

CURSOR MOVEMENT ON OBJECT MOTION

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Science Degree in Computer Science

By

NIRANCHANA.K (Reg. No. 39290072)

JEEVITHA.S (Reg. No. 38290040)



**DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF COMPUTING**

**SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY**

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU**

MARCH - 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE SCHOOL OF COMPUTING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **NIRANCHANA.K (Reg. No. 39290072)** and **JEEVITHA.S (Reg. No. 39290040)** who carried out the project entitled "**CURSOR MOVEMENT ON OBJECT MOTION**" under my supervision from to

Internal Guide

Dr. M. SELVI, M.E., Ph.D.,

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **NIRANCHANA.K (Reg. No. 39290072)** hereby declare that the Project Report entitled "**CURSOR MOVEMENT ON OBJECT MOTION**" done by me under the guidance of **Dr. M. SELVI, M.E, Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Science degree in Computer Science.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., Ph.D.**, Dean, School of Computing and **Dr. S. VIGNESHWARI, M.E., Ph.D.**, and **Dr. L. LAKSHMANAN, M.E., Ph.D.**, Head of the Department, Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. M. SELVI, M.E., Ph.D.**, for her valuable guidance, suggestions, and constant encouragement that paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Computer Science and Engineering who were helpful in many ways For the Completion of the project.

ABSTRACT

As modern technology of human-computer interactions become important in our everyday lives, varieties of mouse of all kinds of shapes and sizes were invented, from a casual office mouse to hard-core gaming mouse. However, there are some limitations to this hardware as they are not as environmentally friendly as it seems. For example, the physical mouse requires a flat surface to operate, not to mention that it requires a certain area to fully utilize the functions offered. Furthermore, some of this hardware is completely useless when it comes to interacting with the computers remotely due to the cable lengths limitations, rendering it inaccessible.

In the proposed AI virtual mouse system, the concept of advancing human-computer interaction using computer vision is given. Cross comparison of the testing of the AI virtual mouse system is difficult because only limited numbers of datasets are available. The hand gestures and fingertip detection have been tested in various illumination conditions and also been tested with different distances from the webcam for tracking of the hand gesture and hand tip detection.

TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	ABSTRACT	v
	LIST OF FIGURES	viii
1	INTRODUCTION	1
	1.1 OVERVIEW OF PROJECT	1
2	LITERATURE SURVEY	2
3	AIM AND SCOPE OF PRESENT INVESTIGATION	5
	3.1 AIM OF THE PROJECT	5
	3.2 SCOPE AND OBJECTIVE	5
	3.3 SYSTEM REQUIREMENTS	6
	3.3.1 HARDWARE REQUIREMENTS	6
	3.3.2 SOFTWARE REQUIREMENTS	6
	3.4 SOFTWARE USED	7
	3.4.1 PYTHON LANGUAGE	7
	3.4.2 PYTHON CHARACTERISTICS	7
	3.4.3 APPLICATIONS OF PYTHON	7
	3.4.4 OPENCV PACKAGE	8
	3.5 ANACONDA NAVIGATOR	9
	3.5.1 ANACONDA	9
	3.5.2 APPLICATIONS IN ANACONDA	11
	3.5.3 VS CODE	12
	3.5.4 NEW FEATURES OF ANACONDA 5.3	12

4	EXPERIMENTAL OR MATERIAL METHODS	14
	4.1 DESIGN METHODOLOGY	14
	4.1.1 EXISTING SYSTEM	14
	4.1.2 PROPOSED SYSTEM	14
	4.2 MODULE DESCRIPTION	14
	4.2.1 MEDIAPIPE FRAMEWORK	16
	4.2.2 OPENCV LIBRARY	20
	4.3 ARCHITECTURE DIAGRAM	21
	4.3.1 REAL-TIME VIDEO FOR WEB CAMERA	22
	4.3.2 CONVERTING VIDEO INTO IMAGES AND PROCESSING THEM	22
	4.3.3 EXTRACTION OF DIFFERENT COLOR FROM IMAGES	22
	4.3.4 PERFORMING DIFFERENT MOUSE ACTION BY COLOR POINTER	22
5	RESULTS AND PERFORMANCE ANALYSIS	23
	5.1 MOUSE MOVEMENT USING HANDGESTURE	23
	5.2 LEFT CLICK USING HANDGESTURE	24
	5.3 RIGHT CLICK USING HANDGESTURE	25
	5.4 DOUBLE CLICK USING HANDGESTURE	26
	5.5 BRIGHTNESS CONTROL, VOLUME CONTROL, AND SCROLL FUNCTION	27
	5.6 NO ACTION PERFORMED	28
6	CONCLUSION AND FUTURE ENHANCEMENT	29
	6.1 CONCLUSION	29
	6.2 FUTURE ENHANCEMENT	29
	REFERENCES	30
	APPENDIX	31
	A. SOURCE CODE	31

LIST OF FIGURES

FIGURE NO:	FIGURE NAME	PAGE NO
3.1	Anaconda Distribution	10
3.2	Anaconda Navigator Home Page	11
4.1	Mediapipe Framework	17
4.2	OpenCV library process videos to identify hand	20
4.3	Architecture Diagram	21
5.1	Mouse Movement using HandGesture	23
5.2	Left click Using Hand Gesture	24
5.2.1	Left Click clicks the Help Tool	24
5.3	Right Click Using Hand Gesture	25
5.3.1	Right Click performs like this	25
5.4	Double Click Using Hand Gesture	26
5.4.1	Double Click opens the Notepad	26
5.5	Brightness Control, Volume Control, And Scroll Function	27
5.6	No Action Performed Using Hand Gesture	28

CHAPTER 1

1. INTRODUCTION

1.1 OVERVIEW OF PROJECT

A virtual mouse is software that allows users to give mouse inputs to a system without using an actual mouse. To the extreme it can also be called as hardware because it uses an ordinary web camera. A virtual mouse can usually be operated with multiple input devices, which may include an actual mouse or a computer keyboard. Virtual mouse which uses web camera works with the help of different image processing techniques.

In this the hand movements of a user are mapped into mouse inputs. A web camera is set to take images continuously. Most laptops today are equipped with webcams, which have recently been used in security applications utilizing face recognition. In order to harness the full potential of a webcam, it can be used for vision-based HCI, which would effectively eliminate the need for a computer mouse or mouse pad. The usefulness of a webcam can also be greatly extended to other HCI applications such as a sign language database or motion controller. Over the past decades there have been significant advancements in HCI technologies for gaming purposes, such as the Microsoft Kinect and Nintendo Wii. These gaming technologies provide a more natural and interactive means of playing videogames. Motion controls is the future of gaming and it have tremendously boosted the sales of video games, such as the Nintendo Wii which sold over 50 million consoles within a year of its release. HCI using hand gestures is very intuitive and effective for one-to-one interaction with computers and it provides a Natural User Interface (NUI). There has been extensive research towards novel devices and techniques for cursor control using hand gestures. Besides HCI, hand gesture recognition is also used in sign language recognition, which makes hand gesture recognition even more significant.

CHAPTER 2

2. LITERATURE SURVEY

1. Mouse Control using a Web Camera based on Color Detection

Authors: Abhik Banerjee, Abhirup Ghosh, Koustuvmoni Bharadwaj, IJCTT, Volume.9, Mar 2014

In this paper, an object tracking based virtual mouse application has been developed and implemented using a webcam. Applied in modern gaming consoles to create interactive games where a person's motions are tracked and interpreted as commands. The presence of other colored objects in the background might cause the system to give an erroneous response. We present an approach for Human computer Interaction (HCI), where we have tried to control the mouse cursor movement and click events of the mouse using hand gestures. Hand gestures were acquired using a camera based on color detection technique. This method mainly focuses on the use of a Web Camera to develop a virtual human computer interaction device in a cost-effective manner.

2. Real Time Static & Dynamic Hand Gesture Recognition

Authors: Angel, Neethu. P.S, International Journal of Scientific & Engineering Research Volume 4, Issue3, March-2013.

Hand gesture recognition system plays a vital role in our day-to-day life. This hand gesture recognition system provides us natural, innovative, user friendly way of interaction with the computer which is more familiar to us. Gesture recognition has a wide area of application including human machine interaction, sign language, game technology robotics etc. There are different static hand gestures defined these are representing one, two, three, four, five. There are different dynamic hand gestures defined these are representing waving hand, fist hand, vertical hand, and horizontal hand. We developed a simple and fast motion image-based algorithm. Gestures recognition deals with the goal of interpreting human gestures via mathematical algorithm. In general, it is suitable to control home appliances using hand gestures.

3. A Survey of Glove-Based Input

Authors: D. J. Sturman and D. Zeltzer, IEEE Computer Graphics and Applications, 14: 30-39, 1994.

The primary objective is to introduce sensor gloves to the non-specialist readers interested in selecting one of these devices for their particular application. In Design and Manufacturing, glove-based systems are used to interact with computer-generated (typically virtual reality) environments. Measurements taken with sensor gloves can be complemented with other types of measurements.

Clumsy intermediary devices constrain our interaction with computers and their applications. Glove-based input devices let us apply our manual dexterity to the task. We provide a basis for understanding the field by describing key hand-tracking technologies and applications using glove-based input. The bulk of development in glove-based input has taken place very recently, and not all of it is easily accessible in the literature. We present a cross-section of the field to date. Hand-tracking devices may use the following technologies: position tracking, optical tracking, marker systems, silhouette analysis, magnetic tracking or acoustic tracking. Actual glove technologies on the market include: Sayre glove, MIT LED glove, Digital Data Entry Glove, Data Glove, Dexterous HandMaster, Power Glove, CyberGlove and Space Glove. Various applications of glove technologies include projects into the pursuit of natural interfaces, systems for understanding signed languages, teleoperation and robotic control, computer-based puppetry, and musical performance.

