A PROJECT REPORT ON

VIRTUAL MOUSE USING HAND GESTURES

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF

DIPLOMA IN COMPUTER ENGINEERING

SUBMITTED TO

STATE BOARD OF TECHNICAL EDUCATION, TELANGANA

UNDER THE GUIDENCE OF

SRI G.VENKATESHWARA PRASAD, Principal of GIOE.

SMT KODALI RADHIKA, H.O.D. of CME.

MR. KANTHOLLA ANIL KUMAR, Lecturer in CME.

GOVERNMENT INSTITUTE OF ELECTRONICS, SECUNDERABAD

(Approved by TS SBTET, Hyderabad)

East Marred pally, Secunderabad-500026

2019-2022.

TEAM GUIDED BY

MR. KANTHOLLA ANIL KUMAR, Lecturer in CME

TEAM MEMBERS

1. M. SAI GOWTHAM KUMAR	**19054-CM-017**
2. NAKKA VISHNU PRIYA	**19054-CM-028**
3. S. YESHWANTH	**19054-CM-029**
4. K. GEETANJALI	**19054-CM-031**
5. M. VAMSHI KRISHNA	**19054-CM-034**
6. P. PRABHAS CHANDER	**19054-CM-035**

ACKNOWLEDGMENT

We owe our gratitude to **SRI. G.VENKATESHWARA PRASAD,** M.Tech, Principal of Government Institute Of Electronics, Secunderabad for extending the college facilities to the successful pursuit of our project so far and for his kind patronage.

We sincerely give thanks to **SMT. KODALI RADHIKA,** M.Tech, H.O.D. of Computer Engineering and physical systems at GIOE for all the facilities provided to us in the pursuit of this project.

We are indebted to our project guide **MR. KANTHOLLA ANIL KUMAR,** Lecturer in CME at GIOE. We feel it’s an immense pleasure to be indebted to our guide for her valuable support, advice and encouragement. We thank her for her dedicated guidance towards this project.

We are grateful to all the staff members of CME department, of Government institute of electronics who have helped directly or indirectly during the project.

We acknowledge our deep sense of gratitude to our loving parents for being a constant source of information and motivation.

CONTENTS

1. ABSTRACT

Gesture-controlled laptops and computers have recently gained a lot of attraction. Leap motion is the name for this technique. Waving our hand in front of our computer/laptop allows us to manage certain of its functionalities. Over slides and overheads, computer-based presentations have significant advantages. Audio, video, and even interactive programmes can be used to improve presentations.

Unfortunately, employing these techniques is more complicated than using slides or overheads. The speaker must operate various devices with unfamiliar controls (e.g., keyboard, mouse, VCR remote control). In the dark, these devices are difficult to see, and manipulating them causes the presentation to be disrupted.

Hand gestures are the most natural and effortless manner of communicating. The camera’s output will be displayed on the monitor. The concept is to use a simple camera instead of a classic or standard mouse to control mouse cursor functions.

The Virtual Mouse provides an infrastructure between the user and the system using only a camera. It allows users to interface with machines without the use of mechanical or physical devices, and even control mouse functionalities. This study presents a method for controlling the cursor’s position without the need of any electronic equipment. While actions such as clicking and dragging things will be carried out using various hand gestures.

As an input device, the suggested system will just require a webcam. The suggested system will require the use of OpenCV and Python as well as other tools. The camera’s output will be presented on the system’s screen so that the user can further calibrate it.

2. INTRODUCTION

With the development technologies in the areas of augmented reality and devices that we use in our daily life, these devices are becoming compact in the form of Bluetooth or wireless technologies. This project proposes a Virtual Mouse System that makes use of the hand gestures and hand tip detection for performing mouse functions in the computer using computer vision.

The main objective of the proposed system is to perform computer mouse cursor functions and scroll function using a web camera or a built-in camera in the computer or a laptop instead of using a traditional mouse device. Hand gesture and hand tip detection by using computer vision is used as a Human Computing Interface (HCI) with the computer. With the use of the Virtual Mouse System, we can track the fingertip of the hand gesture by using a built-in camera or web camera and perform the mouse cursor operations and scrolling function and also move the cursor with it.

While using a wireless or a Bluetooth mouse, some devices such as the mouse, the dongle to connect to the PC, and also a battery to power the mouse to operate are used, but in this project, the user uses his/her built-in camera or a webcam and uses his/her hand gestures to control the computer mouse operations.

In the proposed system, the web camera captures and then processes the frames that have been captured and then recognizes the various hand gestures and hand tip gestures and then performs the particular mouse function. Programming language is used for developing the Virtual Mouse System, and also OpenCV which is the library for computer vision is used in the Virtual Mouse System.

This model makes use of the MediaPipe package for the tracking of the hands and for tracking of the tip of the hands, and also Pynput, Autopy, and PyAutoGUI packages were used for moving around the window screen of the computer for performing functions such as left click, right click, and scrolling functions.

The results of the proposed model showed very high accuracy level, and the proposed model can work very well in real-world application with the use of a CPU without the use of a GPU.

2.1. Problem Description and Overview

The proposed Virtual Mouse System can be used to overcome problems in the real world such as situations where there is no space to use a physical mouse and also for the persons who have problems in their hands and are not able to control a physical mouse. Also, amidst of the COVID-19 situation, it is not safe to use the devices by touching them because it may result in a possible situation of spread of the virus by touching the devices, so the proposed Virtual Mouse can be used to overcome these problems since hand gesture and hand Tip detection is used to control the PC mouse functions by using a webcam or a built-in camera.

