CA 4 Project Code Appendix and Output Screenshots
(“WaveControl” - Hands-Free Windows Navigation)

Ruthvika Muchala (E0121036), Ayushmaan Das (E0121037)

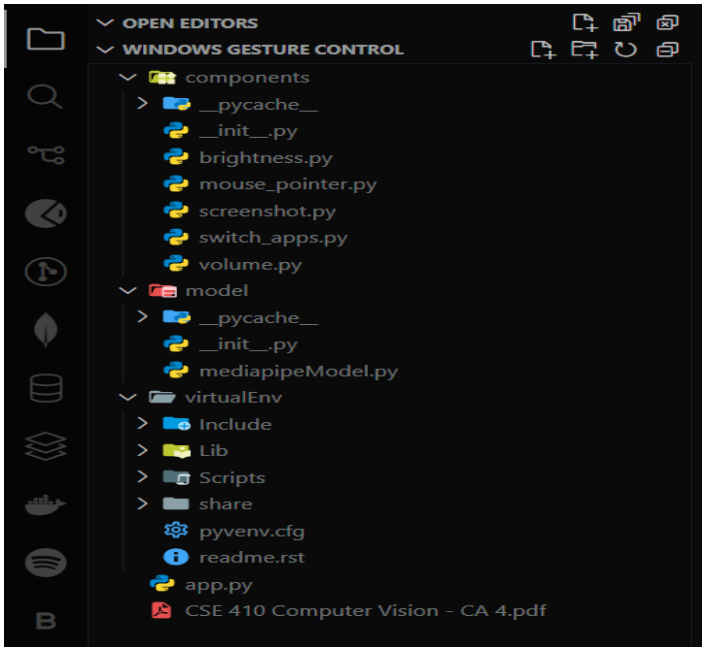
About the Project:

This Windows Gesture Control project uses a simple Tkinter interface to manage system functions like brightness, volume, mouse control, screenshots, and app switching via hand gestures. It integrates Mediapipe for gesture recognition, allowing users to interact with their computer hands-free through intuitive, customizable commands for seamless control.

Project Outlines:

- 1. Project Overview: The Windows Gesture Control project enables users to control system functions like brightness, volume, and app switching using gestures.
- 2. Technology Stack: It uses Python Open CV, Mediapipe for gesture recognition, Tkinter for the GUI.
- 3. Key Features: Includes gesture-based controls for brightness, mouse pointer, volume, screenshots, and application switching.
- 4. User Interface: A clean, responsive interface with buttons to activate different gesture controls, styled with a modern color palette.
- 5. Objective: To provide a hands-free, intuitive way to control , improving accessibility and user experience.

Project Directory Structure:



Codes and Python Scripts:

brightness.py

```
import cv2
from math import hypot
import screen_brightness_control as sbc
import numpy as np


def adjust_brightness(mpHands, hands, Draw):
    # Start capturing video from webcam
    cap = cv2.VideoCapture(0)

    while True:
        # Read video frame by frame
        _, frame = cap.read()

        # Flip image
        frame = cv2.flip(frame, 1)

        # Convert BGR image to RGB image
        frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Process the RGB image
        Process = hands.process(frameRGB)

        landmarkList = []
        # if hands are present in image(frame)
        if Process.multi_hand_landmarks:
            # detect handmarks
            for handlm in Process.multi_hand_landmarks:
                for _id, landmarks in enumerate(handlm.landmark):
                    # store height and width of image
                    height, width, color_channels = frame.shape

                    # calculate and append x, y coordinates
                    # of handmarks from image(frame) to lmList
                    x, y = int(landmarks.x * width), int(landmarks.y * height)
                    landmarkList.append([_id, x, y])

                # draw Landmarks
                Draw.draw_landmarks(frame, handlm, mpHands.HAND_CONNECTIONS)
```

```
        # If landmarks list is not empty
        if landmarkList != []:
            # store x,y coordinates of (tip of) thumb
            x_1, y_1 = landmarkList[4][1], landmarkList[4][2]

            # store x,y coordinates of (tip of) index finger
            x_2, y_2 = landmarkList[8][1], landmarkList[8][2]

            # draw circle on thumb and index finger tip
            cv2.circle(frame, (x_1, y_1), 7, (0, 255, 0), cv2.FILLED)
            cv2.circle(frame, (x_2, y_2), 7, (0, 255, 0), cv2.FILLED)

            # draw line from tip of thumb to tip of index finger
            cv2.line(frame, (x_1, y_1), (x_2, y_2), (0, 255, 0), 3)

            # calculate square root of the sum of
            # squares of the specified arguments.
            L = hypot(x_2 - x_1, y_2 - y_1)

            # 1-D linear interpolant to a function
            # with given discrete data points
            # (Hand range 15 - 220, Brightness
            # range 0 - 100), evaluated at length.
            b_level = np.interp(L, [15, 220], [0, 100])

            # set brightness
            sbc.set_brightness(int(b_level))

        # Display Video and when 'q' is entered, destroy
        # the window
        cv2.imshow("Image", frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break
```

