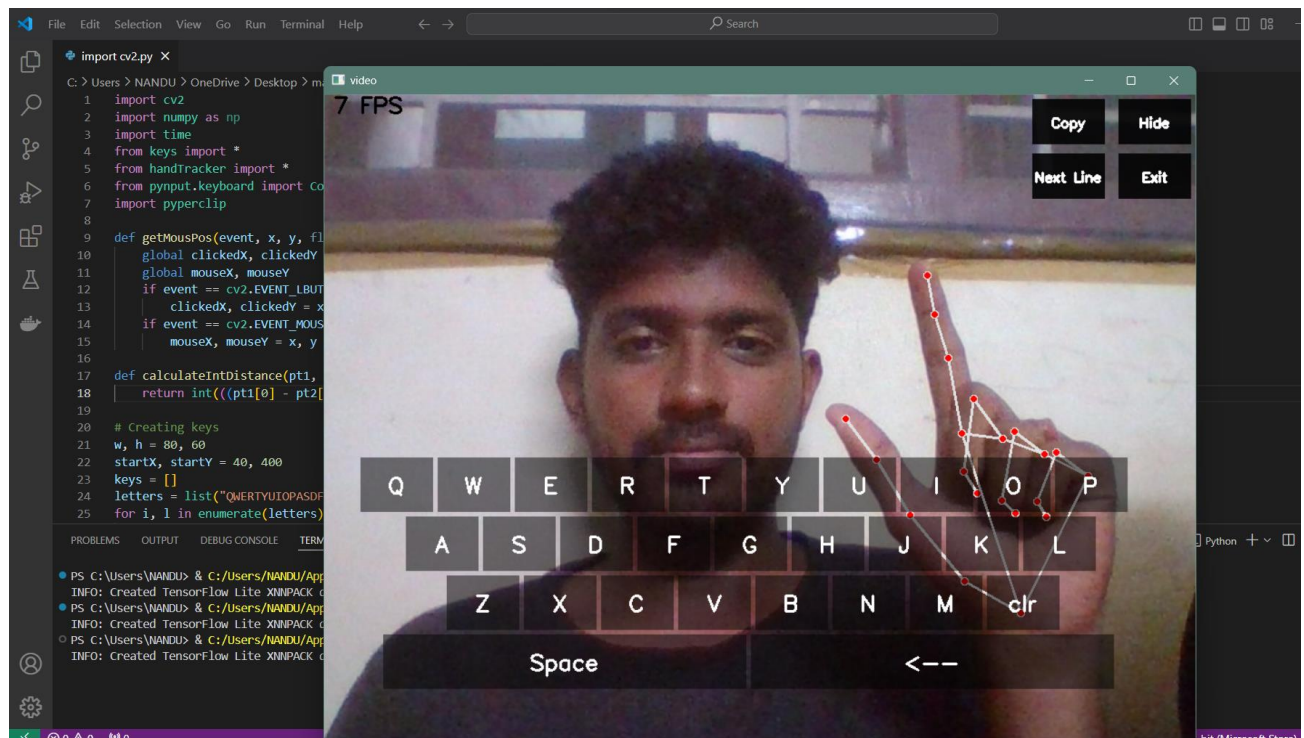


Fig.4.7 KEYBOARD INTERFACE

Key Selections:

To select a keyboard key, we need to tap the forefinger and thumb finger and to move the cursor around the keyboard index finger is used.