2.2. Objective

The main objective of the proposed Virtual Mouse System is to develop an alternative to the regular and traditional mouse system to perform and control the mouse functions, and this can be achieved with the help of a web camera that captures the hand gestures and hand tip and then processes these frames to perform the particular mouse function such as Left Click, Right Click, and Scrolling function.

2.3. Abbrevations

GUI	Graphical User Interface
IDE	Integrated Development Environment
HCI	Human Computing Interaction
OpenCV	Open source Computer Vision
JSON	Java Script Object Notation
PYCAW	Python Core Audio Windows

3. PROJECT REQUIREMENTS

HARDWARE REQUIREMENTS:

RESOURCES	MINIMUM	RECOMMENDED
CPU	11th Gen Intel Core i5 or AMD Ryzen 3 Quad Core	11th Gen Intel Core i7 or AMD Ryzen 5 Quad Core
RAM	8GB DDR2 or better.	16GB DDR3 or better.
Storage	128GB HDD or better.	526GB HDD or better.
OS	Windows 7 (32-bit) or better.	Windows 10 (64-bit) or better.
Web Camera	Windows 7-720p	Windows 10-1080p

SOFTWARE REQUIREMENTS:

Operating System	Windows 10 or 11
Software	Python Framework
Editor	PyCharm or Python IDLE

4. 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 environment 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 are completely useless when it comes to interact with the computers remotely due to the cable lengths limitations, rendering it inaccessible.

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.

5. PACKAGES

OPEN CV:

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a template interface that works seamlessly with STL containers.

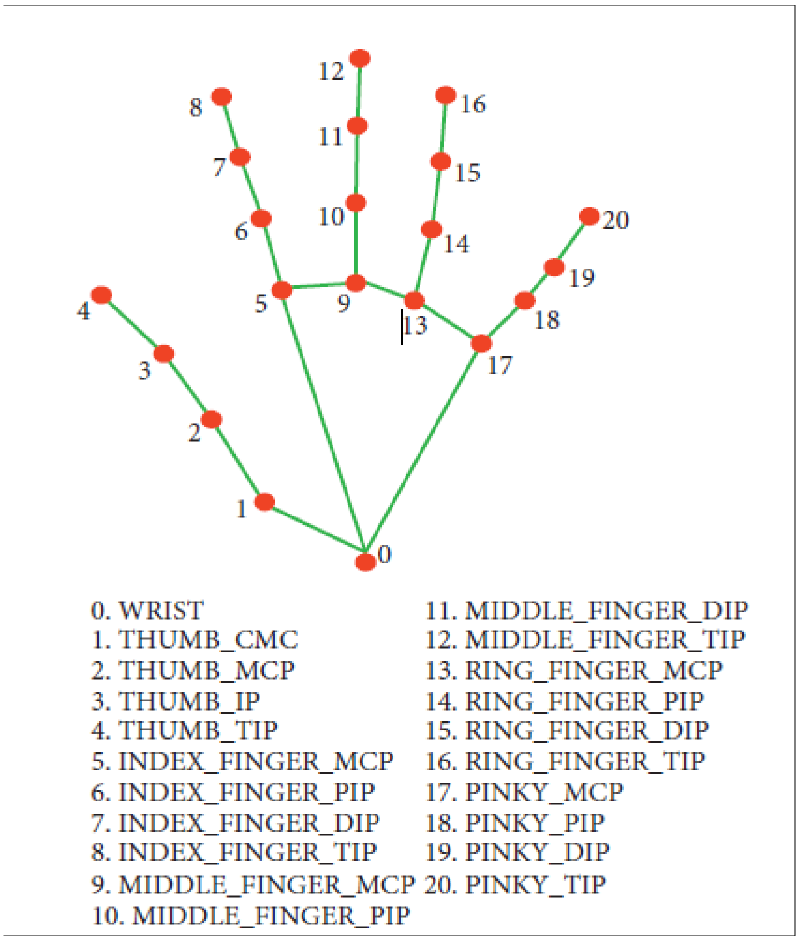
MEDIA PIPE:

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 analysing 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.

Applications of MediaPipe:

1. Object tracking.
2. Box tracking.
3. Face mesh.
4. Hair segmentation.
5. Live hand tracking.
6. Iris detection.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

PYAUTOGUI:

PyAutoGUI is essentially a Python package that works across Windows, MacOS X and Linux which provides the ability to simulate mouse cursor moves and clicks as well as keyboard button presses with PyAutoGUI is a Python automation library used to click, drag, scroll, move, etc. It can be used to click at an exact position.

It provides many features, and a few are given below:

1. We can move the mouse and click in the other applications' window.
2. We can send the keystrokes to the other applications.Ex: Filling out the form, typing the search query to browser, etc.
3. It allows us to locate a window of the application, and move, maximize, minimize, resizes, or close it.

MATH:

The math module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math . It gives access to the underlying C library functions.

ENUM:

Enum is a class in python for creating enumerations, which are a set of symbolic names (members) bound to unique, constant values. The members of an enumeration can be compared by these symbolic names, and the enumeration itself can be iterated over.