4. A Real Time Hand Gesture Recognition System Using Motion History Image

Authors: Chen-Chiung Hsieh and Dung-Hua Liou, iccps, 2010.

Hand gesture recognition is based man-machine interface is being developed vigorously in recent years. Due to the effect of lighting and complex background, most visual hand gesture recognition systems work only under restricted environment.

An adaptive skin color model based on face detection is utilized to detect skin color regions like hands. To classify the dynamic hand gestures, we developed a simple and fast motion history image-based method. Four groups of hand-like directional patterns were trained for the up, down, left, right hand gestures classifiers. Together with fist hand and waving hand gestures, there were totally six hand gestures defined. In general, it is suitable to control most home appliances. Five persons doing 250 hand gestures at near, medium, and far distances in front of the web camera were tested. Experimental results show that the accuracy is 94.1% in average and the processing time is 3.81 ms per frame. These demonstrated the feasibility of the proposed system.

5. Multi-scale gesture recognition from time-varying contours

Authors: H. Li and M. Greesspan, Proc. IEEE International Conference on Computer Vision, 2005.

A novel method is introduced to recognize and estimate the scale of time-varying human gestures. It exploits the changes in contours along spatiotemporal directions. Each contour is first parameterized as a 2D function of radius vs. cumulative contour length, and a 3D surface is composed from a sequence of such functions. In a two-phase recognition process, dynamic time warping is employed to rule out significantly different gesture models, and then mutual information (MI) is applied for matching the remaining models. The system has been tested on 8 gestures performed by 5 subjects with varied time scales. The two-phase process is compared against exhaustively testing three similarity measures based upon MI, correlation, and nonparametric kernel density estimation. Experimental results demonstrate that the exhaustive application of MI is the most robust with a recognition rate of 90.6%, however, the two-phase approach is much more computationally efficient with a comparable recognition rate of 90.0%.

CHAPTER 3

3. AIM AND SCOPE OF PRESENT INVESTIGATION

3.1 AIM OF THE PROJECT

Hand gestures and hand tracking is an important task with many real world applications. For the purpose of detection of hand gestures and hand tracking, the MediaPipe framework is used, and OpenCV library is used for computer vision. The algorithm makes use of the machine learning concepts to track and recognize the hand gestures and hand tip.

3.2 SCOPE AND OBJECTIVE

There are generally two approaches for hand gesture recognition, which are hardware based (Quam 1990; Zhu et al 2006), where the user must wear a device, and the other is vision based (Shrivastava 2013; Wang and Popović 2009), which uses image processing techniques with inputs from a camera. The proposed system is vision based, which uses image processing techniques and inputs from a computer webcam. Vision based gesture recognition systems are generally broken down into four stages, skin detection, hand contour extraction, hand tracking and gesture recognition. The input frame would be captured from the webcam and the skin region would be detected using skin detection. The hand contour would then be found and used for hand tracking and gesture recognition. Hand tracking would be used to navigate the computer cursor and hand gestures would be used to perform mouse functions such as right click, left click, scroll up and scroll down. The scope of the project would therefore be to design a vision-based CC system, which can perform the mouse function previously stated.

3.3 SYSTEM REQUIREMENTS

3.3.1 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. The minimal hardware requirements are as follows,

1. Processor : Pentium IV
2. RAM :8GB
3. Processor : 2.4 GHz
4. Main Memory : 8GB RAM
5. Hard Disk Drive : 1tb
6. Web Camera

3.3.2 Software Requirements

Software requirements deals with defining resource requirements and prerequisites that needs to be installed on a computer to provide functioning of an application. The minimal software requirements are as follows,

1. Front end : python
2. IDE : anaconda
3. Operating System : Windows 10

3.4 SOFTWARE USED:

3.4.1 Python Language

Python is an object-oriented programming language created by Guido Rossum in 1989. It is ideally designed for rapid prototyping of complex applications. It has interfaces to many OS system calls and libraries and is extensible to C or C++. Many large companies use the Python programming language include NASA, Google, YouTube, BitTorrent, etc. Python programming is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and other advanced fields of Computer Science. Python had deep focus on code readability & this class will teach you python from basics.

3.4.2 Python Characteristics

- It provides rich data types and easier to read syntax than any other programming languages
- It is a platform independent scripted language with full access to operating system API's
- Compared to other programming languages, it allows more run-time flexibility
- It includes the basic text manipulation facilities of Perl and Awk
- A module in Python may have one or more classes and free functions
- Libraries in Pythons are cross-platform compatible with Linux, Macintosh, and Windows
- For building large applications, Python can be compiled to byte-code
- Python supports functional and structured programming as well as OOP
- It supports interactive mode that allows interacting Testing and debugging of snippets of code
- In Python, since there is no compilation step, editing, debugging and testing is fast.

3.4.3 Applications of Python

Programming Web Applications:

You can create scalable Web Apps using frameworks and CMS (Content Management System) that are built on Python. Some of the popular platforms for

creating Web Apps are: Django, Flask, Pyramid, Plone, Django CMS. Sites like Mozilla, Reddit, Instagram and PBS are written in Python.

Scientific and Numeric Computing:

There are numerous libraries available in Python for scientific and numeric computing. There are libraries like: SciPy and NumPy that are used in general purpose computing. And, there are specific libraries like: Earthy for earth science, Astray for Astronomy and so on. Also, the language is heavily used in machine learning, data mining and deep learning.

Creating software Prototypes:

Python is slow compared to compiled languages like C++ and Java. It might not be a good choice if resources are limited and efficiency is a must. However, Python is a great language for creating prototypes. For example: You can use Pygmy (library for creating games) to create your game's prototype first. If you like the prototype, you can use language like C++ to create the actual game.

Good Language to Teach Programming:

Python is used by many companies to teach programming to kids and newbies. It is a good language with a lot of features and capabilities. Yet, it's one of the easiest languages to learn because of its simple easy-to-use syntax.

3.4.4 OpenCV Package

Python is a general-purpose programming language started by Guido van Rossum, which became very popular in short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability.

Compared to other languages like C/C++, Python is slower. But another important feature of Python is that it can be easily extended with C/C++. This feature helps us to write computationally intensive codes in C/C++ and create a Python

wrapper for it so that we can use these wrappers as Python modules. This gives us two advantages: first, our code is as fast as original C/C++ code (since it is the actual C++ code working in background) and second, it is very easy to code in Python. This is how OpenCV-Python works, it is a Python wrapper around original C++ implementation.

And the support of NumPy makes the task easier. NumPy is a highly optimized library for numerical operations. It gives a MATLAB-style syntax. All the OpenCV array structures are converted to-and-from NumPy arrays. So whatever operations you can do in NumPy, you can combine it with OpenCV, which increases number of weapons in your arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this.

So OpenCV-Python is an appropriate tool for fast prototyping of computer vision problems.

3.5 ANACONDA NAVIGATOR:

3.5.1 Anaconda:

Anaconda is a free and open source, easy to install distribution of Python and R programming languages. Anaconda provides a working environment which is used for scientific computing, data science, statistical analysis and machine learning.

The latest distribution of Anaconda is Anaconda 5.3 and is released in October, 2018. It has the conda package, environment manager and a collection of 1000+ open source packages long with free community support.

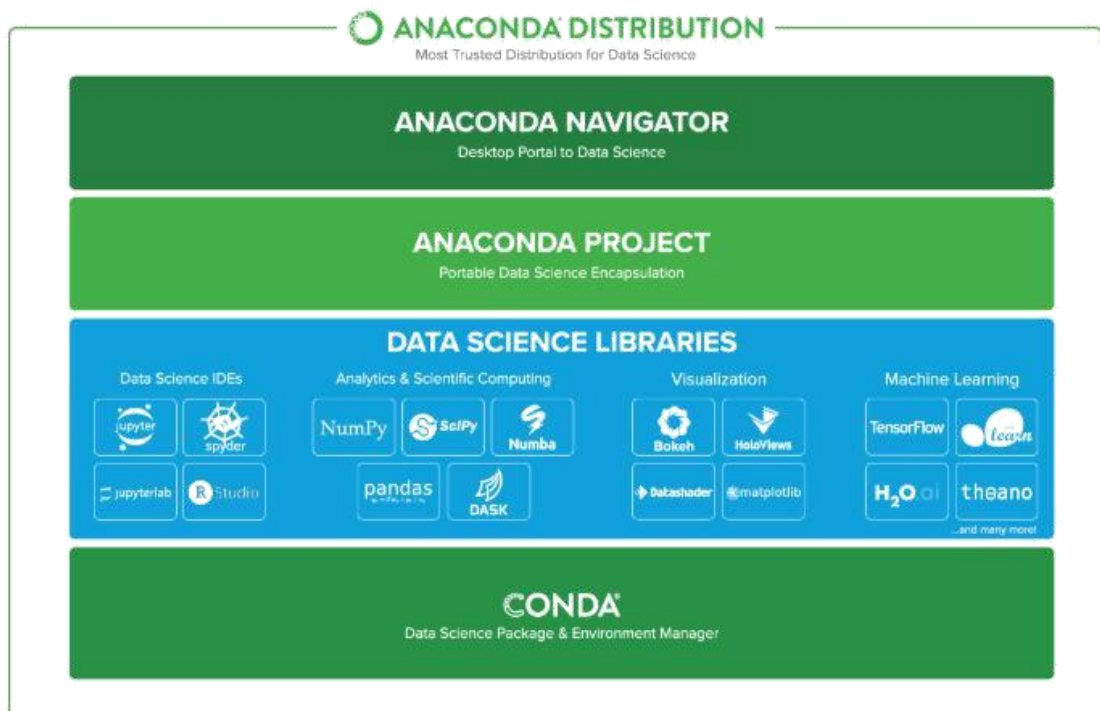


Fig 3.1: Anaconda Distribution

What is Anaconda Navigator?