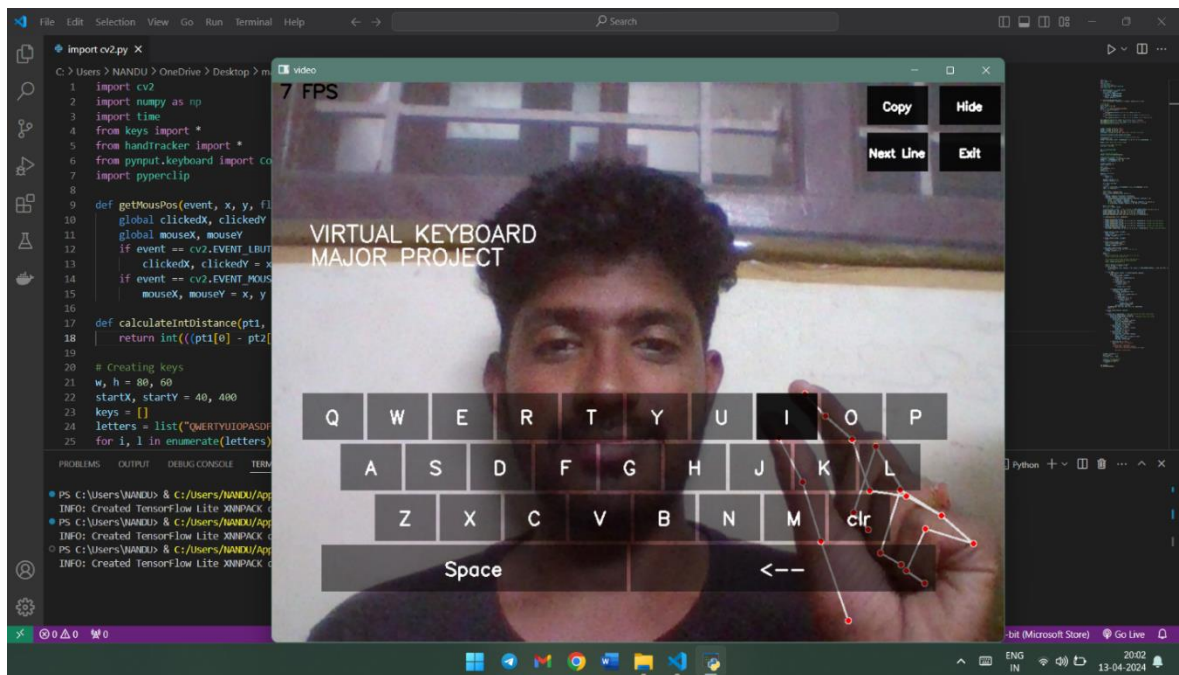


Fig.4.8 KEY SELECTION

Test results of hand gestures for mouse cursor control:

Gesture Name	Input images	Validated images	Recognition rate (input/validated)*100	Function against gesture
Gesture with index finger open	60	59	98.33	Cursor control
Gesture with index and middle finger open and placed close.	60	58	96.66	Left click
Gesture with index finger and middle finger open and placed with some distance.	60	55	91.66	Right click
Gesture with ring finger open.	60	58	96.66	Scroll up
Gesture with little finger open.	60	58	96.66	Scroll down
Gesture with all fingers open.	60	57	95.00	Zoom in
Gesture with index finger open.	60	55	91.66	Screenshot

Fig.4.9 results of hand gestures for mouse cursor control

Test results of hand gestures for virtual keyboard control:

Gesture Name	Input images	Validated images	Recognition rate (input/validated)*100
Gesture when clicked “Show”	60	57	95.00
Gesture when clicked “Hide”	60	57	95.00
Gesture when clicked “Next line”	60	55	91.66
Gesture when clicked “Exit”	60	58	96.66
Gesture when clicked Character keys	60	58	96.66
Gesture when clicked Numerical keys	60	58	96.66
Gesture when clicked Space & Clear	60	55	91.66

Fig.4.10 results of hand gestures for virtual keyboard control

CHAPTER-5

CONCLUSIONS AND FUTURE SCOPE

Hand Gesture detection is a field that has immense potential for numerous innovations. As we have implemented this particular project using Deep Learning concepts, anyone who wants to bring even a tiny change in this field can make it by learning the basics and applying the knowledge gained to his/her project.

These systems can be placed in offices and universities for better accessibility and a more straightforward user interface. The developers implemented this project via software, and we want further to convert it into Hardware via IoT.

IoT can be beneficial for people who are not familiar with the software. Nevertheless, due to Covid situations, we were unable to manage hardware, and also, we could not have worked on Hardware together, the cause of these pandemic situations.

Most physical keyboards & Mouses are not the most comfortable and Convenient. This is an actual-time application and User friendly. The project removes the requirement of having a physical mouse and Keyboard. This system can be integrated into desktops, laptops, and tablets for much more convenience for the user.

