A
Project Report
on

GUESTURE CONTROLLED VIRTUAL MOUSE

Submitted by :
V. Ganesh Naik R180366

Team :
K. Manivarma R180238
V. Ganesh Naik R180366

Under the guidance of

Mr. K. Vinod Kumar
(Assistant Professor)

Department of Computer Science Engineering
Rajiv Gandhi University of Knowledge Technologies
R K Valley, Y.S.R. Kadapa (Dis) – 516330

This Project Report has been submitted in fulfilment of the requirements for the Degree of Bachelor of Technology in Software Engineering

December -2023
Rajiv Gandhi University of Knowledge Technologies
IIIT R.K. Valley, Y.S.R. Kadapa (Dist.) – 516330

CERTIFICATE

This is to certify that the report entitled “Gesture Controlled Virtual Mouse” submitted by K.Manivarma, bearing ID. No. R180238 and V. Ganesh Naik, bearing ID. No. R180366 in partial fulfilment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering is a bonafide work carried out by them under my supervision and guidance.

The report has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

GUIDE **Head ofDepartment**

Mr. K. Vinod Kumar
Assistant Professor

Mr. N. Satyanandaram
HOD OF CSE

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would incomplete without the mention of the people who make it possible and who’s constant guidance and encouragement crown all the efforts success.

We would like to express my sincere gratitude to **Mr. k. Vinod kumar sir**, our project internal guide for valuable suggestions and keen interest throughout the progress of my course of research.

We are grateful to **N. Satyanandaram sir, HOD CSE**, for providing excellent computing facilities and a congenial atmosphere for progressing with our project. At the outset, I would like to thank Rajiv Gandhi University of Knowledge Technologies, for providing all the necessary resources and support for the successful completion of my course work.

DECLARATION

We here by declare that this report entitled “Gesture Controlled Virtual Mouse” submitted by us under the guidance and supervision of Mr. K. Vinod Kumar, is a bonafide work. We also declare that it has not been of submitted previously in part or in full to this University or other institution for the award of any degree or diploma.

Date :- 19-12-2023	K. Manivarma(R180238)
Place:- RK Valley	V.GaneshNaik(R180366)

TABLE

ABSTRACT

This project introduces a Gesture-Controlled Virtual Mouse (GVM) as a pioneering solution to redefine the conventional methods of human-computer interaction. The GVM system leverages computer vision techniques to interpret hand gestures, allowing users to control a virtual mouse through intuitive and natural movements. The project focuses on enhancing user experience by providing a hands-free and adaptable input method for navigating digital interfaces.

The GVM employs a depth-sensing camera to capture real-time hand gestures, which are then processed using advanced algorithms to recognize specific actions such as pointing, clicking, and scrolling. The translated gestures seamlessly control the virtual mouse, offering users an alternative and engaging means of interacting with computing devices.

Key aspects of the Gesture-Controlled Virtual Mouse project include:

1. Gesture Recognition: The project implements robust gesture recognition algorithms to accurately interpret a variety of hand movements, enabling users to execute diverse mouse actions effortlessly.

2. Real-time Interaction: Utilizing computer vision technologies, the GVM ensures real-time responsiveness, creating a natural and immediate connection between the user's gestures and on-screen actions.

3. Adaptability: The system is designed to be adaptable across different computing platforms, making it suitable for desktops, laptops, and potentially extending to virtual reality environments, thereby broadening its applicability.

4. User Customization: The GVM allows users to customize gesture commands, accommodating individual preferences and optimizing the virtual mouse control experience for diverse user needs.

INTRODUCTION

Gesture Controlled Virtual Mouse makes human computer interaction simple by making use of Hand Gestures and Voice Commands. The computer requires almost no direct contact. All i/o operations can be virtually controlled by using static and dynamic hand gestures along with a voice assistant. This project makes use of the state-of-art Machine Learning and Computer Vision algorithms to recognize hand gestures and voice commands, which works smoothly without any additional hardware requirements. It leverages models such as CNN implemented by MediaPipe running on top of pybind11. It consists of two modules: One which works direct on hands by making use of MediaPipe Hand detection, and other which makes use of Gloves of any uniform color. Currently it works on Windows platform.

