

**AI Mini project**

**AI Virtual Mouse**

Akshansh Sharma (RA2011031010010)

Pratham Saini (RA2011031010024)

Parthiv Sen (RA2011031010035)

## **INDEX**

Sr No.	Topic
1	Objective
2	Introduction
3	LITERATURE SURVEY
4	Situation
5	Limitation and future enhancement
6	Algorithm and Techniques used
7.	System Development
7	Code and Screen Shot
7	CONCLUSIONS and future works
8	REFERENCES

## **Objective**

The objective for making a virtual mouse in an AI project can vary depending on the specific application and context. like:-

1. **To provide an alternative input method for people with disabilities:** The virtual mouse can be designed to be controlled using various body movements, such as head or eye movements, to provide a way for people with limited mobility to interact with computers.
2. **To improve the user experience:** The virtual mouse can be designed to enhance the user experience by providing a more intuitive and natural way to interact with applications or devices, such as touchless interfaces in public spaces.
3. **To enhance productivity:** The virtual mouse can be designed to automate certain tasks or workflows, such as data entry or image editing, to improve efficiency and reduce errors.
4. **To enable new forms of human-machine interaction:** The virtual mouse can be designed to explore new ways of interacting with machines, such as using gestures or voice commands, that can be more efficient or intuitive than traditional input methods.

Overall, the objective of making a virtual mouse in an AI project is to create a tool that can improve human-machine interaction, increase accessibility, and enhance productivity.

### **Introduction**

A **virtual mouse** is a 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, including an actual mouse or a computer keyboard. The virtual mouse which uses web camera works with the help of different image processing techniques.

In this the hand movements of a user is 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 have tremendously boosted video game sales, 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.

The virtual mouse project uses machine learning algorithms and computer vision techniques to enable the computer to recognize and interpret user inputs accurately. It involves training the AI model to detect specific movements or commands and then mapping those inputs to various computer functions.

The applications of the virtual mouse project are numerous, ranging from enabling people with disabilities to access and control computers to creating touchless interfaces in public spaces. Additionally, this technology has the potential to enable new forms of human-machine interaction, leading to increased efficiency and productivity in various fields.

Overall, the virtual mouse project represents a significant step forward in the development of AI technology and has the potential to revolutionize the way we interact with computers.

### **Literature survey**

As modern technology of human computer interactions become important in our everyday lives, varieties of mouse with all kind of shapes and sizes were invented, from a casual office mouse to a hard-core gaming mouse. However, there are some limitations to these hardware as they are not as environmental 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 these hardware are completely useless when it comes to interact with the computers remotely due to the cable lengths limitations, rendering it inaccessible.

Multi-point Interactive Whiteboards are available using the Wiimote [4]. The components used are IR pen, computer with Windows XP (installed with Microsoft .NET framework, the Wiimote Connect program and the Wiimote Whiteboard software), wiimote controller, a beamer capable of a 1024 x 786 pixel resolution. Here the wiimote controller tracks the infra-red source on the white board and sends info to PC via Bluetooth. The teaching platform comprises of a Wii-mote-based multi-touch teaching station, a Wii-mote-based interactive whiteboard and a Wii-mote-based stylus input conversion tool [5]. According to the literature survey, most people have used the Wii-mote to configure it as a virtual marker.

Multi-point Interactive Whiteboards are available using the Wiimote [4]. The components used are IR pen, computer with

Windows XP (installed with Microsoft .NET framework, the Wiimote Connect program and the Wiimote Whiteboard software), wiimote controller, a beamer capable of a 1024 x 786 pixel resolution. Here the wiimote controller tracks the infra-red source on the white board and sends info to PC via Bluetooth. The teaching platform comprises of a Wii-mote-based multi-touch teaching station, a Wii-mote-based interactive whiteboard and a Wii-mote-based stylus input conversion tool [5]. According to the literature survey, most people have used the Wii-mote to configure it as a virtual marker.