Anaconda Navigator is a desktop graphical user interface (GUI) included in the Anaconda distribution. It allows us to launch applications provided in the Anaconda distribution and easily manage conda packages, environments and channels without the use of command-line commands. It is available for Windows, macOS and Linux.

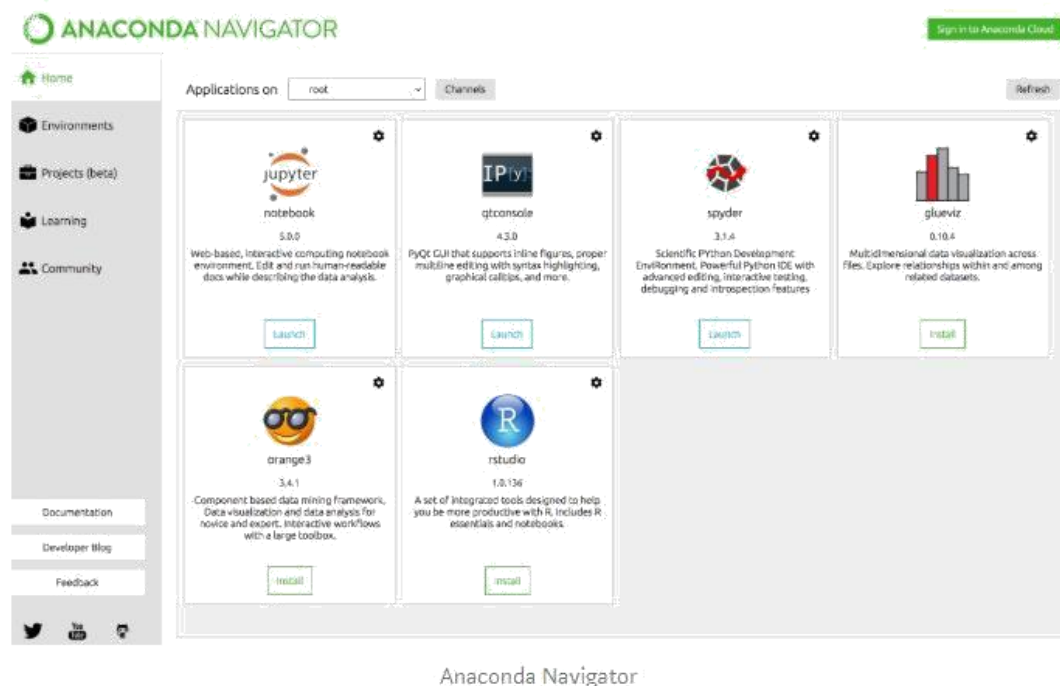


Fig 3.2: Anaconda Navigator Home Page

3.5.2 Applications Of Anaconda

The Anaconda distribution comes with the following applications along with Anaconda Navigator.

1. JupyterLab
2. Jupyter Notebook
3. Qt Console
4. Spyder
5. Glueviz
6. Orange3
7. RStudio

8. Visual Studio Code

3.5.3 VS Code:

VS Code is a free, open source streamlined cross-platform code editor with excellent support for Python code editing, IntelliSense, debugging, linting, version control, and more. Additionally, the Python Extension for Visual Studio Code tailors VS Code into a Python IDE. It is a streamlined code editor with support for development operations like debugging, task running and version control.

VS Code is free for both private and commercial use, runs on Windows, macOS, and Linux, and includes support for linting, debugging, task running, version control and Git integration, IntelliSense code completion, and conda environments. VS Code is openly extensible and many extensions are available.

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs, such as Visual Studio IDE.

3.5.4 New Features of Anaconda 5.3

Compiled with Latest Python release: Anaconda 5.3 is compiled with Python 3.7, taking advantage of Python's speed and feature improvements.

- **Better Reliability:**

The reliability of Anaconda has been improved in the latest release by capturing and storing the package metadata for installed packages.

- **Enhanced CPU Performance:**

The Intel Math Kernel Library 2019 for Deep Neural Networks(MKL 2019) has been introduced in Anaconda 5.3 distribution. Users deploying

TensorFlow can make use of MKL 2019 for Deep Neural Networks. These Python binary packages are provided to achieve high CPU performance.

- **New packages are added:**

There are over 230 packages which has been updated and added in the new release.

- **Work in Progress:**

There is a casting bug in Numpy with Python 3.7 but the team is currently working on patching it until Numpy is updated

CHAPTER 4

4. EXPERIMENTAL OR MATERIAL METHODS

4.1 DESIGN METHODOLOGY

4.1.1 Existing System:

The existing virtual mouse control system consists of simple mouse operations using a hand recognition system, in which we can control the mouse pointer, left click, right click, and drag, and so on. The use of hand recognition in the future will not be used. Even though there are a variety of systems for hand recognition, the system they used is static hand recognition, which is simply a recognition of the shape made by the hand and the definition of action for each shape made, which is limited to a few defined actions and causes a lot of confusion.

4.1.2 Proposed System:

In the proposed system, the concept of advancing the human-computer interaction using computer vision is given. The accuracy is very good and high for all the gestures except the scroll function. Compared to previous approaches for virtual mouse, our model system worked very well with 97% accuracy.

4.2 MODULE DESCRIPTION

Module 1: Camera Used in the AI Virtual Mouse System

The proposed AI virtual mouse system is based on the frames that have been captured by the webcam in a laptop or PC. By using the Python computer vision library OpenCV, the video capture object is created and the web camera will start capturing video. The web camera captures and passes the frames to the AI virtual system.

Module 2: Capturing the Video and Processing

The AI virtual mouse system uses the webcam where each frame is captured till the termination of the program. The video frames are processed from BGR to RGB color space to find the hands in the video frame by frame as shown in the following code:

```
def findHands(self, img , draw = True):  
    imgRGB = cv2.cvtColor(img , cv2.COLOR_BGR2RGB)  
    self.results = self.hands.process(imgRGB)
```

Module 3: Rectangular Region for Moving through the Window

The AI virtual mouse system makes use of the transformational algorithm, and it converts the coordinates of fingertip from the webcam screen to the computer window full screen for controlling the mouse. When the hands are detected and when we find which finger is up for performing the specific mouse function, a rectangular box is drawn with respect to the computer window in the webcam region where we move throughout the window using the mouse cursor.

Module 4: Detecting Which Finger Is Up and Performing the Particular Mouse Function.

In this stage, we are detecting which finger is up using the tip Id of the respective finger that we found using the MediaPipe and the respective co-ordinates of the fingers that are up, and according to that, the particular mouse function is performed.

Module 5: Mouse Functions Depending on the Hand Gestures and Hand Tip Detection Using Computer Vision For the Mouse Cursor Moving around the Computer Window

If the index finger is up with tip Id = 1 or both the index finger with tip Id = 1 and the middle finger with tip Id = 2 are up, the mouse cursor is made to move around the window of the computer using the AutoPy package of Python

Module 6: Model Creation:

Algorithm's Used:

- Mediapipe Framework
- OpenCV Library

4.2.1 Mediapipe Framework:

MediaPipe is a framework which is used for applying in a machine learning pipeline, and it is an opensource framework of Google. The MediaPipe framework is useful for cross platform development since the framework is built using the time series data. The MediaPipe framework is multimodal, where this framework can be applied to various audios and videos. The MediaPipe framework is used by the developer for building and analyzing the systems through graphs, and it also been used for developing the systems for the application purpose. The steps involved in the system that uses MediaPipe are carried out in the pipeline configuration. The pipeline created can run in various platforms allowing scalability in mobile and desktops. The MediaPipe framework is based on three fundamental parts; they are performance evaluation, framework for retrieving sensor data, and a collection of components which are called calculators, and they are reusable. A pipeline is a graph which consists of components called calculators, where each calculator is connected by streams in which the packets of data flow through. Developers are able to replace or define custom calculators anywhere in the graph creating their own application. The calculators and streams combined create a data-flow diagram; the graph is created with MediaPipe where each node is a calculator and the nodes are connected by streams.

Single-shot detector model is used for detecting and recognizing a hand or palm in real time. The single-shot detector model is used by the MediaPipe. First, in the hand detection module, it is first trained for a palm detection model because it is easier to train palms. Furthermore, the non-maximum suppression works significantly better on small objects such as palms or fists. A model of hand landmark consists of locating joint or knuckle co-ordinates in the hand region.