In the dynamic landscape of human-computer interaction, the demand for more intuitive and natural interfaces has led to the exploration of innovative technologies. The Gesture-Controlled Virtual Mouse (GVM) project emerges at the intersection of computer vision, machine learning, and user interface design, aiming to redefine how users interact with digital environments. This project introduces a groundbreaking approach to navigating computing devices by seamlessly translating hand gestures into virtual mouse actions, offering a hands-free and engaging alternative to traditional input methods.

Traditional input devices, such as mice and touchpads, have long been the primary means for users to interact with computers. However, these methods often present limitations in terms of adaptability and accessibility. The GVM project seeks to address these limitations by harnessing the power of gesture recognition and real-time processing to create an interface that is not only intuitive but also adaptable across various computing platforms.

The core concept of the Gesture-Controlled Virtual Mouse lies in its ability to interpret and respond to a user's hand gestures, providing a natural and immersive interaction experience. Leveraging computer vision technologies, the system captures and analyzes the intricate movements of the user's hands, recognizing gestures that correspond to standard mouse actions, including pointing, clicking, scrolling, and more.

As the digital landscape continues to evolve, the GVM project stands at the forefront of innovation, promising to revolutionize the way users engage with their devices. This introduction sets the stage for a comprehensive exploration of the project, delving into the underlying technologies, implementation strategies, and potential applications of the Gesture-Controlled Virtual Mouse. Through this endeavor, we aim to contribute to the ongoing evolution of human-computer interaction, creating a more accessible, adaptive, and user-friendly computing experience for individuals across diverse contexts and abilities.

PURPOSE

The Gesture-Controlled Virtual Mouse (GVM) project serves a multifaceted purpose aimed at revolutionizing human-computer interaction by introducing a hands-free and intuitive input method. The primary purposes of the GVM project include:

1. Enhancing User Experience: The project seeks to improve the overall user experience by providing a more natural and immersive way to interact with digital interfaces. Gesture control eliminates the need for physical input devices, such as mice or touchpads, allowing users to navigate and manipulate on-screen elements using intuitive hand movements.

2. Accessibility: One of the key purposes of the GVM project is to enhance accessibility for users with physical disabilities or limitations. By introducing a gesture-based control system, individuals with mobility challenges can interact with computing devices without relying on traditional input devices, thereby promoting inclusivity in the digital space.

3. Adaptability Across Platforms: The GVM project aims to create a versatile solution that can be seamlessly integrated into various computing platforms, including desktops, laptops, and potentially virtual reality environments. This adaptability ensures that the gesture-controlled interface can cater to different user needs and preferences.

4. Innovation in Human-Computer Interaction: The project serves as a platform for innovation in the field of human-computer interaction. By exploring novel methods of input, the GVM project contributes to the ongoing evolution of interface design, potentially inspiring new ways for users to engage with and control their digital devices.

5. Customization and Personalization: The GVM project allows users to customize and define their own set of gestures, tailoring the virtual mouse control experience to individual preferences. This purpose emphasizes the project's commitment to providing a personalized computing experience that adapts to the unique requirements of each user.

6. Research and Development: The project serves as a research and development initiative to explore the technical challenges associated with gesture recognition, real-time processing, and the integration of such technologies into practical computing applications. It contributes to the advancement of computer vision and machine learning techniques in the context of human-computer interaction.

OVERALL DESCRIPTION

Gesture Controlled Virtual Mouse makes human computer interaction simple by making use of Hand Gestures and Voice Commands. The computer requires almost no direct contact. All i/o operations can be virtually controlled by using static and dynamic hand gestures along with a voice assistant. This project makes use of the state-of-art Machine Learning and Computer Vision algorithms to recognize hand gestures and voice commands, which works smoothly without any additional hardware requirements. It leverages models such as CNN implemented by MediaPipe running on top of pybind11. It consists of two modules: One which works direct on hands by making use of MediaPipe Hand detection, and other which makes use of Gloves of any uniform color. Currently it works on Windows platform.