COMTYPES:

comtypes is a lightweight Python COM package, based on the ctypes FF library, in less than 10000 lines of code (not counting the tests). comtypes allows to define, call, and implement custom and dispatch-based COM interfaces in pure Python. It works on 64 bit Windows.

PYCAW:

Python Core Audio Windows Library, working for both Python2 and Python3.

GOOGLE.PROTOBUF.JSON_FORMAT:

It contains routines for printing protocol messages in JSON(Java Script Object Notation) format.

6. SYSTEM DEVELOPMENT

The various functions and conditions used in the system are explained in the flowchart of the real-time Virtual Mouse System.

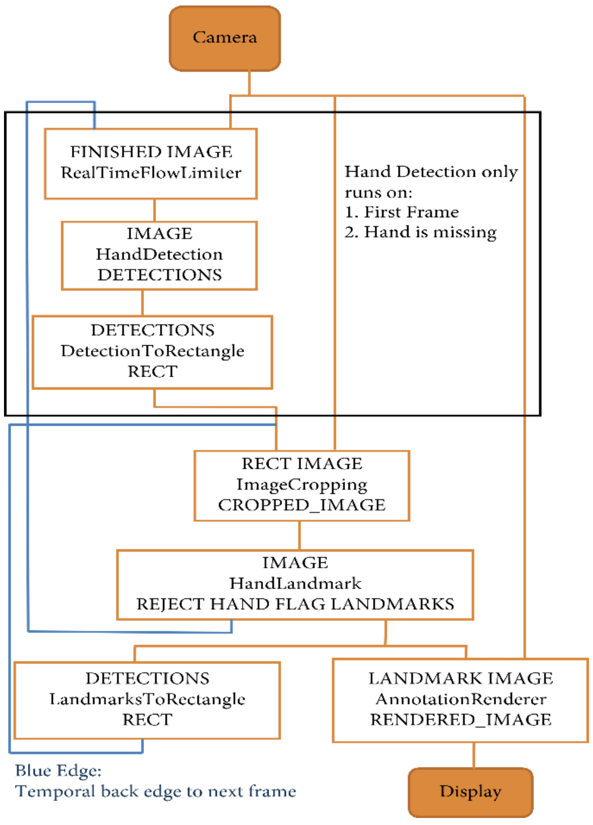
6.1. CAMERA USED IN THE VIRTUAL MOUSE SYSTEM:

The proposed 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 Virtual System.

6.2. CAPTURING THE VIDEO AND PROCESSING:

The 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)
```

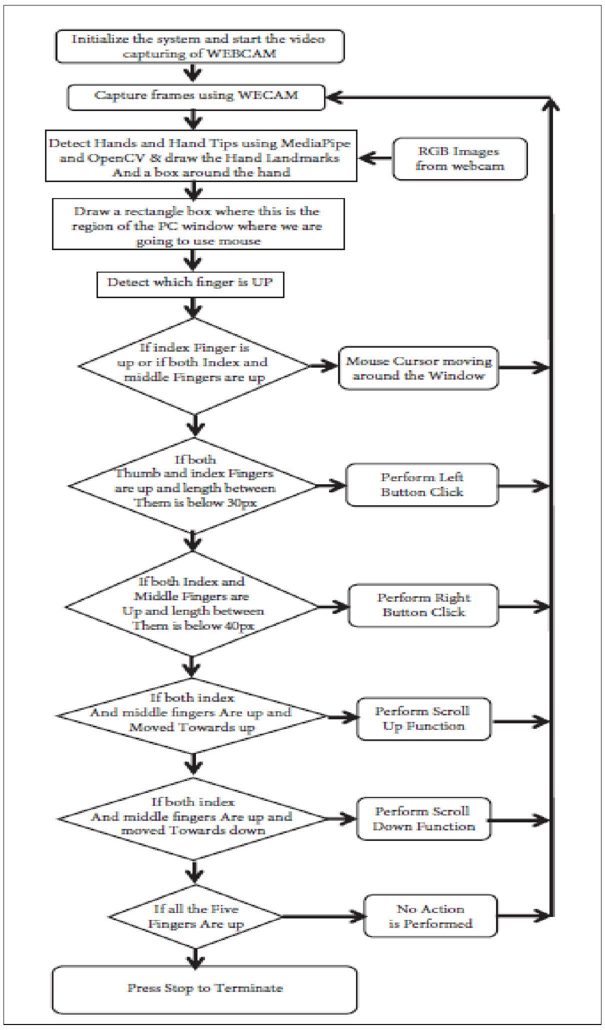


6.3. DETECTING WHICH FINGER IS UP AND PERFORMING THE PARTICULAR MOUSE FUNCTION:

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.

6.4. 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 PyAutoGUI package of Python.



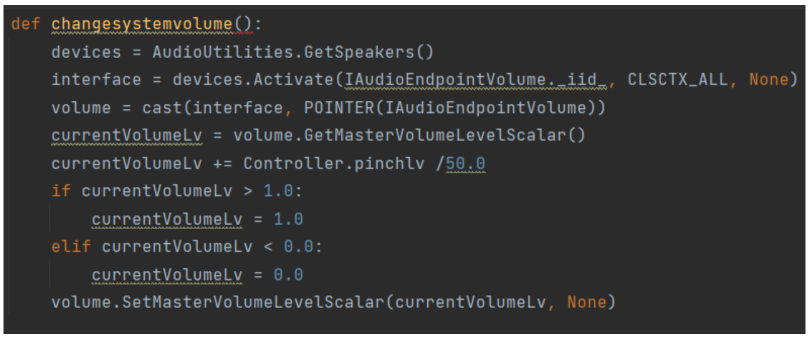
Initialize the system and start the video capturing of WEBCAM
Capture frames using WECAM
Detect Hands and Hand Tips using MediaPipe and OpenCV & draw the Hand Landmarks And a box around the hand
RGB Images from webcam
Draw a rectangle box where this is the region of the PC window where we are going to use mouse
Detect which finger is UP
If index Finger is up or if both Index and middle Fingers are up
Mouse Cursor moving around the Window
If both Thumb and index Fingers are up and length between Them is below 30px
Perform Left Button Click
If both Index and Middle Fingers are Up and length between Them is below 40px
Perform Right Button Click
If both index And middle fingers Are up and Moved Towards up
Perform Scroll Up Function
If both index And middle fingers Are up and moved Towards down
Perform Scroll Down Function
If all the Five Fingers Are up
No Action is Performed
Press Stop to Terminate

Flow Chart of Virtual Mouse

6.5. Mouse Functions:

Volume Control:

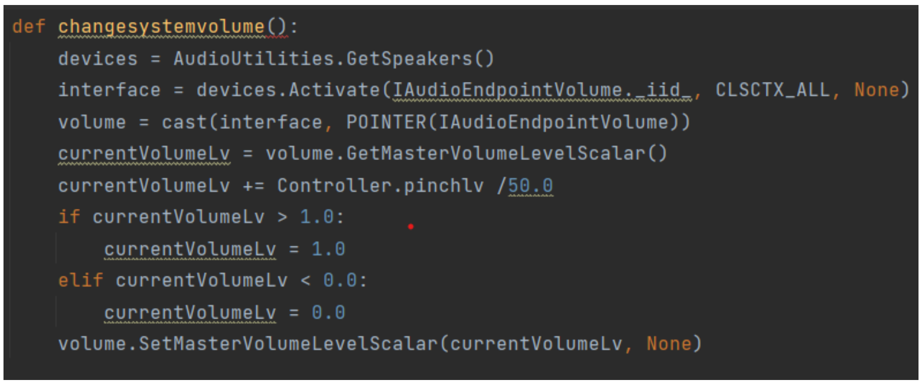
If Index and Thumb finger are pinched together and other three fingers are up, when the hand (right) is moved up and down, then the Volume Control operation is performed.