The current system is comprised of a generic mouse and trackpad monitor control system, as well as the absence of a hand gesture control system. The use of a hand gesture to access the monitor screen from a distance is not possible. Even though it is primarily attempting to implement, the scope is simply limited in the virtual mouse field. 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. As technology advances, there are more and more alternatives to using a mouse.

A special sensor (or built-in webcam) can track head movement to move the mouse pointer around on the screen. In the absence of a mouse button, the software's dwell delay feature is usually used. Clicking can also be accomplished with a well-placed switch.

## **Situation**

The situation of using virtual mouse exists due to several reasons:

1. **Accessibility:** Some individuals may have physical disabilities that prevent them from using a traditional mouse. Virtual mouse technology allows them to control a cursor on a computer screen using hand gestures or other body movements.
2. **Convenience:** Virtual mouse technology can be more convenient for some users, as it eliminates the need for a physical mouse or touchpad. This can be particularly useful for individuals who are traveling or working in a confined space.
3. **Health concerns:** Prolonged use of traditional mouse devices can lead to repetitive strain injuries (RSI), which can cause pain and discomfort in the hands and wrists. Virtual mouse technology can provide an alternative that reduces the risk of developing RSI.
4. **Novelty:** Virtual mouse technology is still a relatively new field, and some individuals may be interested in exploring its capabilities and potential applications.
5. **Innovation:** The development of virtual mouse technology is part of the broader trend of technological innovation, as researchers and engineers continue to explore new ways of interacting with digital devices.



Overall, the situation of using virtual mouse technology exists due to a combination of practical, health-related, and technological factors. As virtual mouse technology continues to evolve and improve, it is likely to become an increasingly popular and important tool for a wide range of users.

### **Limitation**

Without knowing the specific details of the virtual mouse project in question, it's difficult to provide a comprehensive list of limitations. However, here are some general limitations that could apply:

1. Limited functionality: Depending on the technology used and the goals of the project, the virtual mouse may not be able to replicate all the features and capabilities of a physical mouse. For example, it may not be able to scroll or click with the same precision.
2. Hardware limitations: The virtual mouse may require specific hardware components or sensors to function properly, which could limit its compatibility with certain devices or operating systems.
3. Dependence on software: The virtual mouse may require specialized software to be installed and configured, which could be a barrier for some users or systems.
4. User training: Users may need to be trained on how to use the virtual mouse, which could be time-consuming or difficult for some people.
5. Accessibility limitations: Some users with disabilities or mobility impairments may not be able to use the virtual mouse effectively, which could limit its overall usefulness.
6. Cost: Depending on the complexity and features of the virtual mouse project, it could be expensive to develop and implement, which could limit its availability to certain users or organizations.

## **Future Enhancement-**

1. Volume control option
2. Brightness control option
3. More fluent and easy to control movements

### **Algorithm and techniques used**

There are many AI algorithms that can be used to implement a virtual mouse. The choice of algorithm would depend on the specific requirements and context of the virtual mouse application. Here are a few examples of AI algorithms that could be used:

1. **Reinforcement Learning:** This algorithm can be used to train the virtual mouse to learn how to move and interact with its environment based on rewards and punishments. The virtual mouse would be rewarded when it moves towards the target and punished when it moves away from the target.
2. **Neural Networks:** Neural networks can be used to train the virtual mouse to recognize and classify different objects in its environment, such as the target object it needs to move towards.
3. **Genetic Algorithms:** Genetic algorithms can be used to optimize the behavior of the virtual mouse. This algorithm can be used to evolve the behavior of the mouse over time to achieve the best possible results.
4. **Decision Trees:** Decision trees can be used to implement a set of rules that the virtual mouse follows to make decisions about its movement and interactions with the environment.

Overall, the choice of AI algorithm depends on the specific requirements and context of the virtual mouse application, as well as the available data and resources.