CHAPTER 6

REFERENCES

- <https://www.hindawi.com/journals/jhe/2021/8133076/>
- <https://ieeexplore.ieee.org/abstract/document/7489570>
- <https://ieeexplore.ieee.org/abstract/document/9456388>
- <https://ieeexplore.ieee.org/abstract/document/9137854>
- D.-S. Tran, N.-H. Ho, H.-J. Yang, S.-H. Kim, and G. S. Lee, “Real-time virtual mouse system using RGB-D images and fingertip detection,” *Multimedia Tools and Applications*, vol. 80, no. 7, pp. 10473–10490, 2021.
- In 2021, Surya Narayan Sharma and Dr A Rengarajan published a paper, “Hand Gesture Recognition using OpenCV and Python
- <https://technologymoon.com/community/artificial-intelligence-f30/where-did-ai-come-from-t58685.html>
- <https://www.mulheresquefazemadiferenca.com.br/when-should-iskq/machine-learning-is-a-subset-of-a0edbc>
- Mašinska vizija - (2009) <https://opencv.org/books>
- Adrian Kaehler – (2008) *Learning OpenCV 3: Computer Vision in C++ with OpenCV Library* - Fig.2.5 OpenCV Block D
- J. T. Camillo Lugaresi, "MediaPipe: A Framework for Building Perception Pipelines," 2019, <https://arxiv.org/abs/1906.08172>
- <https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html>

CHAPTER 7

7.1 SOURCE CODE

Virtual Mouse Code:

HandTracker.py

```
HandTrackingModule.py X
C:\Users\NANDU> OneDrive\ Desktop\ major proj\ mouse > HandTrackingModule.py handDetector > _init_

1 import cv2
2 import mediapipe as mp
3 import math
4
5 class handDetector():
6     def __init__(self, mode=False, maxHands=2, detectionCon=0.5, trackCon=0.5):
7         self.mode = mode
8         self.maxHands = maxHands
9         self.detectionCon = detectionCon
10        self.trackCon = trackCon
11
12        self.mpHands = mp.solutions.hands
13        self.hands = self.mpHands.Hands(static_image_mode=self.mode, max_num_hands=self.maxHands, min_detection_confidence=self.detectionCon, min_tracking_confidence=self.trackCon)
14
15        self.mpDraw = mp.solutions.drawing_utils
16        self.tipIds = [4, 8, 12, 16, 20]
17
18        # Add event handling variables
19        self.mouse_position = (0, 0)
20        self.mouse_down = False
21        self.mouse_button = None
22
23    def findHands(self, img, draw=True):
24        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
25        self.results = self.hands.process(imgRGB)
26
27        if self.results.multi_hand_landmarks:
28            for handLms in self.results.multi_hand_landmarks:
29                if draw:
30                    self.mpDraw.draw_landmarks(img, handLms, self.mpHands.HAND_CONNECTIONS)
31        return img
32
33    def findPosition(self, img, handNo=0, draw=True):
34        xlist = []
35        ylist = []
36        bbox = []
37        self.lmlist = []
38        if self.results.multi_hand_landmarks:
39            myHand = self.results.multi_hand_landmarks[handNo]
40            for id, lm in enumerate(myHand.landmark):
41                h, w, c = img.shape
42                cx, cy = int(lm.x * w), int(lm.y * h)
43                xlist.append(cx)
44                ylist.append(cy)
45                self.lmlist.append([id, cx, cy])
46
47                if draw:
48                    cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)
49            xmin, xmax = min(xlist), max(xlist)
50            ymin, ymax = min(ylist), max(ylist)
51            bbox = xmin, ymin, xmax, ymax
52
53            if draw:
54                cv2.rectangle(img, (bbox[0], bbox[1]), (bbox[2] + 20, bbox[3] + 20), (0, 255, 0), 2)
55
56        return self.lmlist, bbox
57
58    def findDistance(self, p1, p2, img, draw=True):
59        x1, y1 = self.lmlist[p1][1], self.lmlist[p1][2]
60        x2, y2 = self.lmlist[p2][1], self.lmlist[p2][2]
61        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
62
63        if draw:
64            cv2.circle(img, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
65            cv2.circle(img, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
66            cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
67            cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)
68
69        length = math.hypot(x2 - x1, y2 - y1)
70        return length, img, [x1, y1, x2, y2, cx, cy]
71
72    def fingersUp(self):
73        fingers = []
74
75        # Thumb
76        if self.lmlist[self.tipIds[0]][1] > self.lmlist[self.tipIds[0] - 1][1]:
77            fingers.append(1)
78        else:
79            fingers.append(0)
80
81        # 4 fingers
82        for id in range(1, 5):
83            if self.lmlist[self.tipIds[id]][2] < self.lmlist[self.tipIds[id] - 2][2]:
84                fingers.append(1)
85            else:
86                fingers.append(0)
87        return fingers
```

```

89  def handle_mouse_event(self, event, x, y, flags, param):
90      self.mouse_position = (x, y)
91      if event == cv2.EVENT_LBUTTONDOWN:
92          self.mouse_down = True
93          self.mouse_button = 'left'
94      elif event == cv2.EVENT_LBUTTONUP:
95          self.mouse_down = False
96          self.mouse_button = None
97      elif event == cv2.EVENT_RBUTTONDOWN:
98          self.mouse_down = True
99          self.mouse_button = 'right'
100     elif event == cv2.EVENT_RBUTTONUP:
101         self.mouse_down = False
102         self.mouse_button = None
103