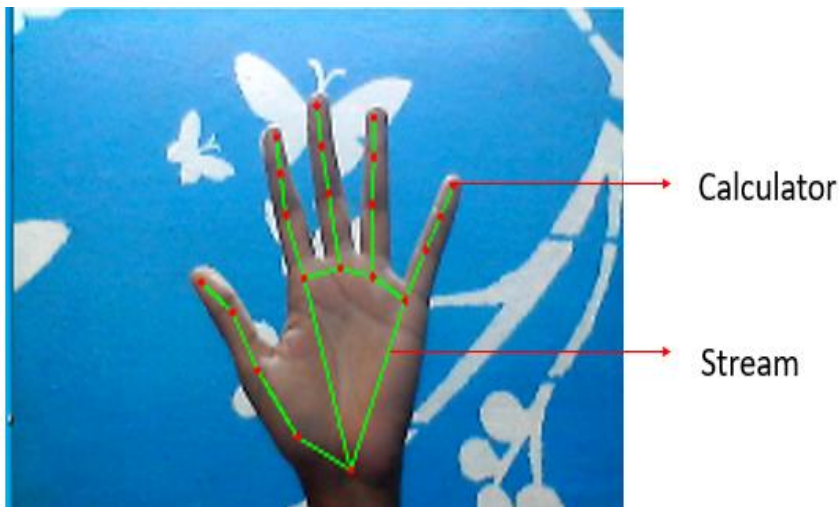


Fig 4.1: MediaPipe Framework

The followings are important concepts in MediaPipe:

The basics:

Packet

The basic data flow unit. A packet consists of a numeric timestamp and a shared pointer to an immutable payload. The payload can be of any C++ type, and the payload's type is also referred to as the type of the packet. Packets are value classes and can be copied cheaply. Each copy shares ownership of the payload, with reference-counting semantics. Each copy has its own timestamp. See also Packet.

Graph

MediaPipe processing takes place inside a graph, which defines packet flow paths between nodes. A graph can have any number of inputs and outputs, and data flow can branch and merge. Generally, data flows forward, but backward loops are possible.

Nodes

Nodes produce and/or consume packets, and they are where the bulk of the graph's work takes place. They are also known as "calculators", for historical reasons. Each node's interface defines a number of input and output ports, identified by a tag and/or an index.

Streams

A stream is a connection between two nodes that carries a sequence of packets, whose timestamps must be monotonically increasing.

Side packets

A side packet connection between nodes carries a single packet (with unspecified timestamp). It can be used to provide some data that will remain constant, whereas a stream represents a flow of data that changes over time.

Packet Ports

A port has an associated type; packets transiting through the port must be of that type. An output stream port can be connected to any number of input stream ports of the same type; each consumer receives a separate copy of the output packets, and has its own queue, so it can consume them at its own pace. Similarly, a side packet output port can be connected to as many side packet input ports as desired.

Input and output:

Data flow can originate from source nodes, which have no input streams and produce packets spontaneously (e.g. by reading from a file); or from graph input streams, which let an application feed packets into a graph.

Similarly, there are sink nodes that receive data and write it to various destinations (e.g. a file, a memory buffer, etc.), and an application can also receive output from the graph using callbacks.

Runtime behavior:

Graph lifetime

Once a graph has been initialized, it can be started to begin processing data, and can process a stream of packets until each stream is closed or the graph is canceled. Then the graph can be destroyed or started again.

Node lifetime

There are three main lifetime methods the framework will call on a node:

- Open: called once, before the other methods. When it is called, all input side packets required by the node will be available.
- Process: called multiple times, when a new set of inputs is available, according to the node's input policy.
- Close: called once, at the end.

In addition, each calculator can define constructor and destructor, which are useful for creating and deallocating resources that are independent of the processed data.

Input policies

The default input policy is deterministic collation of packets by timestamp. A node receives all inputs for the same timestamp at the same time, in an invocation of its Process method; and successive input sets are received in their timestamp order. This can require delaying the processing of some packets until a packet with the same timestamp is received on all input streams, or until it can be guaranteed that a packet with that timestamp will not be arriving on the streams that have not received it.

Other policies are also available, implemented using a separate kind of component known as an `InputStreamHandler`.

Real-time streams

MediaPipe calculator graphs are often used to process streams of video or audio frames for interactive applications. Normally, each Calculator runs as soon as all of its input packets for a given timestamp become available. Calculators used in real-time graphs need to define output timestamp bounds based on input timestamp bounds in order to allow downstream calculators to be scheduled promptly.

4.2.2 OpenCV Library:

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.



Fig 4.2: OpenCV library process videos to identify hand

Step 1:

To build this Hand Gesture Recognition project, we'll need four packages. So, first, import these.

```
# import necessary packages for hand gesture recognition project using Python OpenCV
import cv2
import numpy as np
import mediapipe as mp
import tensorflow as tf
from tensorflow.keras.models import load_model
```

Step 2:

Initialize models

Step 3:

Read frames from a webcam

3.1 – We create a Video Capture object and pass an argument '0'. It is the camera ID of the system. In this case, we have 1 webcam connected with the system. If you have multiple webcams then change the argument according to your camera ID. Otherwise, leave it default.

3.2 – The cap.read() function reads each frame from the webcam.

3.3 – cv2.flip() function flips the frame.

3.4 – cv2.imshow() shows frame on a new openCV window.

Step 4 – Detect hand keypoints

Step 5 – Recognize hand gestures

4.3 ARCHITECTURE DIAGRAM

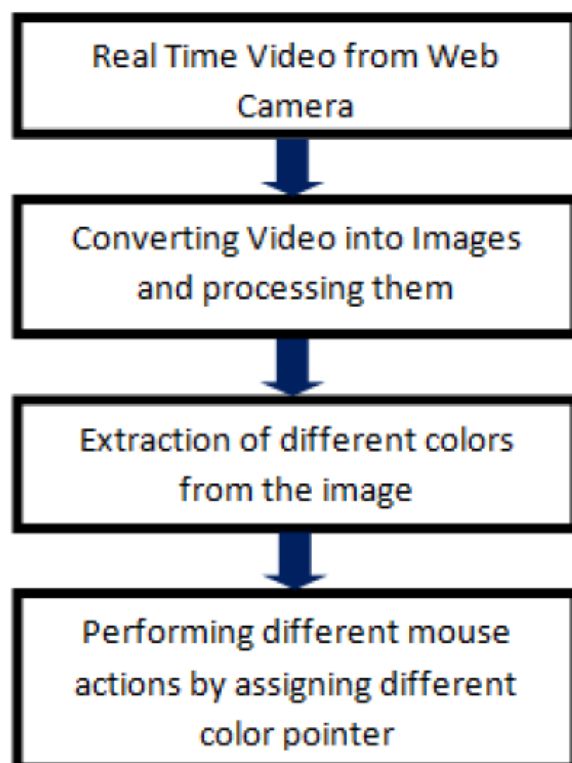


Fig 4.3: Architecture Diagram

4.3.1 Real-Time Video From Web Camera:

The proposed AI virtual mouse system is based on the frames that have been captured by the webcam on a laptop or PC. By using the Python computer vision library OpenCV, the video capture object is created and the web camera will start capturing video. The web camera captures and passes the frames to the AI virtual system.

4.3.2 Converting Video into Images and Processing them:

The AI virtual mouse system makes use of the transformational algorithm, and it converts the coordinates of the fingertip from the webcam screen to the computer window full screen for controlling the mouse. When the hands are detected and when we find which finger is up for performing the specific mouse function, a rectangular box is drawn with respect to the computer window in the webcam region where we move throughout the window using the mouse cursor.

4.3.3 Extraction of different colors from the image:

The AI virtual mouse system uses the webcam where each frame is captured till the termination of the program. The video frames are processed from BGR to RGB color space to find the hands in the video frame by frame as shown in the following code:

```
def findHands(self, img , draw = True):  
    imgRGB = cv2.cvtColor(img , cv2.COLOR_BGR2RGB)  
    self.results = self.hands.process(imgRGB)
```

4.3.4 Performing different mouse actions by assigning color pointers:

In this stage, we are detecting which finger is up using the tip Id of the respective finger that we found using the MediaPipe and the respective coordinates of the fingers that are up, and according to that, the particular mouse function is performed.

CHAPTER 5

5. RESULTS AND PERFORMANCE ANALYSIS

5.1. Mouse Movement Using Hand gesture:

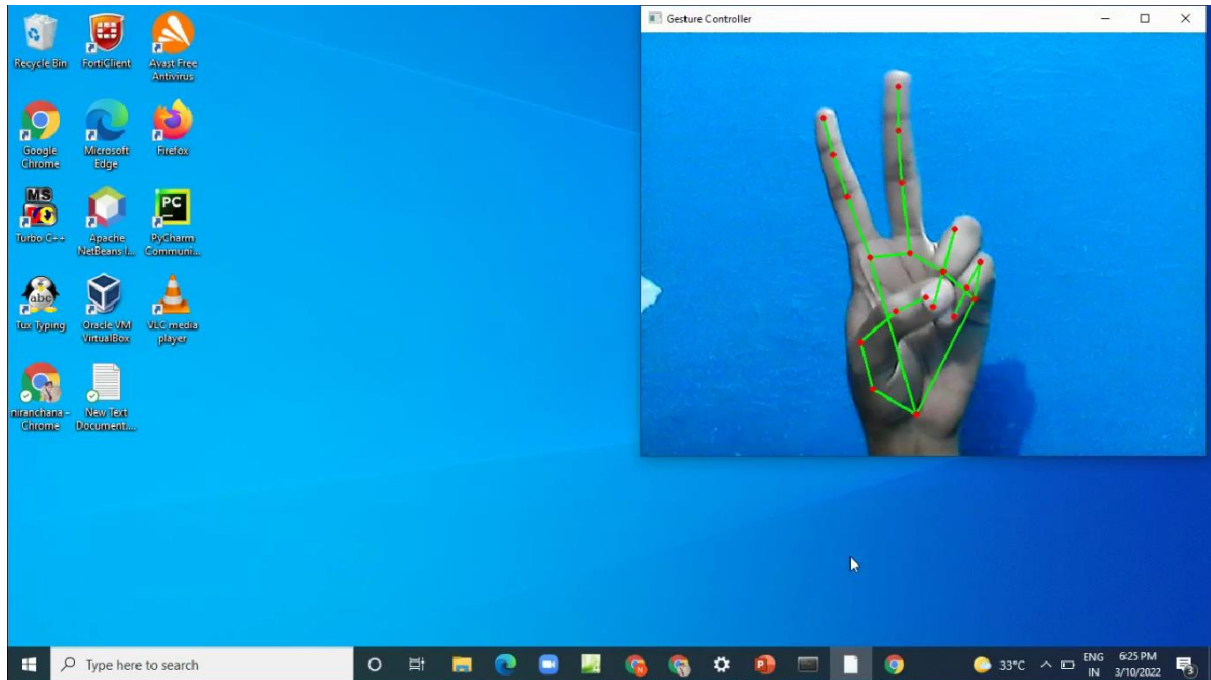


Fig 5.1: Mouse Movement Using Hand gesture

- This is a hand gesture that performs the action of a mouse movement.