Mouse_pointer.py

```
import cv2
from math import hypot
import pyautogui
import numpy as np


def move_pointer(mpHands, hands, Draw):
```

```
# Start capturing video from webcam
cap = cv2.VideoCapture(0)

# Get screen width and height
screen_width, screen_height = pyautogui.size()

# Start capturing video from webcam
while True:
    # Read video frame by frame
    _, frame = cap.read()

    # Flip image
    frame = cv2.flip(frame, 1)

    # Convert BGR image to RGB image
    frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Process the RGB image
    Process = hands.process(frameRGB)

    landmarkList = []
    # If hands are present in image(frame)
    if Process.multi_hand_landmarks:
        # Detect hand landmarks
        for handlm in Process.multi_hand_landmarks:
            for _id, landmarks in enumerate(handlm.landmark):
                # Store height and width of image
                height, width, color_channels = frame.shape

                # Calculate and append x, y coordinates of hand landmarks
                x, y = int(landmarks.x * width), int(landmarks.y * height)
                landmarkList.append([_id, x, y])

            # Draw Landmarks
            Draw.draw_landmarks(frame, handlm, mpHands.HAND_CONNECTIONS)

    # If landmarks list is not empty
    if landmarkList:
        # Store x,y coordinates of (tip of) thumb
        x_1, y_1 = landmarkList[4][1], landmarkList[4][2]

        # Store x,y coordinates of (tip of) index finger
        x_2, y_2 = landmarkList[8][1], landmarkList[8][2]
```

```
            # Draw circle on thumb and index finger tip
            cv2.circle(frame, (x_1, y_1), 7, (0, 255, 0), cv2.FILLED)
            cv2.circle(frame, (x_2, y_2), 7, (0, 255, 0), cv2.FILLED)

            # Draw line from tip of thumb to tip of index finger
            cv2.line(frame, (x_1, y_1), (x_2, y_2), (0, 255, 0), 3)

            # Calculate distance between thumb and index finger
            L = hypot(x_2 - x_1, y_2 - y_1)

            # Normalize coordinates to move the mouse
            mouse_x = np.interp(x_2, [0, frame.shape[1]], [0, screen_width])
            mouse_y = np.interp(y_2, [0, frame.shape[0]], [0, screen_height])

            # Move the mouse
            pyautogui.moveTo(mouse_x, mouse_y)

        # Display video and when 'q' is entered, destroy the window
        cv2.imshow("Image", frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

    # Release resources
    cap.release()
    cv2.destroyAllWindows()
```

screenshot.py

```
import cv2
import mediapipe as mp
from math import hypot
import pyautogui


def take_snapshot(mpHands, hands, Draw):
    # Start capturing video from webcam
    cap = cv2.VideoCapture(0)

    while True:
        # Read video frame by frame
        _, frame = cap.read()

        # Flip image
```

```
        frame = cv2.flip(frame, 1)

        # Convert BGR image to RGB image
        frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Process the RGB image
        Process = hands.process(frameRGB)

        landmarkList = []
        # If hands are present in image(frame)
        if Process.multi_hand_landmarks:
            # Detect hand landmarks
            for handlm in Process.multi_hand_landmarks:
                for _id, landmarks in enumerate(handlm.landmark):
                    # Store height and width of image
                    height, width, color_channels = frame.shape

                    # Calculate and append x, y coordinates of hand landmarks
                    x, y = int(landmarks.x * width), int(landmarks.y * height)
                    landmarkList.append([_id, x, y])

                # Draw Landmarks
                Draw.draw_landmarks(frame, handlm, mpHands.HAND_CONNECTIONS)

            # Check for specific gesture (thumbs up)
            if landmarkList:
                # Store y-coordinate of thumb and index finger
                thumb_y = landmarkList[4][2]
                index_y = landmarkList[8][2]

                # Gesture detection logic
                if thumb_y < index_y:  # Thumbs up if thumb is above index finger
                    # Take a screenshot
                    screenshot = pyautogui.screenshot()
                    screenshot.save("screenshot.png")
                    cv2.putText(
                        frame,
                        "Screenshot Taken!",
                        (10, 50),
                        cv2.FONT_HERSHEY_SIMPLEX,
                        1,
                        (0, 255, 0),
                        2,
                    )
```

```
        # Display video and when 'q' is entered, destroy the window
        cv2.imshow("Image", frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

    # Release resources
    cap.release()
    cv2.destroyAllWindows()
```