```

Main.py:

```

C: > Users > NANDU > OneDrive > Desktop > major proj > mouse > main.py > ...
1  import cv2
2  import mediapipe as mp
3  import pyautogui
4  import numpy as np
5  import math
6  from HandTrackingModule import handDetector
7  import time
8  import keyboard
9
10 cap = cv2.VideoCapture(0)
11 detector = handDetector(maxHands=1)
12
13 wScr, hScr = pyautogui.size()
14 smoothening = 7
15 plocX, plocY = 0, 0
16 clocX, clocY = 0, 0
17
18 mouse_position = (0, 0)
19 mouse_down = False
20 mouse_button = None
21 frameR = 100
22 wCam, hCam = 640, 480
23
24 while True:
25     success, img = cap.read()
26     img = cv2.flip(img, 1)
27     img = detector.findHands(img)
28     lmList, bbox = detector.findPosition(img)
29
30
31     if len(lmList) != 0:
32         x1, y1 = lmList[8][1:]
33         x2, y2 = lmList[12][1:]
34
35         fingers = detector.fingersUp()
36         cv2.rectangle(img, (frameR, frameR), (wCam - frameR, hCam - frameR),
37                       (255, 0, 255), 2)

```

```

38     # 4. Only index Finger :Moving Mode
39     if fingers[1] == 1 and fingers[2] == 0:
40         # 5. convert coordinates
41         x3 = np.interp(x1, (frameR, wCam - frameR), (0, wScr))
42         y3 = np.interp(y1, (frameR, hCam - frameR), (0, hScr))
43
44         # Update interpolation for vertical movement
45         clocX = plocX + (x3 - plocX) / smoothening
46         clocY = plocY + (y3 - plocY) / smoothening
47
48         # Move Mouse
49         pyautogui.moveTo(clocX, clocY)
50         plocX, plocY = clocX, clocY
51
52     elif fingers[3] == 1:
53         pyautogui.scroll(50)
54
55     elif fingers[4] == 1:
56         pyautogui.scroll(-50)
57
58
59     elif fingers[1] == 1 and fingers[2] == 1:
60         length, _, _ = detector.findDistance(8, 12, img)
61         if length < 40:
62             pyautogui.click(button='left')
63         elif length > 40: # Ensure thumb is not up for right-click
64             pyautogui.click(button='right')
65
66     elif all(fingers):
67         pyautogui.keyDown('ctrl')
68         pyautogui.scroll(1)
69         time.sleep(0.1)
70         pyautogui.keyUp('ctrl')
71
72     # Check if all fingers are closed for zoom out
73     elif fingers[0] == 1 and fingers[4] == 1:
74         pyautogui.keyDown('ctrl')

```

```

75         pyautogui.scroll(-0.5)
76         time.sleep(0.1)
77         pyautogui.keyUp('ctrl')
78
79     '''elif fingers[0] == 1:
80         # Capture a screenshot
81         pyautogui.screenshot()
82
83         # Save the screenshot (you can customize the filename and location)
84         screenshot.save("screenshot.png") '''
85
86
87     else:
88         if mouse_down:
89             pyautogui.mouseUp()
90             mouse_down = False
91
92     cv2.imshow("Virtual Mouse", img)
93     if cv2.waitKey(1) == 27: # Press 'Esc' to exit
94         break
95
96 cap.release()
97 cv2.destroyAllWindows()

```

Virtual Keyboard:

HandTracker.py:

```
handTracker.py X
C:\Users\NANDU> OneDrive\ Desktop\ major proj\ keyboard> handTracker.py > ...
1 import mediapipe as mp
2 import numpy as np
3 import cv2
4
5 class HandTracker():
6     def __init__(self, mode=False, maxHands=2, detectionCon=0.5, trackCon=0.5):
7         self.mode = mode
8         self.maxHands = maxHands
9         self.detectionCon = detectionCon
10        self.trackCon = trackCon
11
12        self.mpHands = mp.solutions.hands
13        self.hands = self.mpHands.Hands(self.mode, self.maxHands, self.detectionCon, self.trackCon)
14        self.mpDraw = mp.solutions.drawing_utils
15
16    def findHands(self, img, draw=True):
17        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
18        self.results = self.hands.process(imgRGB)
19
20        if self.results.multi_hand_landmarks:
21            for handLm in self.results.multi_hand_landmarks:
22                if draw:
23                    self.mpDraw.draw_landmarks(img, handLm, self.mpHands.HAND_CONNECTIONS)
24        return img
25
26    def getPostion(self, img, handNo = 0, draw=True):
27        lmList = []
28        if self.results.multi_hand_landmarks:
29            myHand = self.results.multi_hand_landmarks[handNo]
30            for id, lm in enumerate(myHand.landmark):
31                h, w, c = img.shape
32                cx, cy = int(lm.x*w), int(lm.y*h)
33                lmList.append([id, cx, cy])
34
35                if draw:
36                    cv2.circle(img, (cx, cy), 5, (255,0,255), cv2.FILLED)
37        return lmList
```

Keys.py:

```
keys.py X
C:\Users\NANDU> OneDrive\ Desktop\ major proj\ keyboard> keys.py > ...
1 import cv2
2 import numpy as np
3
4 class Key():
5
6     def __init__(self,x,y,w,h,text):
7         self.x = x
8         self.y = y
9         self.w = w
10        self.h = h
11        self.text=text
12
13    def drawkey(self, img, text_color=(255,255,255), bg_color=(0,0,0),alpha=0.5, fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, thickness=2):
14
15        #draw the box
16        bg_rec = img[self.y : self.y + self.h, self.x : self.x + self.w]
17        white_rect = np.ones(bg_rec.shape, dtype=np.uint8) * 255
18        white_rect[:] = bg_color
19        res = cv2.addWeighted(bg_rec, alpha, white_rect, 1-alpha, 1.0)
20
21        # Putting the image back to its position
22        img[self.y : self.y + self.h, self.x : self.x + self.w] = res
23
24        #put the letter
25        tetx_size = cv2.getTextSize(self.text, fontFace, fontScale, thickness)
26        text_pos = (int(self.x + self.w/2 - tetx_size[0][0]/2), int(self.y + self.h/2 + tetx_size[0][1]/2))
27        cv2.putText(img, self.text,text_pos , fontFace, fontScale,text_color, thickness)
28
29    def isOver(self,x,y):
30        if (self.x + self.w > x > self.x) and (self.y + self.h > y > self.y):
31            return True
32        return False
```