5.2. Left Click Using Hand gesture:

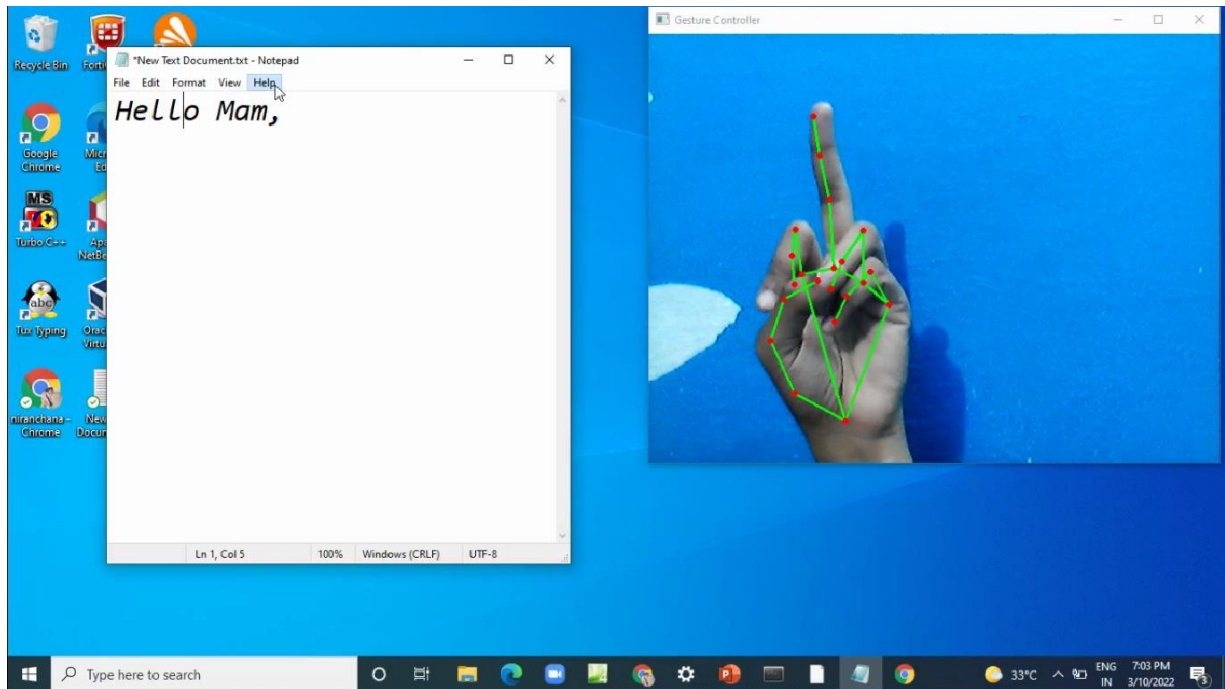


Fig 5.2: Left Click Using Hand gesture

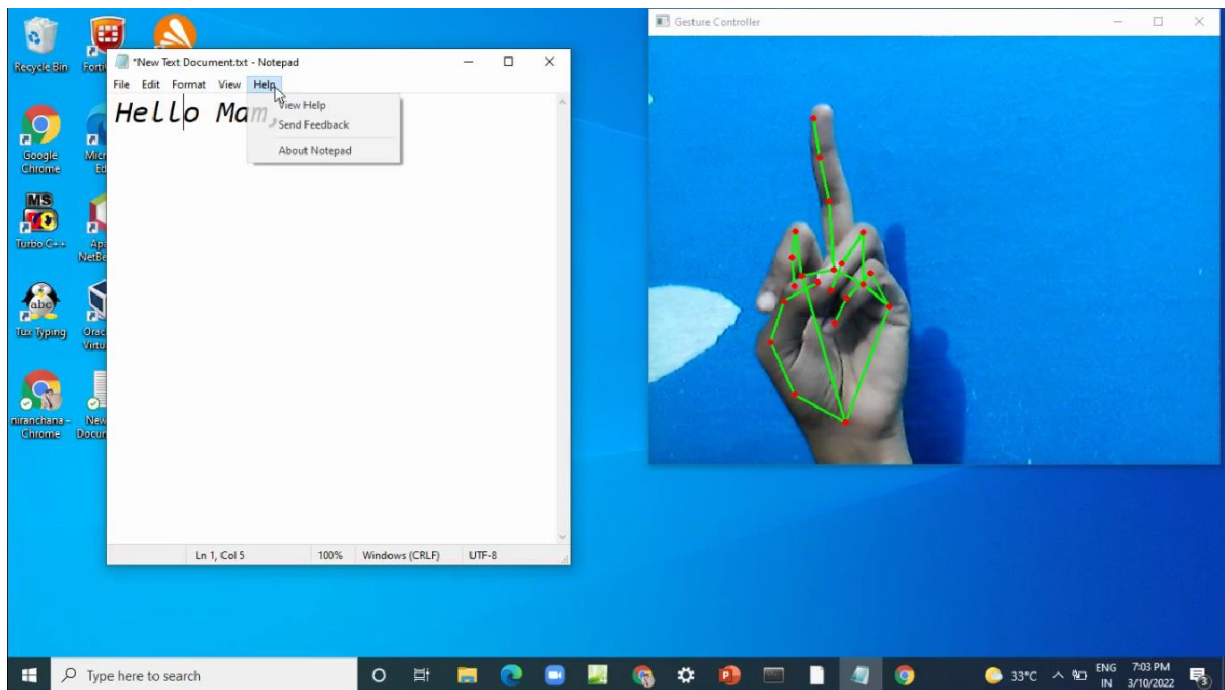


Fig 5.2.1: Left Click clicks the Help tool

- This is a hand gesture that performs the action of a left-click mouse movement and acts as a single click.

5.3. Right Click Using Hand gesture:

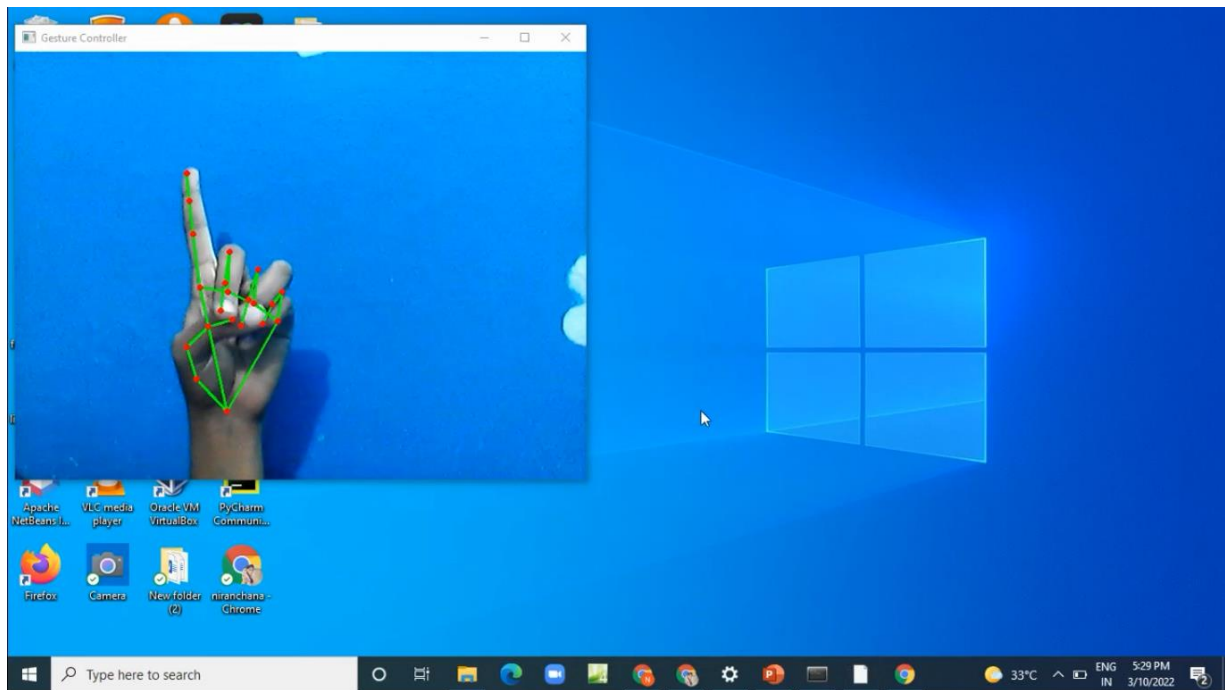


Fig 5.3: Right Click Using Hand gesture

- This is a hand gesture that performs the action of a Right click mouse movement.