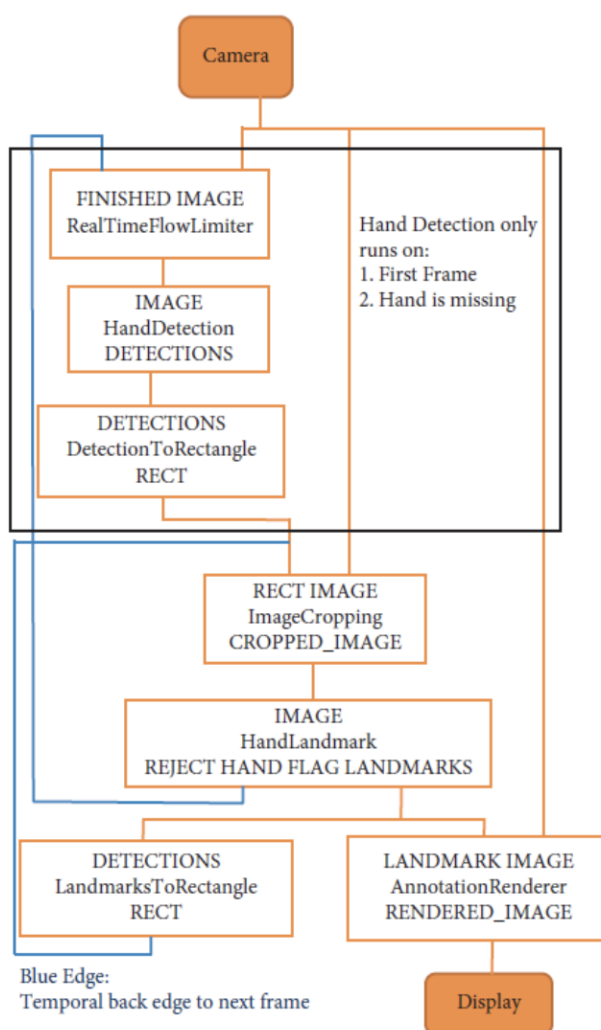
For the purpose of detection of hand gestures and hand tracking, the MediaPipe framework is used, and OpenCV library is used for computer vision. The algorithm makes use of the machine learning concepts to track and recognize the hand gestures and hand tip.

**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,

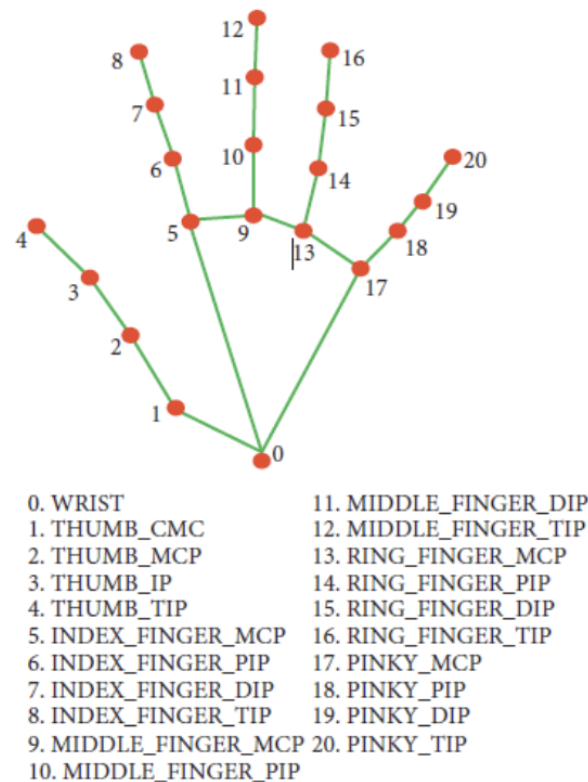
**OpenCV** is a computer vision library which contains image-processing algorithms for object detection. OpenCV is a library of python programming language, and real-time computer vision applications can be developed by using the computer vision library. The OpenCV library is used in image and video processing and also analysis such as face detection and object detection .