```
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)
```

Scrolling:

If Index and Thumb finger are pinched together and other three fingers are up, when the hand (left) is moved up and down, then the Scrolling operation is performed.



```
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)
```

Left Click:

If the Index finger is closed and the Middle finger is up, then the Left Click operation is performed.

Right Click:

If the Middle finger is closed and the Index finger is up, then the Right Click operation is performed.

Double Click:

If Middle and Index finger are touching to each other, then the Double Click operation is performed. The distance between both the fingers should be 0px(pixels).

No Action:

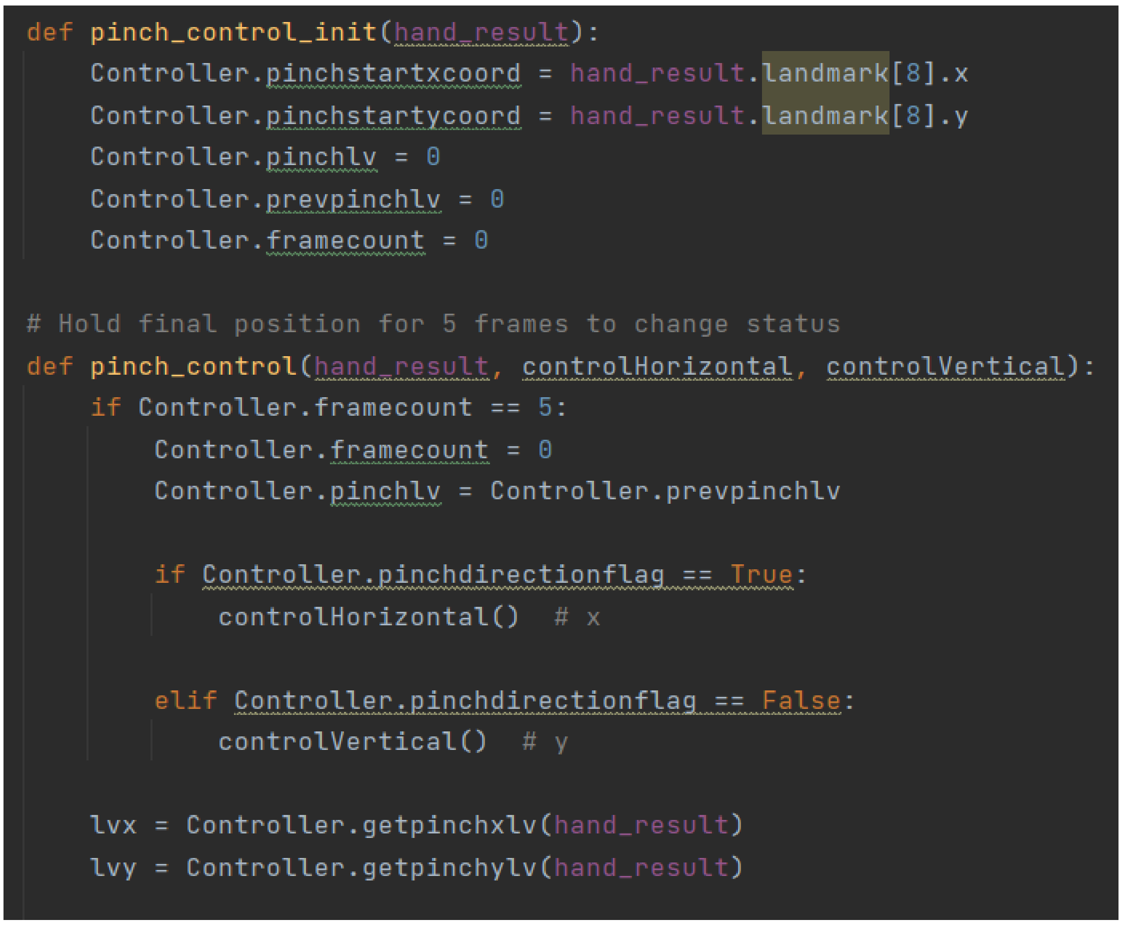
If all five fingers are up, then No Action operation is performed.

Drag Action:

If all five fingers are held/closed together and move the hand, then the Drag Action operation is performed.

Drop Action:

If all fingers are released/opened after moving the hand in the Drag Action, then the Drop Action is performed.