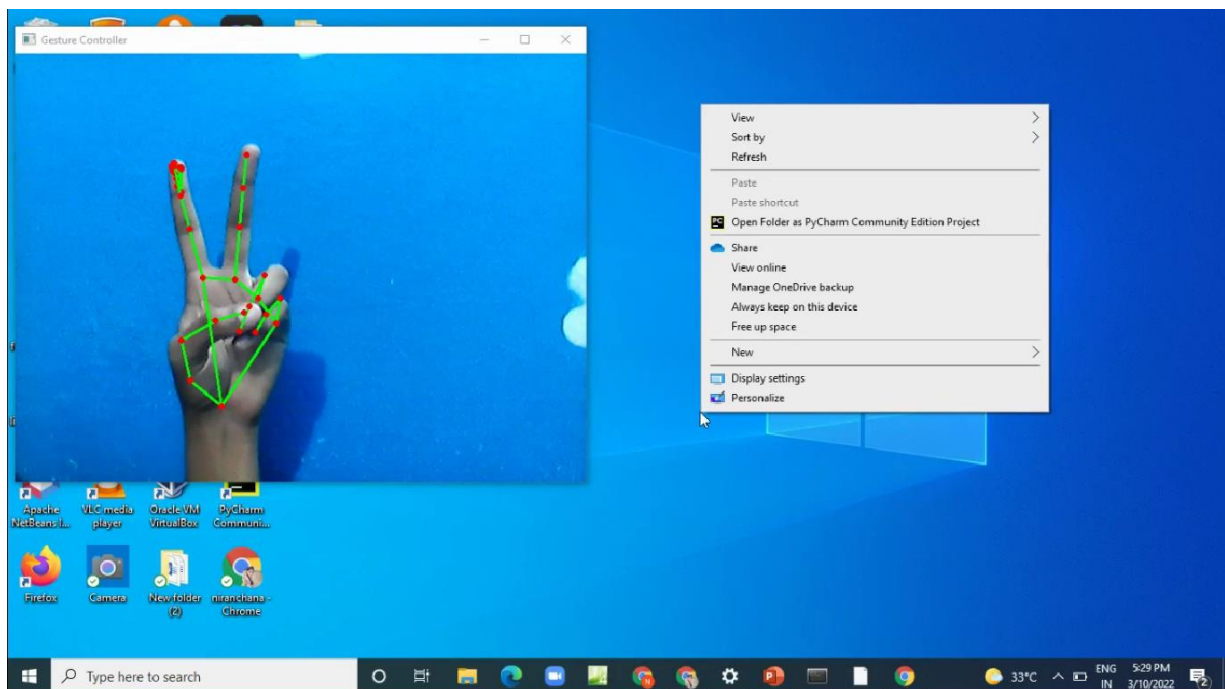


Fig 5.3.1: Right Click performs like this

5.4. Double Click Using Hand gesture:

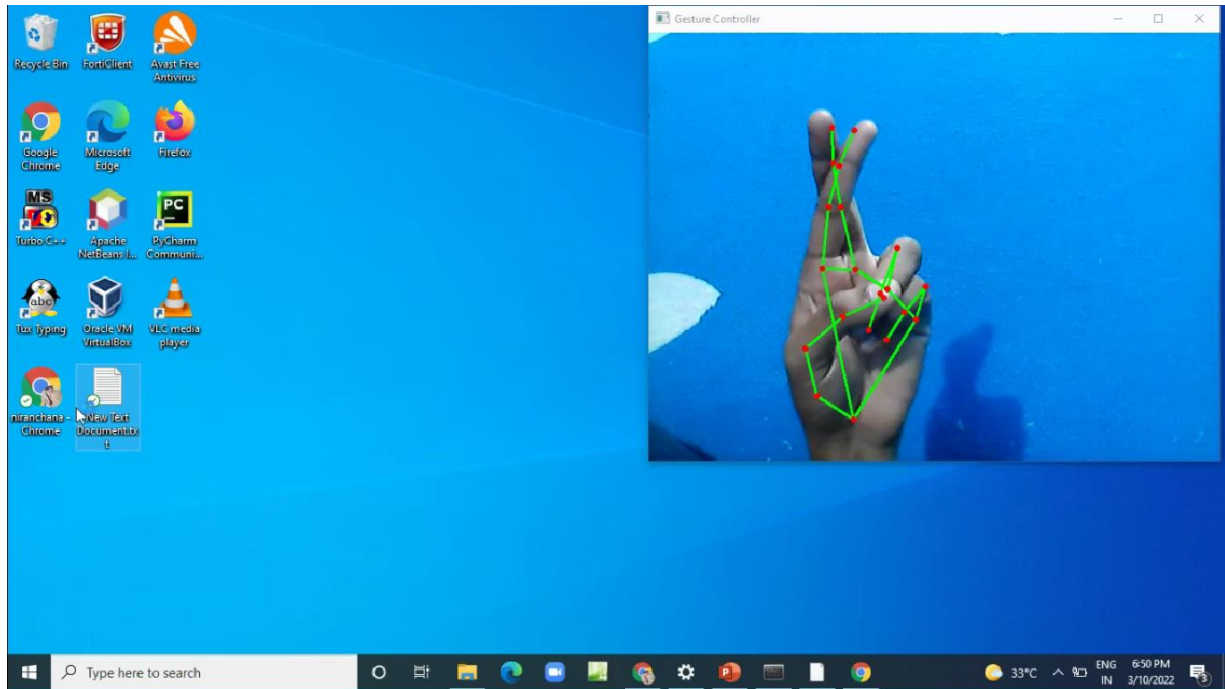


Fig 5.4: Double Click Using Hand gesture

- This is a hand gesture that performs the action of a double click mouse movement.