Switch_apps.py

```
# Importing Libraries
import cv2
import mediapipe as mp
import pyautogui
import time


def tab_switch(mpHands, hands, Draw):
    # Start capturing video from webcam
    cap = cv2.VideoCapture(0)

    # Variables to track previous x-coordinates and gesture states
    previous_x = None
    gesture_state = None  # To track the current gesture state

    while True:
        # Read video frame by frame
        _, frame = cap.read()

        # Flip image
        frame = cv2.flip(frame, 1)

        # Convert BGR image to RGB image
        frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Process the RGB image
        Process = hands.process(frameRGB)

        landmarkList = []
        # If hands are present in image(frame)
        if Process.multi_hand_landmarks:
            # Detect hand landmarks
```

```
        for handlm in Process.multi_hand_landmarks:
            for _id, landmarks in enumerate(handlm.landmark):
                # Store height and width of image
                height, width, color_channels = frame.shape

                # Calculate and append x, y coordinates of hand landmarks
                x = int(landmarks.x * width)
                y = int(landmarks.y * height)
                landmarkList.append([_id, x, y])

            # Draw Landmarks
            Draw.draw_landmarks(frame, handlm, mpHands.HAND_CONNECTIONS)

        # Check for specific gestures
        if landmarkList:
            # Get the x-coordinate of the index finger tip (Landmark 8)
            index_x = landmarkList[8][1]

            # If previous_x is None, set it to current x
            if previous_x is None:
                previous_x = index_x

            # Calculate the swipe distance
            swipe_distance = index_x - previous_x

            # Gesture state management
            if swipe_distance > 50 and gesture_state != "swipe_right":
                # Swipe left to right to switch applications (Alt+Tab)
                pyautogui.hotkey("alt", "tab")
                time.sleep(1)  # Delay to avoid multiple triggers
                cv2.putText(
                    frame,
                    "Switching Applications!",
                    (10, 50),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1,
                    (0, 255, 0),
                    2,
                )
                gesture_state = "swipe_right"  # Update gesture state

            # Reset gesture state if no significant movement is detected
            if abs(swipe_distance) < 30:
                gesture_state = None
```

```
                # Update previous_x with the current index finger position
                previous_x = index_x

        # Display video and when 'q' is entered, destroy the window
        cv2.imshow("Image", frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

    # Release resources
    cap.release()
    cv2.destroyAllWindows()
```

volume.py

```
import cv2
import numpy as np
from math import hypot
from ctypes import cast, POINTER
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume


def adjust_volume(mpHands, hands, Draw):
    # Initialize volume control
    devices = AudioUtilities.GetSpeakers()
    interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
    volume = cast(interface, POINTER(IAudioEndpointVolume))

    # Start capturing video from webcam
    cap = cv2.VideoCapture(0)

    while True:
        # Read video frame by frame
        _, frame = cap.read()

        # Flip image
        frame = cv2.flip(frame, 1)

        # Convert BGR image to RGB image
        frameRGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Process the RGB image
        Process = hands.process(frameRGB)
```

```
landmarkList = []
# if hands are present in image(frame)
if Process.multi_hand_landmarks:
    # detect handmarks
    for handlm in Process.multi_hand_landmarks:
        for _id, landmarks in enumerate(handlm.landmark):
            # store height and width of image
            height, width, color_channels = frame.shape

            # calculate and append x, y coordinates
            # of handmarks from image(frame) to lmList
            x, y = int(landmarks.x * width), int(landmarks.y * height)
            landmarkList.append([_id, x, y])

        # draw Landmarks
        Draw.draw_landmarks(frame, handlm, mpHands.HAND_CONNECTIONS)

# If landmarks list is not empty
if landmarkList:
    # store x,y coordinates of (tip of) thumb
    x_1, y_1 = landmarkList[4][1], landmarkList[4][2]

    # store x,y coordinates of (tip of) index finger
    x_2, y_2 = landmarkList[8][1], landmarkList[8][2]

    # draw circle on thumb and index finger tip
    cv2.circle(frame, (x_1, y_1), 7, (0, 255, 0), cv2.FILLED)
    cv2.circle(frame, (x_2, y_2), 7, (0, 255, 0), cv2.FILLED)

    # draw line from tip of thumb to tip of index finger
    cv2.line(frame, (x_1, y_1), (x_2, y_2), (0, 255, 0), 3)

    # calculate distance between thumb and index finger
    L = hypot(x_2 - x_1, y_2 - y_1)

    # Set volume level (0.0 - 1.0)
    v_level = np.interp(L, [15, 220], [0.0, 1.0])
    volume.SetMasterVolumeLevelScalar(v_level, None)

# Display Video and when 'q' is entered, destroy the window
cv2.imshow("Image", frame)
if cv2.waitKey(1) & 0xFF == ord("q"):
    break
```

```
    # Release resources
    cap.release()
    cv2.destroyAllWindows()
```