VirtualKeyboard.py:

```
numberkeys.py X
C: > Users > NANDU > OneDrive > Desktop > major proj > keyboard > numberkeys.py > ...

1  import cv2
2  import numpy as np
3  import time
4  from keys import *
5  from handTracker import *
6  from pynput.keyboard import Controller
7  import pyperclip
8
9  def getMousPos(event, x, y, flags, param):
10     global clickedX, clickedY
11     global mouseX, mouseY
12     if event == cv2.EVENT_LBUTTONDOWN:
13         clickedX, clickedY = x, y
14     if event == cv2.EVENT_MOUSEMOVE:
15         mouseX, mouseY = x, y
16
17  def calculateIntDistance(pt1, pt2):
18     return int(((pt1[0] - pt2[0]) ** 2 + (pt1[1] - pt2[1]) ** 2) ** 0.5)
19
20  # Creating keys
21  w, h = 80, 60
22  startX, startY = 40, 400
23  start1X, start1Y = 40, 330
24  keys = []
25  numbers = list("1234567890")
26  letters = list("QWERTYUIOPASDFGHJKLZXCVBNM")
27  for i, l in enumerate(numbers):
28     keys.append(Key(startX + i * w + i * 5, start1Y, w, h, l))
29  for i, l in enumerate(letters):
30     if i < 10:
31         keys.append(Key(startX + i * w + i * 5, startY, w, h, l))
32     elif i < 19:
33         keys.append(Key(startX + (i - 10) * w + i * 5, startY + h + 5, w, h, l))
34     else:
35         keys.append(Key(startX + (i - 19) * w + i * 5, startY + 2 * h + 10, w, h, l))
36
37  keys.append(Key(startX + 25, startY + 3 * h + 15, 5 * w, h, "Space"))

38  keys.append(Key(startX + 8 * w + 50, startY + 2 * h + 10, w, h, "clr"))
39  keys.append(Key(startX + 5 * w + 30, startY + 3 * h + 15, 5 * w, h, "<--"))
40  #keys.append(Key(25,170,900,200, ""))
41
42
43
44  showKey = Key(300, 5, 80, 50, 'Show')
45  exitKey = Key(300, 65, 80, 50, 'Exit')
46  copyKey = Key(780, 5, 80, 50, 'Copy')
47  nextLineKey = Key(780, 65, 80, 50, 'Next Line') # New Next Line button
48
49  superbuttons=[showKey,exitKey,copyKey,nextLineKey]
50
51  # Adjusted height for the text box to accommodate multiple lines
52  textBoxHeight = 120
53  textBox = Key(startX, startY - textBoxHeight - 5, 10 * w + 9 * 5, textBoxHeight, '')
54
55  delayo = 1.0 # Adjust this value as needed
56
57  # Time of the last Next Line action
58  last_time = time.time()
59
60
61  cap = cv2.VideoCapture(0)
62  ptime = 0
63
64  # Initiating the hand tracker
65  tracker = HandTracker(detectionCon=1)
66
67  # Getting frame's height and width
68  frameHeight, frameWidth, _ = cap.read()[1].shape
69  showKey.x = int(frameWidth * 1.5) - 85
70  exitKey.x = int(frameWidth * 1.5) - 85
71
72  clickedX, clickedY = 0, 0
73  mouseX, mouseY = 0, 0
74
```



```

75 show = False
76 cv2.namedWindow('video')
77 counter = 0
78 previousClick = 0
79
80 keyboard = Controller()
81 while True:
82     if counter > 0:
83         counter -= 1
84
85     signTipX, signTipY = 0, 0
86     thumbTipX, thumbTipY = 0, 0
87
88     ret, frame = cap.read()
89     if not ret:
90         break
91     frame = cv2.resize(frame, (int(frameWidth * 1.5), int(frameHeight * 1.5)))
92     frame = cv2.flip(frame, 1)
93
94     # Find hands
95     frame = tracker.findHands(frame)
96     lmList = tracker.getPostion(frame, draw=False)
97     if lmList:
98         signTipX, signTipY = lmList[8][1], lmList[8][2]
99         thumbTipX, thumbTipY = lmList[4][1], lmList[4][2]
100         if calculateIntDistance((signTipX, signTipY), (thumbTipX, thumbTipY)) < 50:
101             centerX = int((signTipX + thumbTipX) / 2)
102             centerY = int((signTipY + thumbTipY) / 2)
103             cv2.line(frame, (signTipX, signTipY), (thumbTipX, thumbTipY), (0, 255, 0), 2)
104             cv2.circle(frame, (centerX, centerY), 5, (0, 255, 0), cv2.FILLED)
105
106     ctime = time.time()
107     fps = int(1 / (ctime - ptime))
108
109     cv2.putText(frame, str(fps) + " FPS", (10, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 0), 2)
110     showKey.drawKey(frame, (255, 255, 255), (0, 0, 0), 0.1, fontScale=0.5)