```python
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)
```

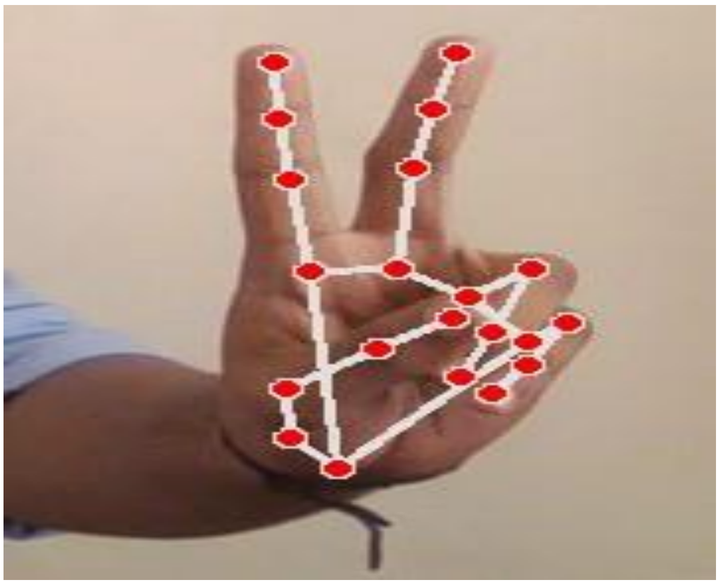
7. IMPLEMENTATION

CODE:

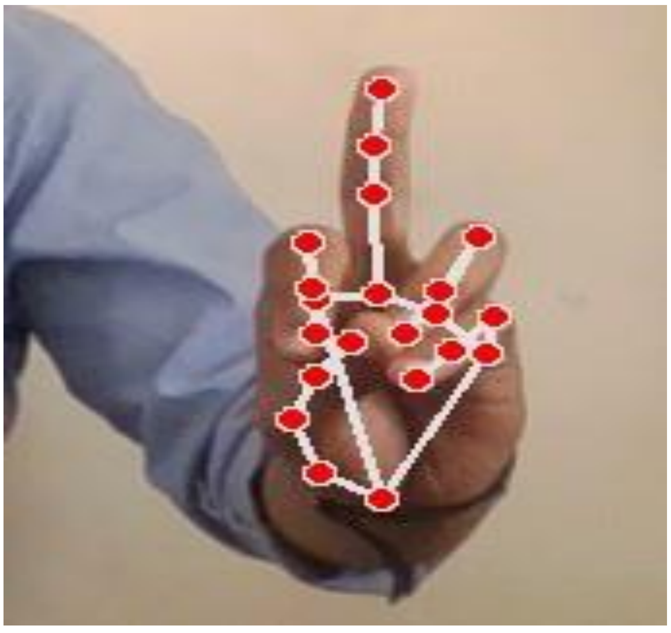
```
import cv2
import mediapipe as mp
import pyautogui
import math
from enum import IntEnum
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume
from google.protobuf.json_format import MessageToDict
import screen_brightness_control 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 Fluctations 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 getpinchxlv(hand_result):
        dist = round((hand_result.landmark[8].x - 
Controller.pinchstartxcoord ) *10 ,1)
        return dist

    def changesystembrightness():
        currentBrightnessLv = [i / 100 for i in sbcontrol.get_brightness()]
        a = [i / 50 for i in range(int(Controller.pinchlv))]
        currentBrightnessLv = currentBrightnessLv + a
        if currentBrightnessLv > [1.0]:
            currentBrightnessLv = 1.0
        elif currentBrightnessLv < [0.0]:
            currentBrightnessLv = [0.0]
        sbcontrol.fade_brightness([i * 100 for i in 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)