Applications:

1. Computing Devices: The GVM project can be applied to traditional computing devices such as desktops, laptops, and tablets. Users can control the virtual mouse through hand gestures, providing a hands-free alternative to conventional input devices.

2. Gaming: In the gaming industry, the GVM project can enhance user experiences by introducing immersive and natural control mechanisms. Gamers can use gestures for actions like pointing, selecting, and manipulating objects within the virtual environment, adding a new dimension to gameplay.

3. Virtual Reality (VR) and Augmented Reality (AR): Gesture control is particularly valuable in VR and AR environments. The GVM project can be utilized to navigate virtual spaces, interact with virtual objects, and perform actions within immersive simulations, making the user experience more intuitive and engaging.

4. Presentations and Conferencing: The GVM project can simplify presentation controls by allowing users to navigate slides, highlight content, and interact with presentation software using gestures. In video conferencing scenarios, users can control applications and collaborate more seamlessly.

5. Accessibility: The GVM project has significant applications in accessibility, providing an alternative input method for individuals with physical disabilities. Users who may face challenges using traditional input devices can benefit from gesture-controlled interfaces to navigate and interact with computers.

6. Healthcare Interfaces: In healthcare settings, where hygiene and touchless interaction are crucial, the GVM project can be applied to control medical software and devices. Surgeons, for example, could use gestures to navigate digital imaging systems during surgeries without physical contact.

7. Smart Homes and IoT Devices: Gesture control can be integrated into smart home systems, allowing users to interact with IoT devices using hand movements. For example, controlling lights, thermostats, or home entertainment systems without physical touch.

8. Education and Training: In educational settings, the GVM project can be applied to interactive learning environments. Students and educators can navigate through educational content, manipulate virtual objects, and engage in collaborative learning experiences using gestures.

9. Industrial and Manufacturing Control: Gesture control has applications in industrial settings where touchless interaction is preferred. Engineers and operators can use gestures to control machinery, monitor processes, and interact with control systems in manufacturing environments.

SYSTEM REQUIREMENTS

- SOFTWARE COMPONENTS

  Operating System (only Windows)

  Technologies : Python, Anaconda, Machine Learning, Computer Vision.

- HARDWARE COMPONENTS

  Processor – core i5 (minimum)

  HardDisk - 512GB

  Ram – 8GB (minimum)

TOOLS AND TECHNOLOGIES USED

Python:

Python is a high-level programming language known for its simplicity, readability, and versatility. It is widely used across various domains, from web development and data analysis to scientific computing and artificial intelligence. Python's design philosophy emphasizes code readability and a clean syntax, which makes it an excellent language for both beginners and experienced developers.

Machine learning:

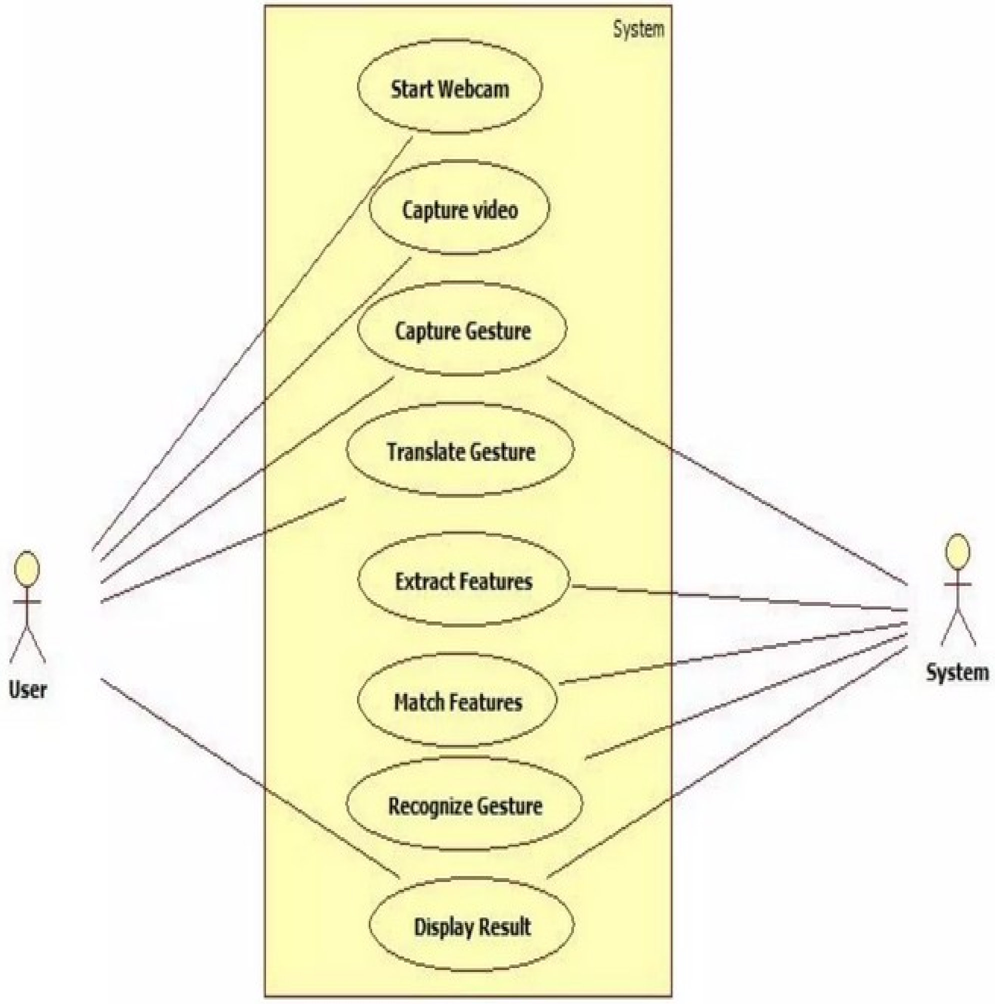
Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. One of its application is handwritten digit recognition

Computer vision:

Computer vision is a multidisciplinary field of artificial intelligence and computer science that focuses on enabling machines to interpret and understand visual information from the world. It involves developing algorithms and systems that empower computers to recognize, analyze, and make decisions based on images and videos. Inspired by human vision, computer vision encompasses tasks such as image recognition, object detection, facial analysis, and scene understanding. By employing techniques from machine learning, pattern recognition, and image processing, computer vision has applications in diverse fields, including healthcare, autonomous vehicles, augmented reality, manufacturing, and more. Its ultimate goal is to enable machines to "see" and comprehend the visual world, bridging the gap between the digital and physical realms.