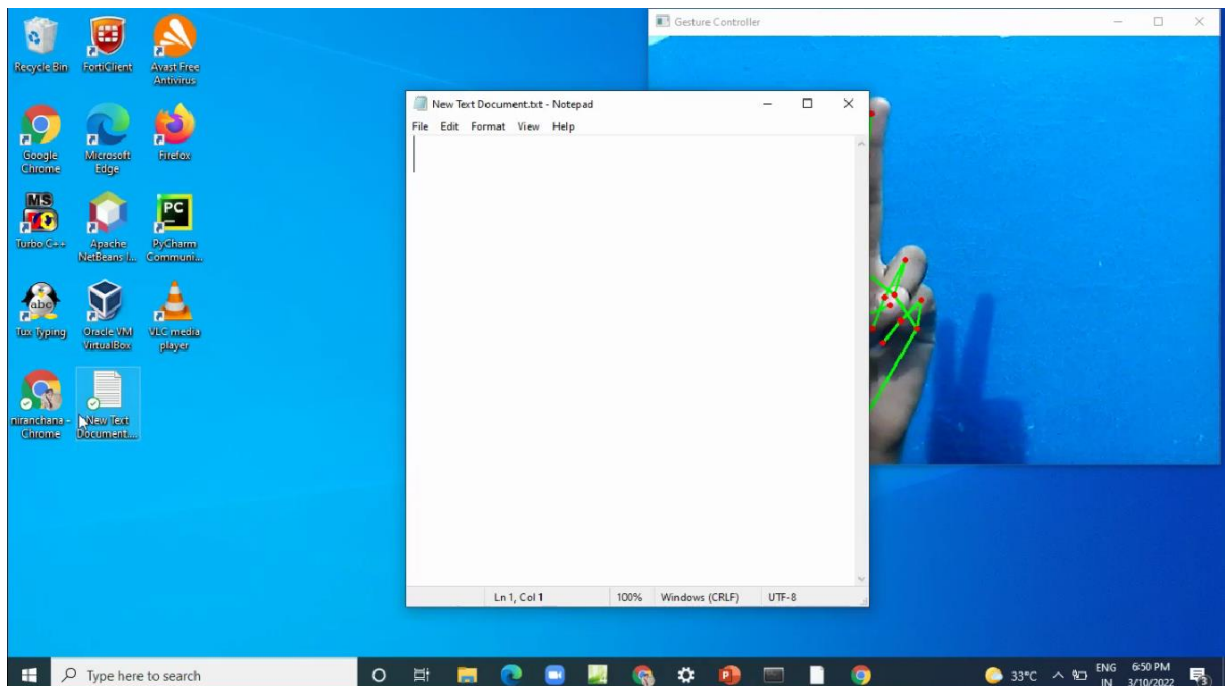


Fig 5.4.1: Double Click Opens the Notepad

5.5. Brightness Control, Volume Control, and Scroll Function Using Hand gesture:

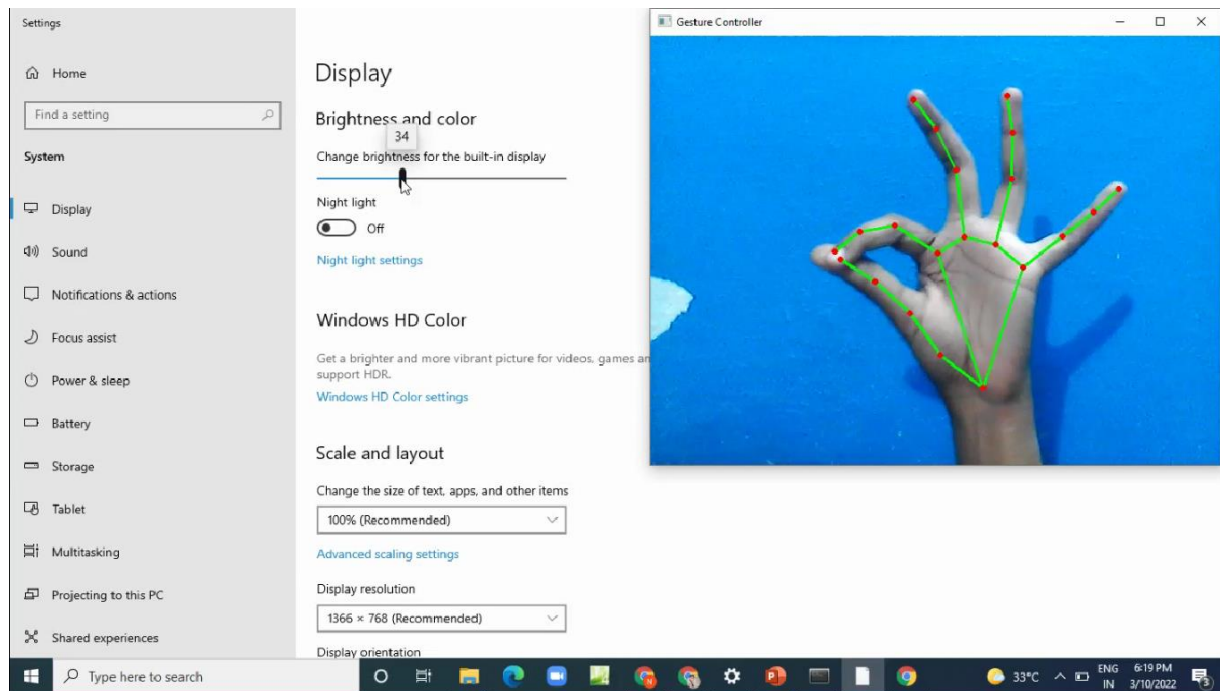


Fig 5.5: Brightness Control, Volume Control, and Scroll Function

- This hand gesture is common for all three functions that are Brightness Control, Volume Control, and Scroll Function.

5.6. No Action Performed Using Hand gesture:

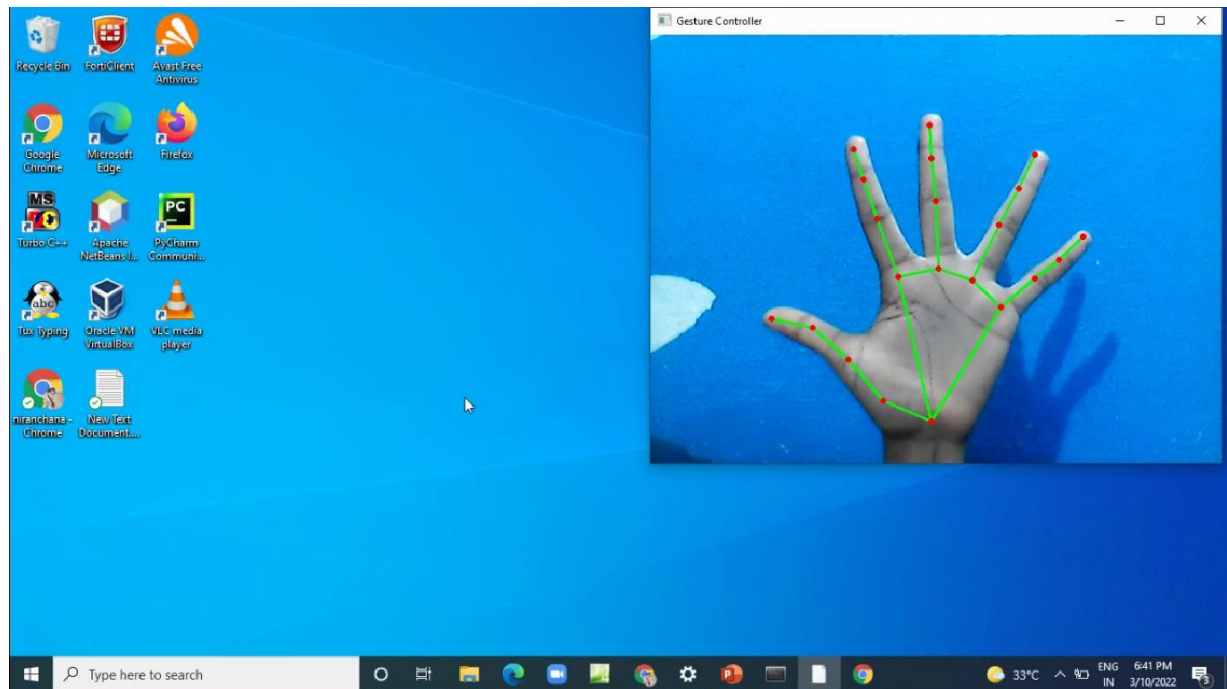


Fig 5.6: No action performed Using Hand gesture

- This is a hand gesture that performs nothing.

CHAPTER 6

6. CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION:

Due to accuracy and efficiency plays an important role in making the program as useful as an actual physical mouse, a few techniques had to be implemented. After implanting such type of application there is big replacement of physical mouse i.e., there is no need of any physical mouse. Each & every movement of physical mouse is done with this motion tracking mouse (virtual mouse).

6.2 FUTURE ENHANCEMENTS:

There are several features and improvements needed in order for the program to be more user friendly, accurate, and flexible in various environments. The following describes the improvements and the features required:

- a) Smart Movement: Due to the current recognition process are limited within 25cm radius, an adaptive zoom in/out functions are required to improve the covered distance, where it can automatically adjust the focus rate based on the distance between the users and the webcam.
- b) Better Accuracy & Performance: The response time are heavily relying on the hardware of the machine, this includes the processing speed of the processor, the size of the available RAM, and the available features of webcam. Therefore, the program may have better performance when it's running on a decent machine with a webcam that performs better in different types of lightings.
- c) Mobile Application: In future this web application also able to use on Android devices, where touchscreen concept is replaced by hand gestures.

REFERENCES

- [1] Abhik Banerjee, Abhirup Ghosh, Koustuvmoni Bharadwaj,” Mouse Control using a Web Camera based on Color Detection”, IJCTT, vol.9, Mar 2014
- [2] Angel, Neethu.P.S,”Real Time Static & Dynamic Hand Gesture Recognition”, International Journal of Scientific & Engineering Research Volume 4, Issue3, March-2013.
- [3] Gaurav Pradhan, Balakirshnan Prabhakaran, “Hand gesture computing”, IEEE Conference, 2011.
- [4] Chen-Chiung Hsieh and Dung-Hua Liou,” A Real Time Hand Gesture Recognition System Using Motion History Image”icsps, 2010.
- [5] H. Li and M. Greesspan, “Multi-scale gesture recognition from time-varying contours”, Proc. IEEE International Conference on Computer Vision, 2005.
- [6] C. Neti, G. Potamianos, J. Luetin, I. Matthews, H. Glotin, D. Vergyri, J. Sison, A. Mashari, and J. Zhou.Audio-“Visual Speech Recognition”, Workshop 2000 Final Report, 2000.
- [7] D. J. Sturman and D. Zeltzer, “A Survey of Glove-Based Input”, IEEE Computer Graphics and Applications, 14: 30-39, 1994.

APPENDIX

A. SOURCE CODE

MACHINE TRAINING CODE:

Imports

```
import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from google.protobuf.json_format import MessageToDict
import screen_brightness_control as sbcontrol
```

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

Gesture Encodings

```
class Gest(IntEnum):
```

```
    # Binary Encoded
```

```
    FIST = 0
```

```
    PINKY = 1
```

```
    RING = 2
```

```
    MID = 4
```

```
    LAST3 = 7
```

```
    INDEX = 8
```

```

FIRST2 = 12

LAST4 = 15
THUMB = 16
PALM = 31

# Extra Mappings
V_GEST = 33
TWO_FINGER_CLOSED = 34
PINCH_MAJOR = 35
PINCH_MINOR = 36

# Multi-handedness Labels
class HLabel(IntEnum):
    MINOR = 0
    MAJOR = 1

# Convert Mediapipe Landmarks to recognizable Gestures
class HandRecog:

    def __init__(self, hand_label):
        self.finger = 0
        self.ori_gesture = Gest.PALM
        self.prev_gesture = Gest.PALM
        self.frame_count = 0
        self.hand_result = None
        self.hand_label = hand_label

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

    def get_signed_dist(self, point):
        sign = -1
        if self.hand_result.landmark[point[0]].y < self.hand_result.landmark[point[1]].y:
            sign = 1

```

```

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

```

```

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

```

```

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

```

Function to find Gesture Encoding using current finger_state.