MediapipeModel.py

```
import mediapipe as mp

def mediapipe_model():
    mpHands = mp.solutions.hands
    hands = mpHands.Hands(
        static_image_mode=False,
        model_complexity=1,
        min_detection_confidence=0.75,
        min_tracking_confidence=0.75,
        max_num_hands=2,
    )

    Draw = mp.solutions.drawing_utils

    return mpHands, hands, Draw
```

app.py

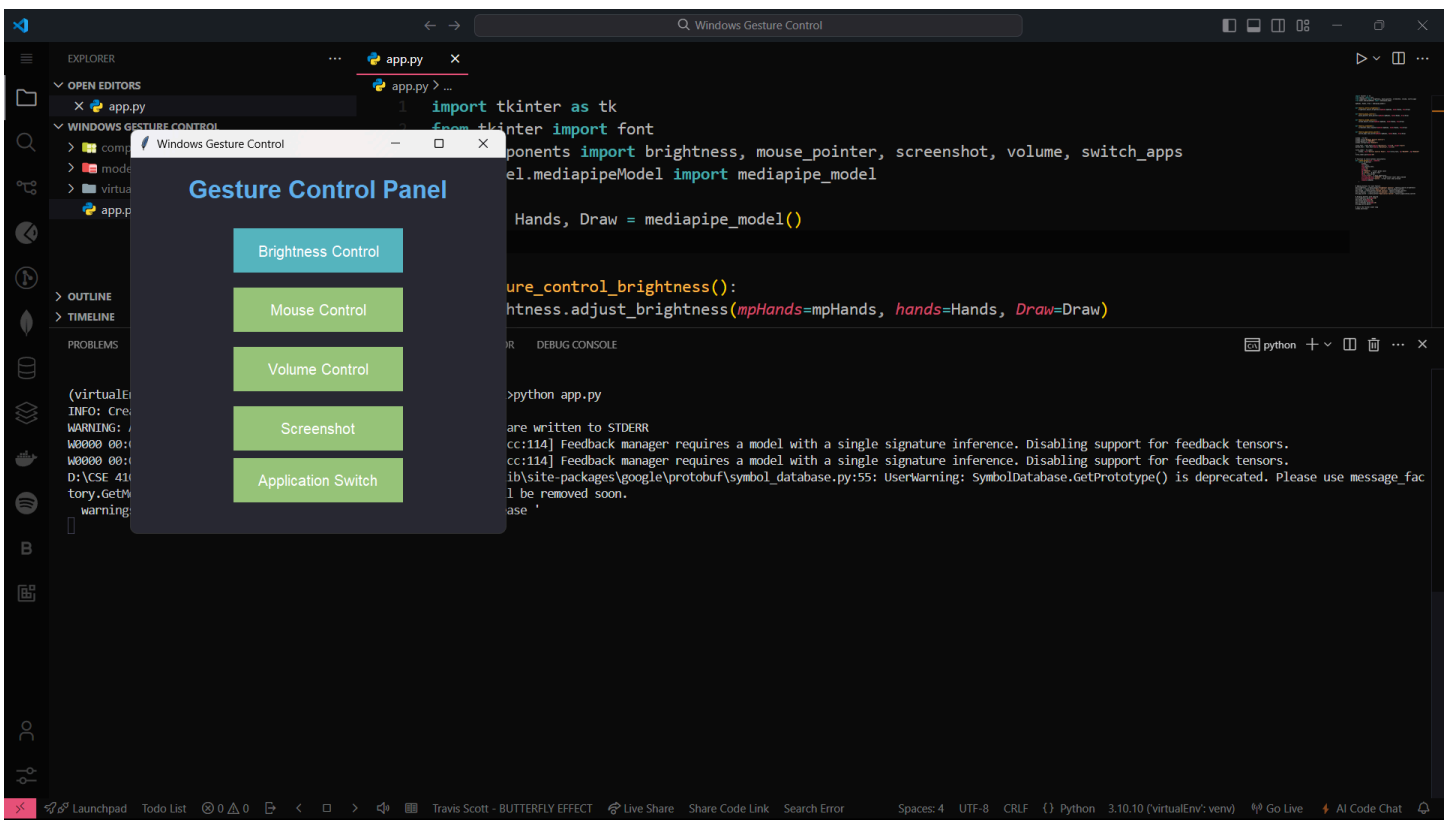
```
import tkinter as tk
from tkinter import font
from components import brightness, mouse_pointer, screenshot, volume, switch_apps
from model.mediapipeModel import mediapipe_model

mpHands, Hands, Draw = mediapipe_model()

def feature_control_brightness():
    brightness.adjust_brightness(mpHands=mpHands, hands=Hands, Draw=Draw)


def feature_mouse_control():
    mouse_pointer.move_pointer(mpHands=mpHands, hands=Hands, Draw=Draw)

def feature_volume_control():
    volume.adjust_volume(mpHands=mpHands, hands=Hands, Draw=Draw)
```

```
def feature_screenshot():
    screenshot.take_snapshot(mpHands=mpHands, hands=Hands, Draw=Draw)

def feature_application_switch():
    switch_apps.tab_switch(mpHands=mpHands, hands=Hands, Draw=Draw)


window = tk.Tk()
window.title("Windows Gesture Control")
window.geometry("500x500")
window.configure(bg="#282C34")

title_font = font.Font(family="Helvetica", size=20, weight="bold")
button_font = font.Font(family="Helvetica", size=12)

title_label = tk.Label(
    window, text="Gesture Control Panel", font=title_font, fg="#61AFEF", bg="#282C34"
)
title_label.pack(pady=20)

# Function to style buttons consistently
def create_button(text, command):
    return tk.Button(
        window,
        text=text,
        font=button_font,
        width=20,
        height=2,
        bg="#98C379",  # Soft green color
        fg="white",  # White text
        bd=0,  # No border
        relief="flat",  # Flat appearance
        activebackground="#56B6C2",  # Different color when pressed
        activeforeground="white",  # Text color when pressed
        command=command,
    )

# Adding buttons for each feature
btn_brightness = create_button("Brightness Control", feature_control_brightness)
btn_mouse = create_button("Mouse Control", feature_mouse_control)
btn_volume = create_button("Volume Control", feature_volume_control)
btn_screenshot = create_button("Screenshot", feature_screenshot)
btn_app_switch = create_button("Application Switch", feature_application_switch)
```