WORKING MODEL

USECASE DIAGRAM



SOURCE CODE

Gesture_Controller.py

```
# 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
    """
    Enum for mapping all hand gesture to binary number.
    """

    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:
    """
    Convert Mediapipe Landmarks to recognizable Gestures.
    """

    def __init__(self, hand_label):
        """
        Constructs all the necessary attributes for the HandRecog object.

         Parameters
            finger : int
                Represent gesture corresponding to Enum 'Gest',
                stores computed gesture for current frame.
            ori_gesture : int
                Represent gesture corresponding to Enum 'Gest',
                stores gesture being used.
            prev_gesture : int
                Represent gesture corresponding to Enum 'Gest',
                stores gesture computed for previous frame.
            frame_count : int
                total no. of frames since 'ori_gesture' is updated.
            hand_result : Object
                Landmarks obtained from mediapipe.
            hand_label : int
                Represents multi-handedness corresponding to Enum 'HLabel'.
      """

        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):
        """
        returns signed euclidean distance between 'point'.

        Parameters
        ----------
        point : list containg two elements of type list/tuple which
represents landmark point.
```

```
        Returns
        -------
        float
        """
        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):
        """
        returns euclidean distance between 'point'.

        Parameters
        ----------
        point : list contaning two elements of type list/tuple which
represents landmark point.

        Returns
        -------
        float
        """
        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):
        """
        returns absolute difference on z-axis between 'point'.

        Parameters
        ----------
        point : list contaning two elements of type list/tuple which
represents landmark point.

        Returns
        -------
        float
        """
        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):
        """
        set 'finger' by computing ratio of distance between finger tip
      , middle knuckle, base knuckle.

        Returns
        -------
        None
        """
        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):
        """
        returns int representing gesture corresponding to Enum 'Gest'.
        sets 'frame_count', 'ori_gesture', 'prev_gesture',
       handles fluctations due to noise.

       Returns
        -------
        int
        """
        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:
    """
    Executes commands according to detected gestures.

    Attributes
    ----------
    tx_old : int
        previous mouse location x coordinate
    ty_old : int
        previous mouse location y coordinate
    flag : bool
        true if V gesture is detected
    grabflag : bool
        true if FIST gesture is detected
    pinchmajorflag : bool
        true if PINCH gesture is detected through MAJOR hand,
        on x-axis 'Controller.changesystembrightness',
        on y-axis 'Controller.changesystemvolume'.
    pinchminorflag : bool
        true if PINCH gesture is detected through MINOR hand,
        on x-axis 'Controller.scrollHorizontal',
        on y-axis 'Controller.scrollVertical'.
```

```
    pinchstartxcoord : int
        x coordinate of hand landmark when pinch gesture is started.
    pinchstartycoord : int
        y coordinate of hand landmark when pinch gesture is started.
    pinchdirectionflag : bool
        true if pinch gesture movment is along x-axis,
        otherwise false
    prevpinchlv : int
        stores quantized magnitued of prev pinch gesture displacment, from
        starting position
    pinchlv : int
        stores quantized magnitued of pinch gesture displacment, from
        starting position
    framecount : int
        stores no. of frames since 'pinchlv' is updated.
    prev_hand : tuple
        stores (x, y) coordinates of hand in previous frame.
    pinch_threshold : float
        step size for quantization of 'pinchlv'.
    """

    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):
         """returns distance beween starting pinch y coord and current hand
position y coord."""
        dist = round((Controller.pinchstartycoord -
hand_result.landmark[8].y)*10,1)
        return dist

    def getpinchxlv(hand_result):
        """returns distance beween starting pinch x coord and current hand
position x coord."""
        dist = round((hand_result.landmark[8].x -
Controller.pinchstartxcoord)*10,1)
        return dist

    def changesystembrightness():
```

```
        """sets system brightness based on 'Controller.pinchlv'."""
        currentBrightnessLv = sbcontrol.get_brightness(display=0)/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(display=0))

    def changesystemvolume():
        """sets system volume based on 'Controller.pinchlv'."""
        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():
        """scrolls on screen vertically."""
        pyautogui.scroll(120 if Controller.pinchlv>0.0 else -120)


    def scrollHorizontal():
        """scrolls on screen horizontally."""
        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):
        """
        returns coordinates of current hand position.

        Locates hand to get cursor position also stabilize cursor by
        dampening jerky motion of hand.

        Returns
        -------
        tuple(float, float)
        """
        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):
        """Initializes attributes for pinch gesture."""
        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):
        """
        calls 'controlHorizontal' or 'controlVertical' based on pinch flags,
        'framecount' and sets 'pinchlv'.

        Parameters
        ----------
        hand_result : Object
            Landmarks obtained from mediapipe.
        controlHorizontal : callback function assosiated with horizontal
            pinch gesture.
        controlVertical : callback function assosiated with vertical
             pinch gesture.

        Returns
        -------
        None
        """
        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):
        """Impliments all gesture functionality."""
        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)
```