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()
gc1 = GestureController()
gc1.start()
```

8. OUTPUT SCREENSHOTS:

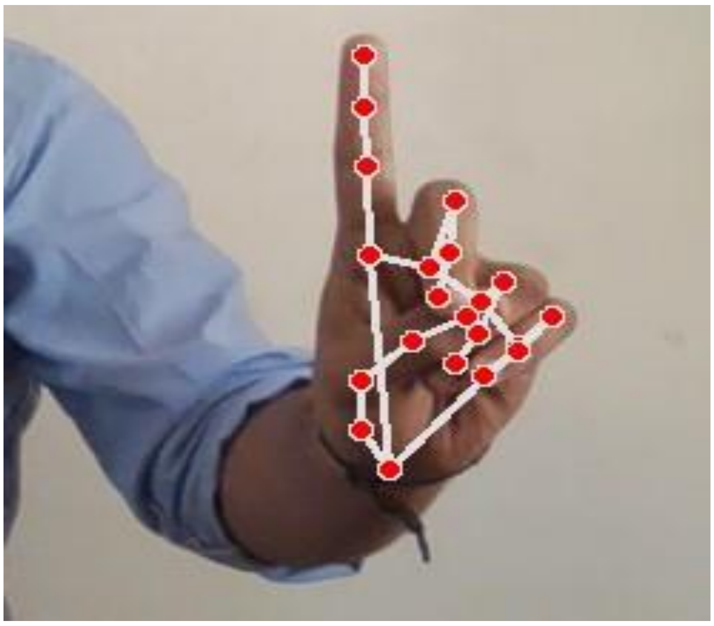
(i) CURSOR MOVEMENT GESTURE



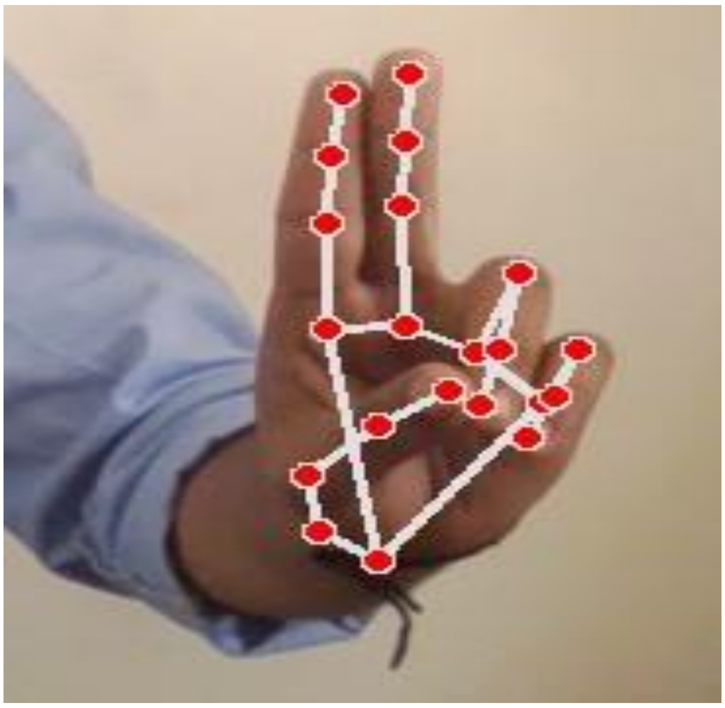
(ii) LELT CLICK GESTURE



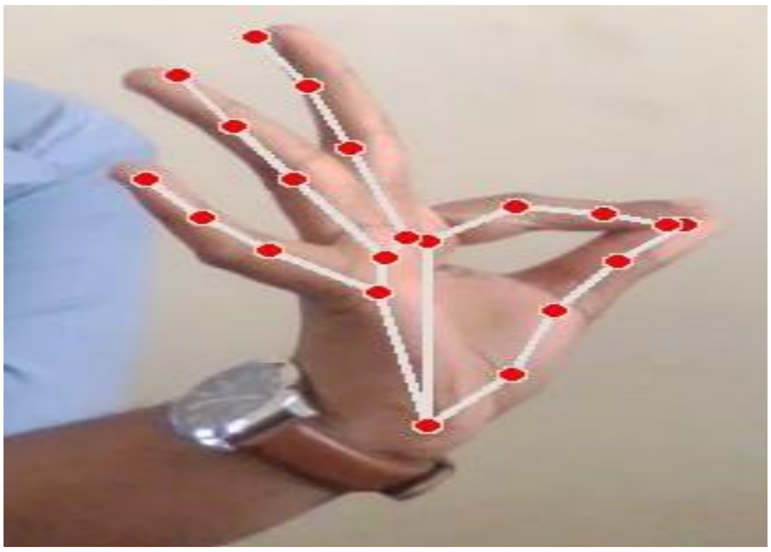
(iii) RIGHT CLICK GESTURE



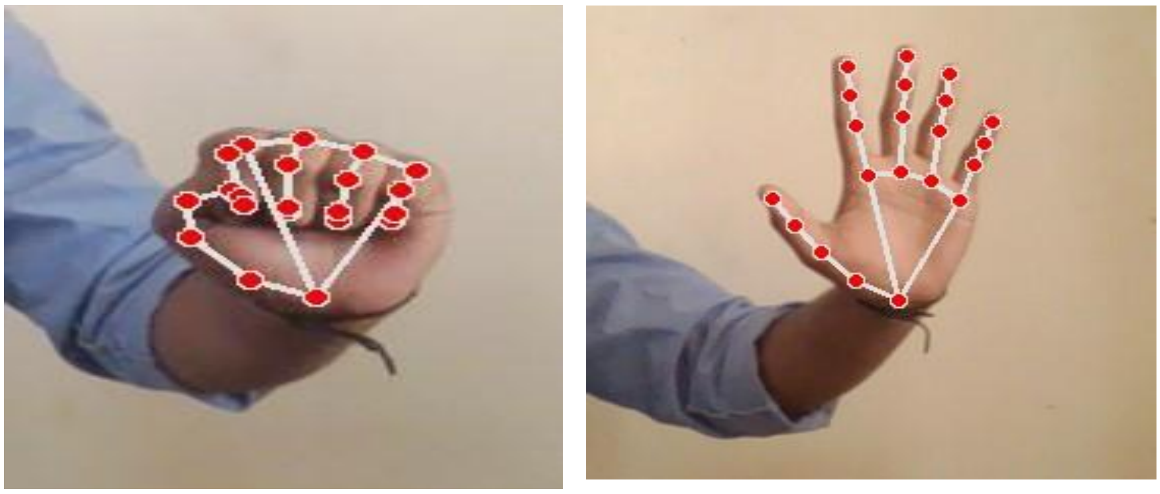
(iv) DOUBLE CLICK GESTURE



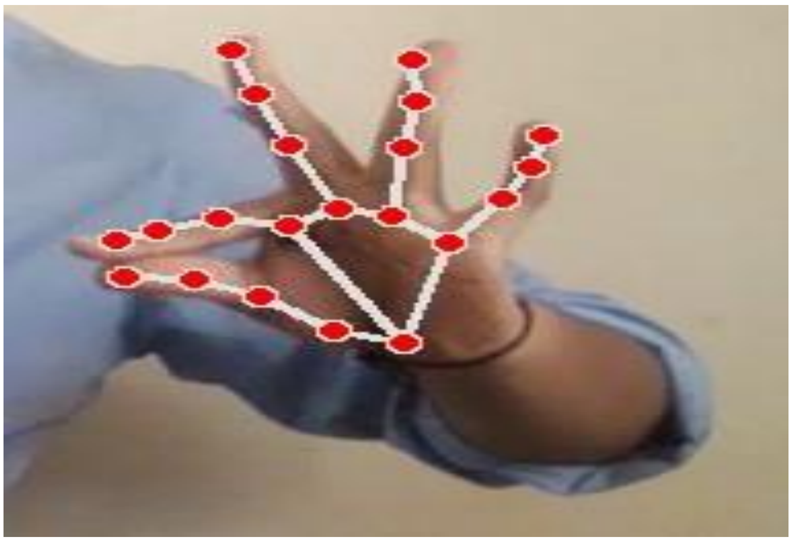
(v) SCROLLING GESTURE



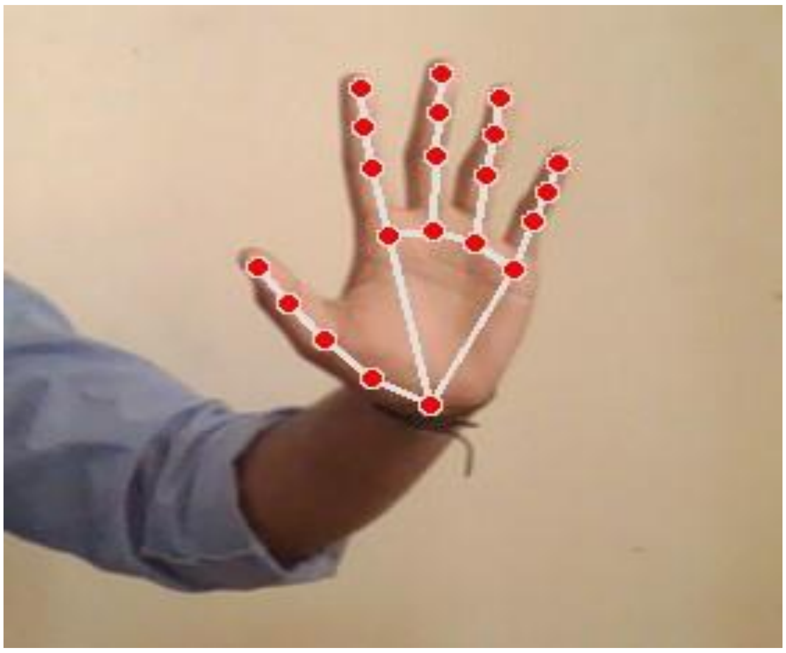
(vi) FILE TRANSFERRING GESTURE



(vii) VOLUME CONTROL



(viii) NO ACTION GESTURE



9. PERFORMANCE ANALYSIS

In the proposed virtual mouse system, the concept of advancing the human-computer interaction using computer vision is given.

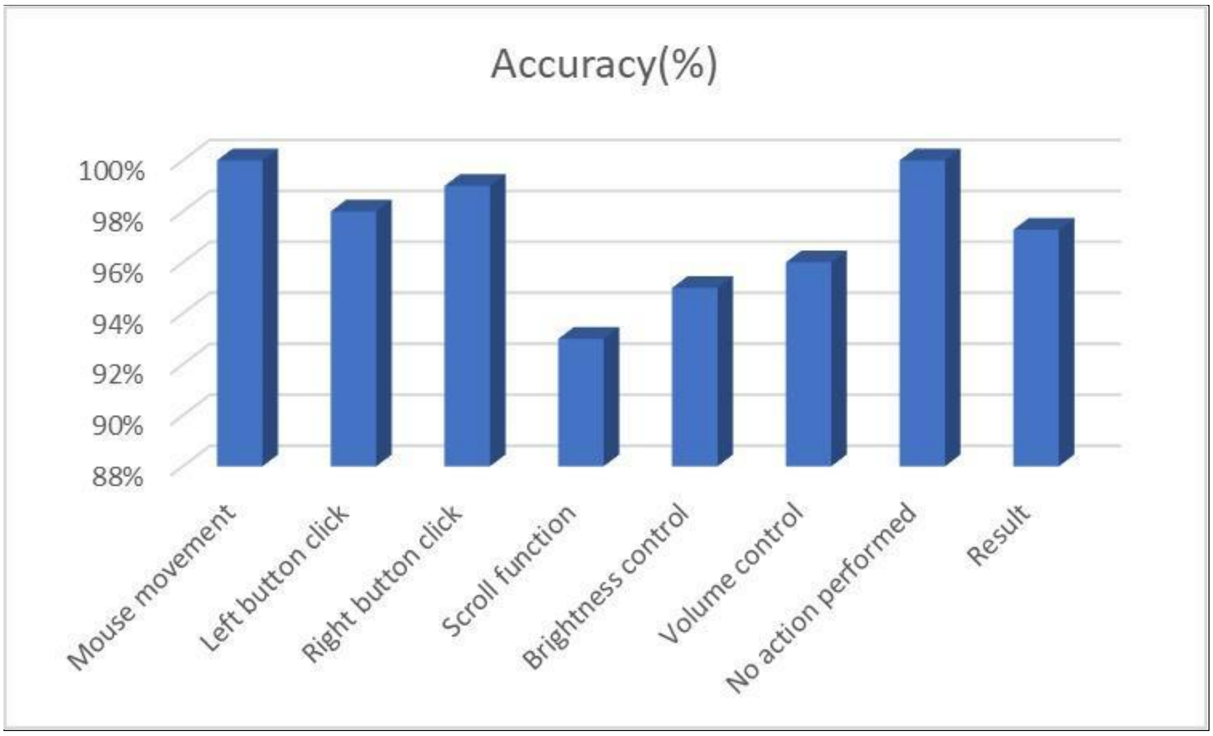
Cross comparison of the testing of the AI virtual mouse system is difficult because only limited numbers of datasets are available. The hand gestures and fingertip detection have been tested in various illumination conditions and also been tested with different distances from the webcam for tracking of the hand gesture and hand tip detection. An experimental test has been conducted to summarize the results shown in Table.