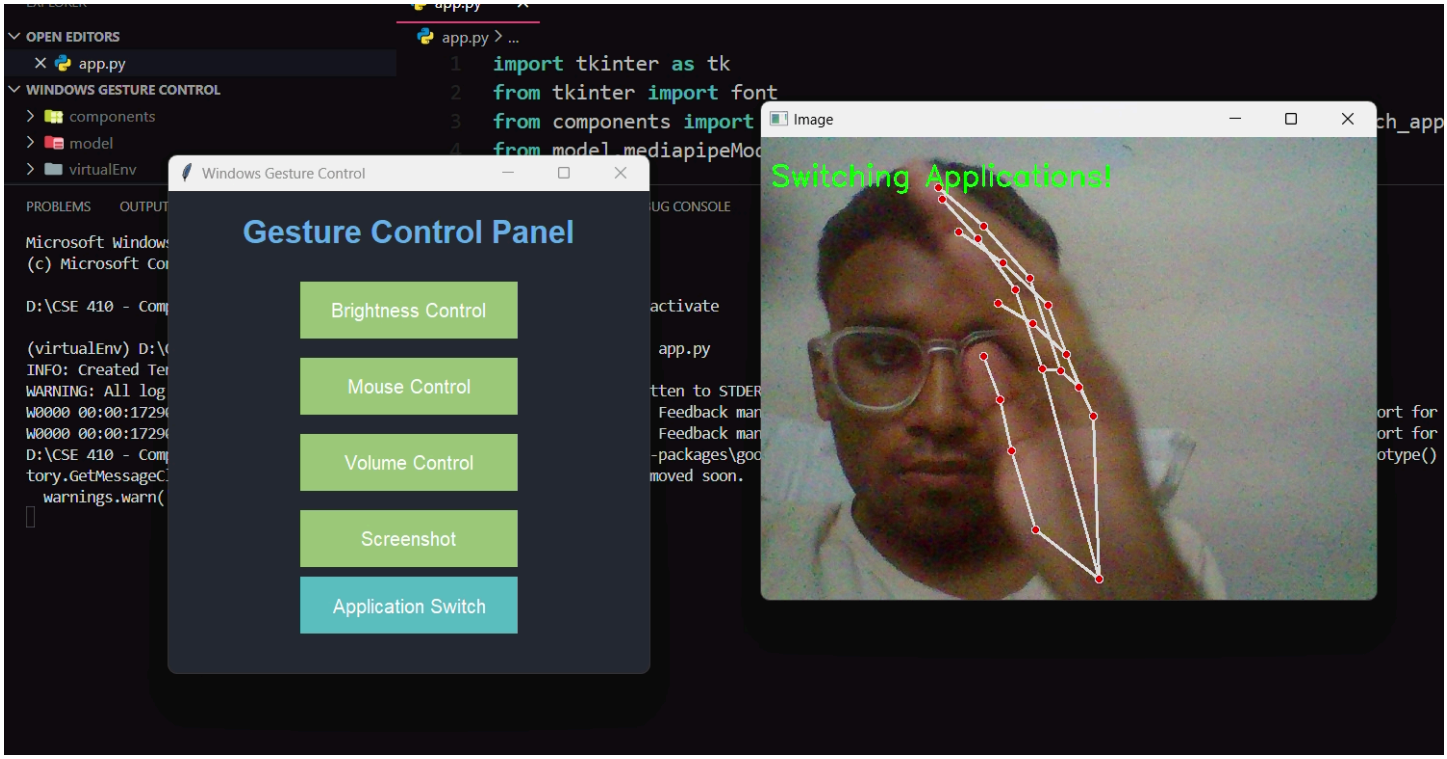
```python
# Display buttons with spacing
btn_brightness.pack(pady=10)
btn_mouse.pack(pady=10)
btn_volume.pack(pady=10)
btn_screenshot.pack(pady=10)
btn_app_switch.pack()

# Start the Tkinter event loop
window.mainloop()
```