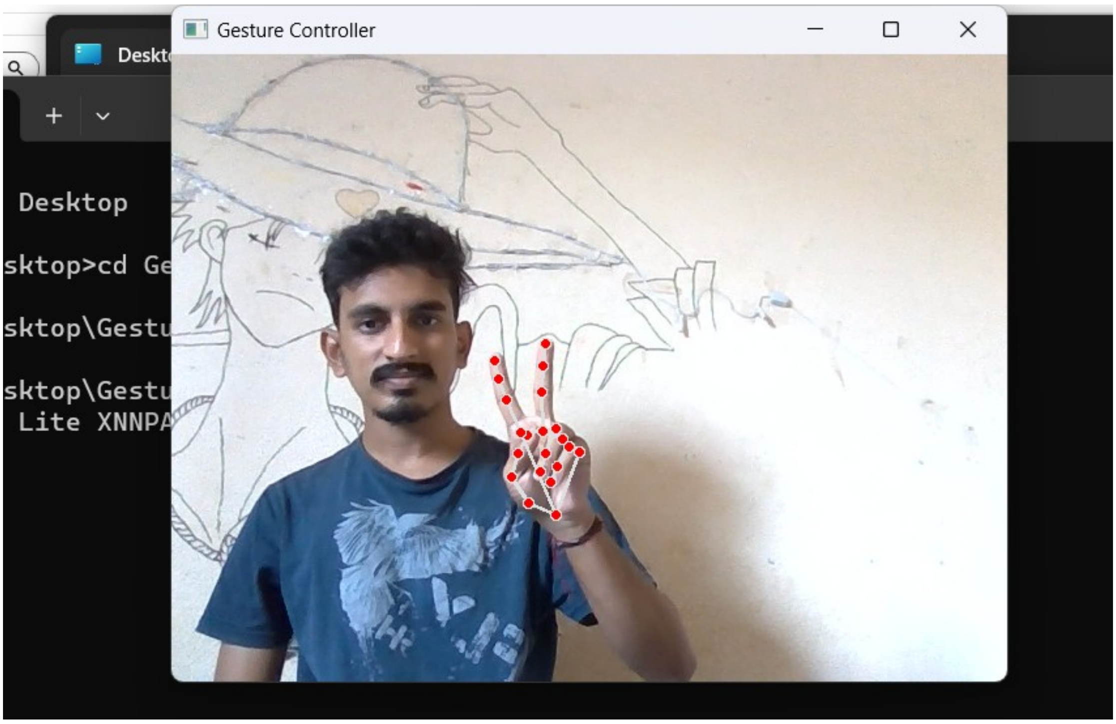
```
        elif gesture == Gest.FIST:
            if not Controller.grabflag :
                Controller.grabflag = True
                pyautogui.mouseDown(button = "left")
            pyautogui.moveTo(x, y, duration = 0.1)

        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:
    """
    Handles camera, obtain landmarks from mediapipe, entry point
    for whole program.

    Attributes
    ----------
    gc_mode : int
        indicates weather gesture controller is running or not,
        1 if running, otherwise 0.
    cap : Object
        object obtained from cv2, for capturing video frame.
    CAM_HEIGHT : int
```