## **System Development**

The various functions and conditions used in the system are explained in the flowchart of the real-time AI virtual mouse system in figure.

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, as shown in Figure. The web camera captures and passes the frames to the AI virtual system.

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 colour 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)
```

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.

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.



## **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(dist/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

elif Gest.FIRST2 == self.finger:

    point = [[8, 12], [5, 9]]

    dist1 = self.get_dist(point[0])

    dist2 = self.get_dist(point[1])

    ratio = dist1/dist2

    if ratio > 1.7:

        current_gesture = Gest.V_GEST

    else:

        if self.get_dz([8, 12]) < 0.1:

            current_gesture = Gest.TWO_FINGER_CLOSED

```

```
else:
```

```
    current_gesture = Gest.MID
```

```
else:
```

```
    current_gesture = self.finger
```

```
if current_gesture == self.prev_gesture:
```

```
    self.frame_count += 1
```

```
else:
```

```
    self.frame_count = 0
```

```
self.prev_gesture = current_gesture
```

```
if self.frame_count > 4:
```

```
    self.ori_gesture = current_gesture
```

```
return self.ori_gesture
```

```
# 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 getpinchxlvl(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()

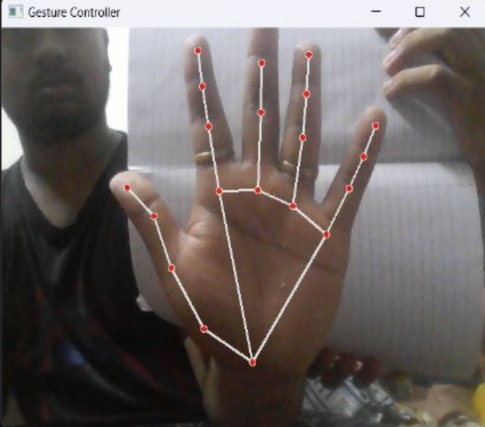
gc1.start()
```

## Screen Shot

test.py

```
292
293     # implementation
294     if gesture == Gest.V_GEST:
295         Controller.flag = True
296         pyautogui.moveTo(x, y, duration=0.1)
297
298     elif gesture == Gest.FIST:
299         if not Controller.grabflag:
300             Controller.grabflag = True
301             pyautogui.mouseDown(button="left")
302             pyautogui.moveTo(x, y, duration=0.1)
303
304     elif gesture == Gest.MID and Controller.flag:
305         pyautogui.click()
306         Controller.flag = False
307
308     elif gesture == Gest.INDEX and Controller.flag:
309         pyautogui.click(button="right")
310         Controller.flag = False
311
312     elif gesture == Gest.TWO_FINGER_CLOSED and Controller.flag:
313         pyautogui.doubleClick()
314         Controller.flag = False
315
316     elif gesture == Gest.PINCH_MINOR:
```

Gesture Controller



PROBLEMS

OUTPUT

TERMINAL

DEBUG CONSOLE

Python

+

□

✖

⋮

## **Conclusion and Future Scope**

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).

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.

## **Refernece**

- 1) OpenCV Website – [www.opencv.org](http://www.opencv.org)
- 2) MSDN Microsoft developers network – [www.msdn.microsoft.com](http://www.msdn.microsoft.com)
- 3) Code project –  
[www.codeproject.com/Articles/498193/Mouse-Control-via-Webcam](http://www.codeproject.com/Articles/498193/Mouse-Control-via-Webcam)
- 4) Aniket Tatipamula's Blog -  
<http://anikettatipamula.blogspot.in/2012/02/hand-gesture-using-opencv.html>
- 5) Microsoft Research Paper-  
<http://research.microsoft.com/en-us/um/people/awf/bmvc02/project.pdf>