****

**Superior University Lahore**

***Lab Task # 6***

**Name:** Muhammad Abdullah Kashif

**Roll No**: SU92-BSAIM-F23-070

**Instructor**: Sir Rasikh Ali

# Course: Programming for Artificial Intelligence (Lab)

**Face Profiling Using OpenCV and MediaPipe**

**Step-by-Step Explaination:**

**1. Import Required Libraries**

import cv2

import mediapipe as mp

import math

* cv2: Used for webcam access and drawing.
* mediapipe: Used to detect face landmarks.
* math: Used to compute distances using the Euclidean formula.

**2. Initialize MediaPipe Modules**

mp\_face\_mesh = mp.solutions.face\_mesh

mp\_drawing = mp.solutions.drawing\_utils

* mp\_face\_mesh: Loads the face mesh model.
* mp\_drawing: Helps draw landmarks if needed.

**📷 3. Open the Webcam**

cap = cv2.VideoCapture(0)

* Captures video from the default webcam (index 0).

**4. Configure the Face Mesh Detector**

with mp\_face\_mesh.FaceMesh(

    static\_image\_mode=False,

    max\_num\_faces=1,

    refine\_landmarks=True,

    min\_detection\_confidence=0.5,

    min\_tracking\_confidence=0.5) as face\_mesh:

* static\_image\_mode=False: It processes video continuously.
* max\_num\_faces=1: Only one face will be detected.
* refine\_landmarks=True: Enables high-accuracy landmark detection.
* min\_detection\_confidence=0.5: Confidence threshold for detection.
* min\_tracking\_confidence=0.5: Threshold for tracking landmarks.

**5. Capture Video Frame by Frame**

    while cap.isOpened():

        success, image = cap.read()

        if not success:

            break

* Continuously reads frames from the webcam.
* success: Indicates if the frame was successfully captured.

**6. Flip and Convert the Frame to RGB**

        image = cv2.flip(image, 1)

        rgb\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

        results = face\_mesh.process(rgb\_image)

* Flips the frame horizontally (mirror image).
* Converts BGR to RGB (MediaPipe uses RGB format).

**7. Detect Face Landmarks**

        image = cv2.flip(image, 1)

        rgb\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

        results = face\_mesh.process(rgb\_image)

* Processes the image and detects facial landmarks.

**8. Extract Facial Landmark Coordinates**

        if results.multi\_face\_landmarks:

            for face\_landmarks in results.multi\_face\_landmarks:

                ih, iw, \_ = image.shape

                def get\_point(index):

                    landmark = face\_landmarks.landmark[index]

                    return int(landmark.x \* iw), int(landmark.y \* ih)

This helper function:

* Converts normalized landmark positions (0 to 1) into image pixel values.

**9. Select Important Facial Points**

                left\_eye = get\_point(33)

                right\_eye = get\_point(263)

                nose\_top = get\_point(1)

                nose\_bottom = get\_point(2)

                left\_lip = get\_point(61)

                right\_lip = get\_point(291)

                left\_face = get\_point(234)

                right\_face = get\_point(454)

                chin = get\_point(152)

                forehead = get\_point(10)

* Points represent important landmarks needed for face profiling.

**10. Draw These Points on the Face**

                for point in important\_points:

                    cv2.circle(image, point, 3, (0, 255, 0), -1)

* Draws green dots on each important facial point.

**11. Calculate Distances Between Points**

                eye\_distance = math.hypot(right\_eye[0] - left\_eye[0], right\_eye[1] - left\_eye[1])

                nose\_length = math.hypot(nose\_bottom[0] - nose\_top[0], nose\_bottom[1] - nose\_top[1])

                lip\_width = math.hypot(right\_lip[0] - left\_lip[0], right\_lip[1] - left\_lip[1])

                face\_width = math.hypot(right\_face[0] - left\_face[0], right\_face[1] - left\_face[1])

                face\_height = math.hypot(chin[0] - forehead[0], chin[1] - forehead[1])

Uses the Euclidean distance formula to compute:

* Eye distance
* Nose length
* Lip width
* Face width and height

**12. Display Text Results on Screen**

                cv2.putText(image, f"Eye Distance: {int(eye\_distance)}px", (30, 30),

                            cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,255,0), 2)

                cv2.putText(image, f"Nose Length: {int(nose\_length)}px", (30, 60),

                            cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,255,0), 2)

                cv2.putText(image, f"Lip Width: {int(lip\_width)}px", (30, 90),

                            cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,255,0), 2)

                cv2.putText(image, f"Face Width: {int(face\_width)}px", (30, 120),

                            cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,255,0), 2)

                cv2.putText(image, f"Face Height: {int(face\_height)}px", (30, 150),

                            cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,255,0), 2)

...

* Writes the measurements on the screen in real-time.

**13. Show the Frame with Overlays**

cv2.imshow('Full Face Profiling', image)

* Displays the frame in a window named **"Full Face Profiling"**.

**14. Exit on ESC Key Press**

if cv2.waitKey(5) & 0xFF == 27:

            break

* Exits the loop if the ESC key is pressed.

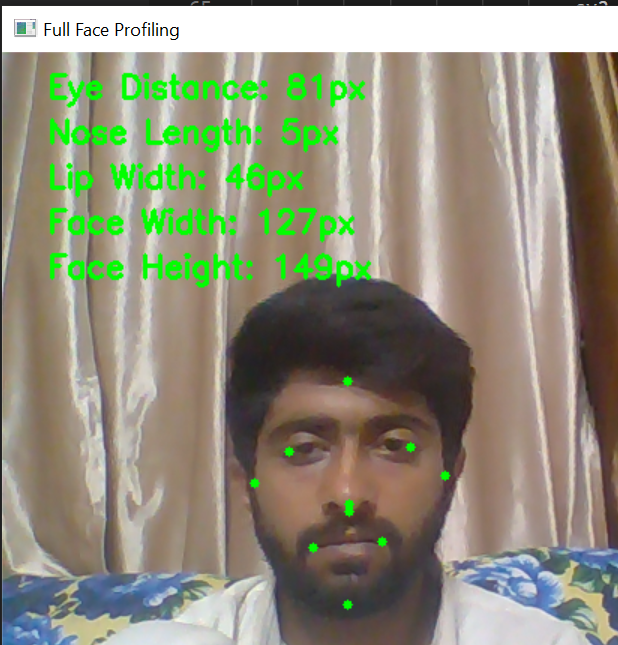
**15. Cleanup**

cap.release()

cv2.destroyAllWindows()

* Releases the camera and closes all OpenCV windows.

**Output Summary:**

****