The test was performed 25 times by 4 persons resulting in 600 gestures with manual labelling, and this test has been made in different light conditions and at different distances from the screen, and each person tested the AI virtual mouse system 10 times in normal light conditions, 5 times in faint light conditions, 5 times in close distance from the webcam, and 5 times in long distance from the webcam, and the experimental results are tabulated in Table.

Mouse function performed	Success	Failure	Accuracy (%)
Mouse movement	100	0	100%
Left button clicks	98	2	98%
Right button clicks	99	1	99%
Scroll function	93	7	93%
Brightness control	95	5	95%
Volume control	96	4	96%
No action performed	100	0	100%
Result	681	19	97.28%

From Table, it can be seen that the proposed AI virtual mouse system had achieved an accuracy of about 97%. From this 97% accuracy of the proposed AI virtual mouse system, we come to know that the system has performed well. As seen in Table, the accuracy is low for "Scroll function" as this is the hardest gesture for the computer to understand. The accuracy for scroll function is low because the gesture used for performing the particular mouse function is harder.

Also, the accuracy is very good and high for all the other gestures. Compared to previous approaches for virtual mouse, our model worked very well with 97% accuracy. The graph of accuracy is shown in figure



10. CONCLUSION AND FUTURE SCOPE

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.

Test Cases:

Test case id	**Scenario**	**Boundary Value**	**Expected Result**	**Actual Result**	**Status**
1	Used in normal environment.	>90%	In normal environment hand gestures can be recognized easily.	Hand gestures got easily recognized and work properly.	**Passed**
2	Used in bright environment.	>60%	In brighter environment, software should work fine as it easily detects the hand movements but in a brighter condition it may not detect the hand gestures as expected.	In bright conditions the software works very well.	**Passed**
3	Used in dark environment	<30%	In dark environment, it should work properly.	In dark environment software didn't work properly in detecting hand gestures.	**Failed**
4	Used at a near distance (15cm) from the web cam.	>80%	At this distance, this software should perform perfectly.	It works fine and all features works properly.	**Passed**
5	Used at a far distance (35cm) from the web cam.	>95%	At this distance, this software should work fine.	At this distance, it is working properly.	**Passed**
6	Used at a farther distance (60cm) from the web cam.	>60%	At this distance, there will be some problem in detecting hand gestures but it should work fine.	At this distance, the functions of this software work properly.	**Passed**

11. REFERENCES

1) OpenCV Website – www.opencv.org
2) MSDN Microsoft developers' network – www.msdn.microsoft.com
3) Code project – 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