Finger_state: 1 if finger is open, else 0

```

def set_finger_state(self):
    if self.hand_result == None:
        return
    points = [[8,5,0],[12,9,0],[16,13,0],[20,17,0]]
    self.finger = 0
    self.finger = self.finger | 0 #thumb
    for idx,point in enumerate(points):

        dist = self.get_signed_dist(point[:2])
        dist2 = self.get_signed_dist(point[1:])

```

```

try:
    ratio = round(dist/dist2,1)
except:
    ratio = round(dist1/0.01,1)

self.finger = self.finger << 1
if ratio > 0.5 :
    self.finger = self.finger | 1

# Handling Fluctuations due to noise
def get_gesture(self):
    if self.hand_result == None:
        return Gest.PALM

    current_gesture = Gest.PALM
    if self.finger in [Gest.LAST3,Gest.LAST4] and self.get_dist([8,4]) < 0.05:
        if self.hand_label == HLabel.MINOR :
            current_gesture = Gest.PINCH_MINOR
        else:
            current_gesture = Gest.PINCH_MAJOR
# Executes commands according to detected gestures
class Controller:
    tx_old = 0
    ty_old = 0
    trial = True
    flag = False
    grabflag = False
    pinchmajorflag = False
    pinchminorflag = False
    pinchstartxcoord = None

```

```

pinchstartycoord = None
pinchdirectionflag = None
prevpinchlv = 0
pinchlv = 0
framecount = 0
prev_hand = None
pinch_threshold = 0.3

def getpinchylv(hand_result):
    dist = round((Controller.pinchstartycoord - hand_result.landmark[8].y)*10,1)
    return dist

def getpinchxlv(hand_result):
    dist = round((hand_result.landmark[8].x - Controller.pinchstartxcoord)*10,1)
    return dist

def changesystembrightness():
    currentBrightnessLv = sbcontrol.get_brightness()/100.0
    currentBrightnessLv += Controller.pinchlv/50.0
    if currentBrightnessLv > 1.0:
        currentBrightnessLv = 1.0
    elif currentBrightnessLv < 0.0:
        currentBrightnessLv = 0.0
    sbcontrol.fade_brightness(int(100*currentBrightnessLv) , start =
sbcontrol.get_brightness())

def changesystemvolume():
    devices = AudioUtilities.GetSpeakers()
    interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
    volume = cast(interface, POINTER(IAudioEndpointVolume))
    currentVolumeLv = volume.GetMasterVolumeLevelScalar()

```

```

currentVolumeLv += Controller.pinchlv/50.0
    if currentVolumeLv > 1.0:
        currentVolumeLv = 1.0
    elif currentVolumeLv < 0.0:
        currentVolumeLv = 0.0
    volume.SetMasterVolumeLevelScalar(currentVolumeLv, None)

def scrollVertical():
    pyautogui.scroll(120 if Controller.pinchlv>0.0 else -120)

def scrollHorizontal():
    pyautogui.keyDown('shift')
    pyautogui.keyDown('ctrl')
    pyautogui.scroll(-120 if Controller.pinchlv>0.0 else 120)
    pyautogui.keyUp('ctrl')
    pyautogui.keyUp('shift')

# Locate Hand to get Cursor Position
# Stabilize cursor by Dampening
def get_position(hand_result):
    point = 9
    position = [hand_result.landmark[point].x ,hand_result.landmark[point].y]
    sx,sy = pyautogui.size()
    x_old,y_old = pyautogui.position()
    x = int(position[0]*sx)
    y = int(position[1]*sy)
    if Controller.prev_hand is None:
        Controller.prev_hand = x,y
    delta_x = x - Controller.prev_hand[0]
    delta_y = y - Controller.prev_hand[1]

```

```

distsq = delta_x**2 + delta_y**2
ratio = 1
Controller.prev_hand = [x,y]

if distsq <= 25:
    ratio = 0
elif distsq <= 900:
    ratio = 0.07 * (distsq ** (1/2))
else:
    ratio = 2.1
x , y = x_old + delta_x*ratio , y_old + delta_y*ratio
return (x,y)

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

# Hold final position for 5 frames to change status
def pinch_control(hand_result, controlHorizontal, controlVertical):
    if Controller.framecount == 5:
        Controller.framecount = 0
        Controller.pinchlv = Controller.prevpinchlv

    if Controller.pinchdirectionflag == True:
        controlHorizontal() #x

    elif Controller.pinchdirectionflag == False:
        controlVertical() #y

```

```

lvx = Controller.getpinchxlv(hand_result)
lvy = Controller.getpinchylv(hand_result)

if abs(lvy) > abs(lvx) and abs(lvy) > Controller.pinch_threshold:
    Controller.pinchdirectionflag = False
    if abs(Controller.prevpinchlv - lvy) < Controller.pinch_threshold:
        Controller.framecount += 1
    else:
        Controller.prevpinchlv = lvy
        Controller.framecount = 0

elif abs(lvx) > Controller.pinch_threshold:
    Controller.pinchdirectionflag = True
    if abs(Controller.prevpinchlv - lvx) < Controller.pinch_threshold:
        Controller.framecount += 1
    else:
        Controller.prevpinchlv = lvx
        Controller.framecount = 0

def handle_controls(gesture, hand_result):
    x,y = None,None
    if gesture != Gest.PALM :
        x,y = Controller.get_position(hand_result)

    # flag reset
    if gesture != Gest.FIST and Controller.grabflag:
        Controller.grabflag = False
        pyautogui.mouseUp(button = "left")

    if gesture != Gest.PINCH_MAJOR and Controller.pinchmajorflag:
        Controller.pinchmajorflag = False

```

```

if gesture != Gest.PINCH_MINOR and Controller.pinchminorflag:
    Controller.pinchminorflag = False

# implementation
if gesture == Gest.V_GEST:
    Controller.flag = True
    pyautogui.moveTo(x, y, duration = 0.1)

elif gesture == Gest.FIST:
    if not Controller.grabflag :
        Controller.grabflag = True
        pyautogui.mouseDown(button = "left")
        pyautogui.moveTo(x, y, duration = 0.1)

elif gesture == Gest.MID and Controller.flag:
    pyautogui.click()
    Controller.flag = False

elif gesture == Gest.INDEX and Controller.flag:
    pyautogui.click(button='right')
    Controller.flag = False

elif gesture == Gest.TWO_FINGER_CLOSED and Controller.flag:
    pyautogui.doubleClick()
    Controller.flag = False

elif gesture == Gest.PINCH_MINOR:
    if Controller.pinchminorflag == False:
        Controller.pinch_control_init(hand_result)
        Controller.pinchminorflag = True
        Controller.pinch_control(hand_result,Controller.scrollHorizontal,
Controller.scrollVertical)

```

```

elif gesture == Gest.PINCH_MAJOR:
    if Controller.pinchmajorflag == False:
        Controller.pinch_control_init(hand_result)
        Controller.pinchmajorflag = True
        Controller.pinch_control(hand_result,Controller.changesystembrightness,
Controller.changesystemvolume)

```

```
'''
```

```
----- Main Class -----
```

```
    Entry point of Gesture Controller
```

```
'''
```

```
class GestureController:
```

```
    gc_mode = 0
```

```
    cap = None
```

```
    CAM_HEIGHT = None
```

```
    CAM_WIDTH = None
```

```
    hr_major = None # Right Hand by default
```

```
    hr_minor = None # Left hand by default
```

```
    dom_hand = True
```

```
    def __init__(self):
```

```
        GestureController.gc_mode = 1
```

```
        GestureController.cap = cv2.VideoCapture(0)
```

```
        GestureController.CAM_HEIGHT =
```

```
GestureController.cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
```

```
        GestureController.CAM_WIDTH =
```

```
GestureController.cap.get(cv2.CAP_PROP_FRAME_WIDTH)
```

```
    def classify_hands(results):
```

```
        left , right = None,None
```

```

try:
    handedness_dict = MessageToDict(results.multi_handedness[0])
    if handedness_dict['classification'][0]['label'] == 'Right':
        right = results.multi_hand_landmarks[0]
    else :
        left = results.multi_hand_landmarks[0]
except:
    pass

```

```

try:
    handedness_dict = MessageToDict(results.multi_handedness[1])
    if handedness_dict['classification'][0]['label'] == 'Right':
        right = results.multi_hand_landmarks[1]
    else :
        left = results.multi_hand_landmarks[1]
except:
    pass

```

```

if GestureController.dom_hand == True:
    GestureController.hr_major = right
    GestureController.hr_minor = left
else :
    GestureController.hr_major = left
    GestureController.hr_minor = right

```

```

def start(self):

```

```

    handmajor = HandRecog(HLabel.MAJOR)
    handminor = HandRecog(HLabel.MINOR)

```

```

    with mp_hands.Hands(max_num_hands = 2,min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:

```

```

        while GestureController.cap.isOpened() and GestureController.gc_mode:
            success, image = GestureController.cap.read()

```

```

if not success:
    print("Ignoring empty camera frame.")
    continue

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

image.flags.writeable = True
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

if results.multi_hand_landmarks:
    GestureController.classify_hands(results)
    handmajor.update_hand_result(GestureController.hr_major)
    handminor.update_hand_result(GestureController.hr_minor)

    handmajor.set_finger_state()
    handminor.set_finger_state()
    gest_name = handminor.get_gesture()

    if gest_name == Gest.PINCH_MINOR:
        Controller.handle_controls(gest_name, handminor.hand_result)
    else:
        gest_name = handmajor.get_gesture()
        Controller.handle_controls(gest_name, handmajor.hand_result)

    for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(image, hand_landmarks,
mp_hands.HAND_CONNECTIONS)
    else:
        Controller.prev_hand = None
        cv2.imshow('Gesture Controller', image)
        if cv2.waitKey(5) & 0xFF == 13:
            break

```

```
GestureController.cap.release()
cv2.destroyAllWindows()

# uncomment to run directly
# gc1 = GestureController()
# gc1.start()
```