Tkinter Application Outputs:

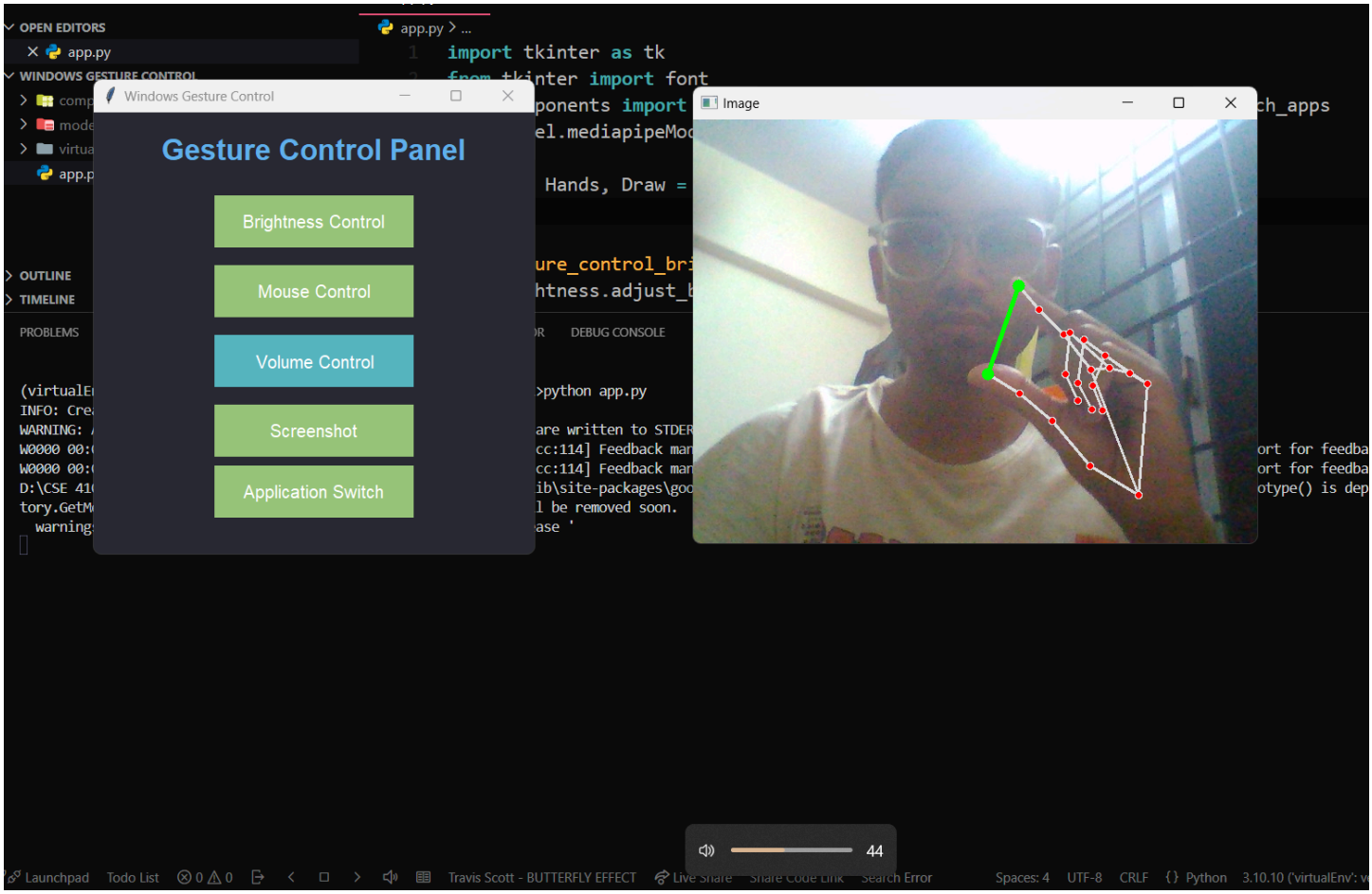
Tkinter GUI Application



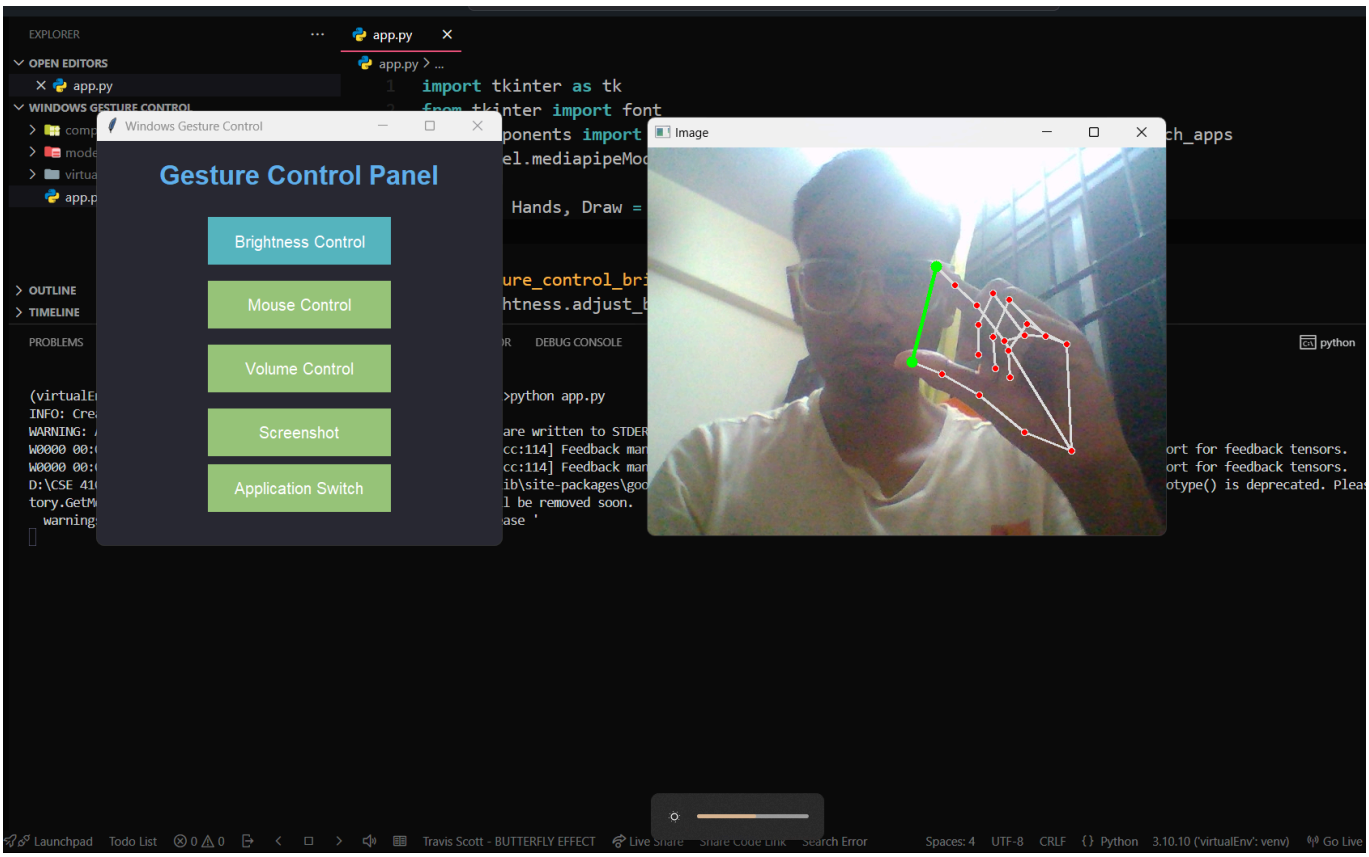