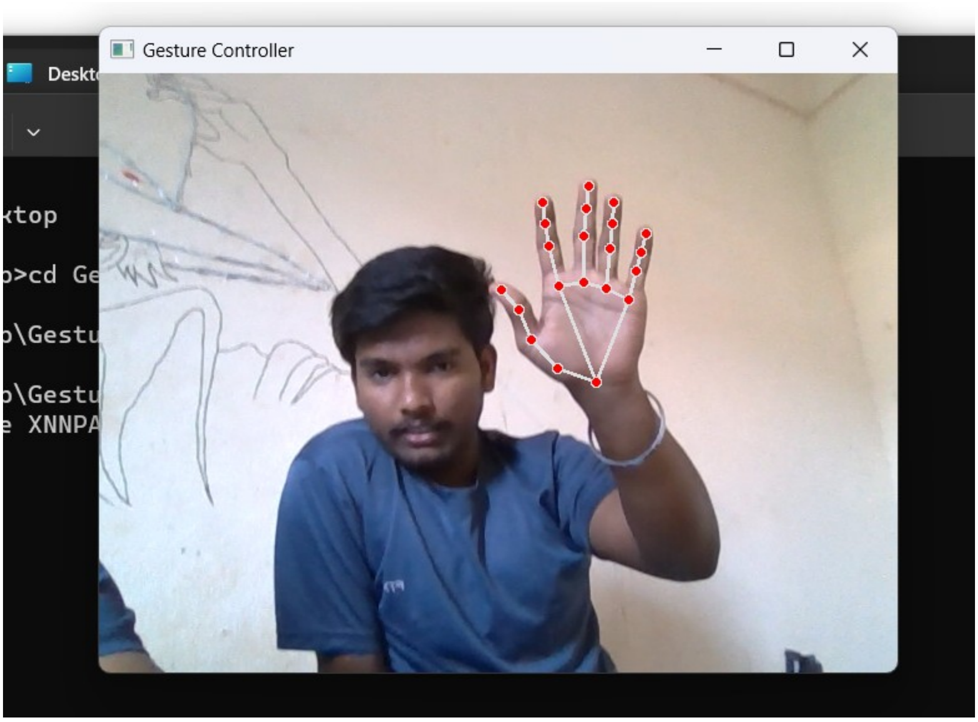
```
        highet in pixels of obtained frame from camera.
    CAM_WIDTH : int
        width in pixels of obtained frame from camera.
    hr_major : Object of 'HandRecog'
        object representing major hand.
    hr_minor : Object of 'HandRecog'
        object representing minor hand.
    dom_hand : bool
        True if right hand is domaniant hand, otherwise False.
        default True.
    """
    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):
        """Initilaizes attributes."""
        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):
        """
        sets 'hr_major', 'hr_minor' based on classification(left, right) of
        hand obtained from mediapipe, uses 'dom_hand' to decide major and
        minor hand.
        """
        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):
        """
        Entry point of whole programm, caputres video frame and passes,
obtains
        landmark from mediapipe and passes it to 'handmajor' and 'handminor'
for
        controlling.
        """

        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()
```

OUTPUT





FUNCTIONAL TESTING

S.NO	ACTION	TEST DESCRIPTION	EXPECTED OUTPUT	ACTUAL OUTPUT	FINAL OUTPUT
1.	Run the Gesture_Controller.py in terminal	Running the main program	Camera should open	Camera opened	Pass
2.	Hand is recognition	To test that hand is recognised or not	Recognised	Recognised	Pass
3.	Test whether the left click by left finger	To test left finger gesture is working or not	Working	Working	Pass
4.	Test whether the left click by left finger	To test right finger gesture is working or not	Working	Working	Pass
5.	Test whether the two figures working as a cursor or not	To check two fingers working as a cursor or not	Working	Working	Pass
6.	Drag and Drop gesture is working or not	To check drag and drop option is working or not	Working	Working	Pass
7.	Check whether gesture to multiple selection	Check whether gesture to multiple selection is working or not	Working	Working	Pass

CONCLUSION

Gesture Controlled Virtual Mouse makes human computer interaction simple by making use of Hand Gestures and Voice Commands. The computer requires almost no direct contact. All i/o operations can be virtually controlled by using static and dynamic hand gestures along with a voice assistant. This project makes use of the state-of-art Machine Learning and Computer Vision algorithms to recognize hand gestures and voice commands, which works smoothly without any additional hardware requirements. It leverages models such as CNN implemented by MediaPipe running on top of pybind11. It consists of two modules: One which works direct on hands by making use of MediaPipe Hand detection, and other which makes use of Gloves of any uniform color. Currently it works on Windows platform.

As the digital landscape continues to evolve, the GVM project stands at the forefront of innovation, promising to revolutionize the way users engage with their devices. This introduction sets the stage for a comprehensive exploration of the project, delving into the underlying technologies, implementation strategies, and potential applications of the Gesture-Controlled Virtual Mouse. Through this endeavor, we aim to contribute to the ongoing evolution of human-computer interaction, creating a more accessible, adaptive, and user-friendly computing experience for individuals across diverse contexts and abilities.

References:

Google. Mediapipe. https://developers.google.com/mediapipe
Pyautogui. https://pyautogui.readthedocs.io/en/latest/
Computer Vision. https://docs.opencv.org/3.4/d6/d00/tutorial_py_root.html