```

```

111 exitKey.drawKey(frame, (255, 255, 255), (0, 0, 0), 0.1, fontScale=0.5)
112 copyKey.drawKey(frame, (255, 255, 255), (0, 0, 0), 0.1, fontScale=0.5)
113 nextLineKey.drawKey(frame, (255, 255, 255), (0, 0, 0), 0.1, fontScale=0.5)
114
115 cv2.setMouseCallback('video', getMousPos)
116
117 if showKey.isOver(mouseX, mouseY):
118     showKey.drawKey(frame, (0, 255, 0), (0, 0, 0), 0.1, fontScale=0.5) # Change color when hovered
119 if exitKey.isOver(mouseX, mouseY):
120     exitKey.drawKey(frame, (0, 255, 0), (0, 0, 0), 0.1, fontScale=0.5) # Change color when hovered
121 if copyKey.isOver(mouseX, mouseY):
122     copyKey.drawKey(frame, (0, 255, 0), (0, 0, 0), 0.1, fontScale=0.5) # Change color when hovered
123 if nextLineKey.isOver(mouseX, mouseY):
124     nextLineKey.drawKey(frame, (0, 255, 0), (0, 0, 0), 0.1, fontScale=0.5) # Change color when hovered
125
126
127 if showKey.isOver(clickedX, clickedY):
128     show = not show
129     showKey.text = "Hide" if show else "Show"
130     clickedX, clickedY = 0, 0
131
132 if exitKey.isOver(clickedX, clickedY):
133     break
134
135 if copyKey.isOver(clickedX, clickedY):
136     pyperclip.copy(textBox.text)
137     print("Text copied to clipboard")
138     clickedX, clickedY = 0, 0
139
140 if nextLineKey.isOver(clickedX, clickedY):
141     textBox.text += "\n" # Add new line to the text box
142     clickedX, clickedY = 0, 0
143
144 alpha = 0.5
145 if show:
146     #textBox.drawKey(frame, (255, 255, 255), (0, 0, 0), 0.3)
147     # above to not to show centered text

```

```

149 #cv2.rectangle(frame,(25,170),(925,375),(128,128,128),2)
150 # Split text into lines based on newline characters
151 lines = textBox.text.split('\n')
152
153 # Render each line of text on the frame
154 line_y = textBox.y + textBox.h - 190
155 for line in lines:
156     cv2.putText(frame, line, (textBox.x + 10, line_y), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
157     line_y += 30
158
159 for k in keys:
160     if k.isOver(mouseX, mouseY) or k.isOver(signTipX, signTipY):
161         alpha = 0.1
162         if k.isOver(clickedX, clickedY):
163             if k.text == '<--':
164                 textBox.text = textBox.text[:-1]
165             elif k.text == 'clr':
166                 textBox.text = ''
167             elif len(textBox.text) < 999:
168                 if k.text == 'Space':
169                     textBox.text += " "
170                 else:
171                     textBox.text += k.text
172
173             if k.isOver(thumbTipX, thumbTipY):
174                 clickTime = time.time()
175                 if clickTime - previousClick > 0.4:
176                     if k.text == '<--':
177                         textBox.text = textBox.text[:-1]
178                     elif k.text == 'clr':
179                         textBox.text = ''
180                     elif len(textBox.text) < 30:
181                         if k.text == 'Space':
182                             textBox.text += " "
183                         else:
184                             textBox.text += k.text

```

```

185         keyboard.press(k.text)
186         previousClick = clickTime
187     k.drawKey(frame, (255, 255, 255), (0, 0, 0), alpha=alpha)
188     alpha = 0.5
189
190     if exitKey.isOver(signTipX, signTipY):
191         break
192
193     for control_key in superbuttons: # Only check the last four keys (control keys)
194         if control_key.isOver(signTipX, signTipY):
195             control_key.drawKey(frame, (0, 255, 0)) # Highlight control key in green
196             if control_key.text == 'Copy':
197                 current_time = time.time()
198                 if current_time - last_time >= delayo:
199                     pyperclip.copy(textBox.text)
200                     print("Text copied to clipboard")
201                     last_time = current_time
202             if control_key.text == 'Next line':
203                 current_time = time.time()
204                 if current_time - last_time >= delayo:
205                     textBox.text += '\n'
206                     last_time = current_time
207             if control_key.text == 'Show':
208                 current_time = time.time()
209                 if current_time - last_time >= delayo:
210                     show = not show
211                     showKey.text = "Hide" if show else "Show"
212                     signTipX, signTipY = 0, 0
213                     last_time = current_time
214
215             if control_key.text == 'Exit':
216                 print("Text copied to clipboard")
217                 break
218             '''current_time = time.time()
219             if current_time - last_time >= delayo:
220                 break # Exit the loop and terminate the program
221

```



```
222         last_time = current_time'''
223
224
225     clickedX, clickedY = 0, 0
226     ptime = ctime
227     cv2.imshow('video', frame)
228
229     # Check for key press to exit
230     pressedKey = cv2.waitKey(1)
231     if pressedKey == ord('q'):
232         break
233
234     cap.release()
235     cv2.destroyAllWindows()
236
237
```