Switching Application



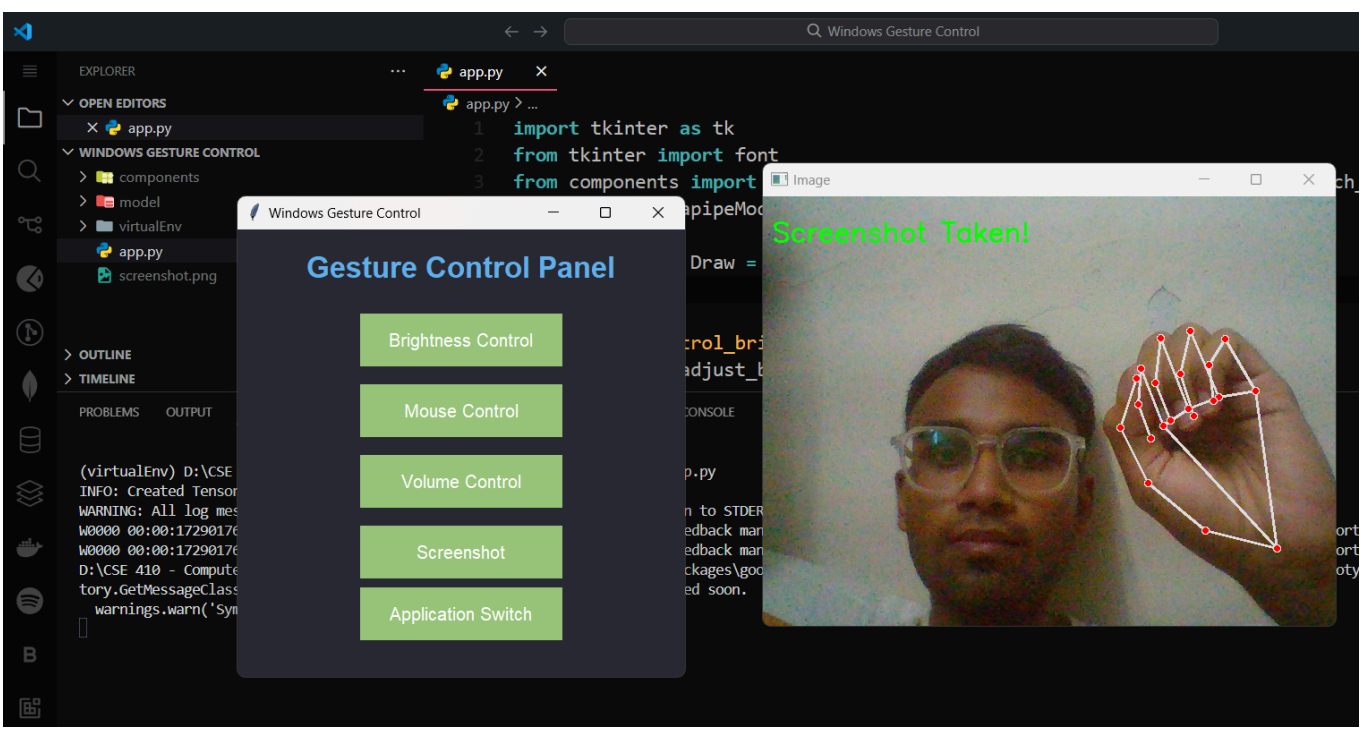
Controlling media Volume



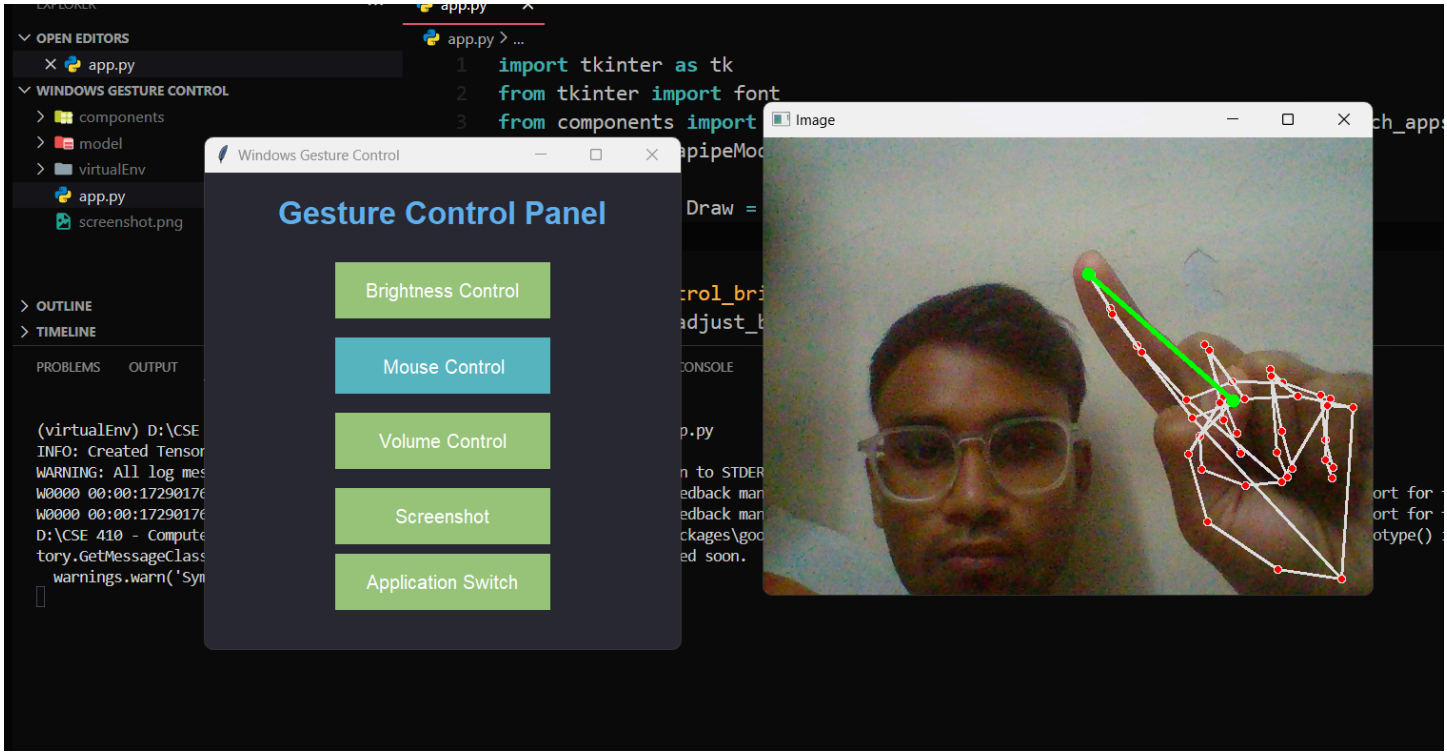
Controlling Screen Brightness



Taking Screenshot by Closing Fist



Controlling the Movement of Mouse Pointer



End of File
Thank You