LAB # 13

ML × AR -- GESTURE RECOGNITION

Performance Metric	Exceeds Expectations (5–4)	Meets Expectations (3–2)	Does Not Meet Expectations (0-1 marks)	Marks (out of 20)
Understanding of Concept (4 marks)	Demonstrates clear and in-depth understanding of theory; strongly relates it well to lab objectives.	Basic understanding of theory; provides minimal or partial connection to lab objectives.	Little or no understanding of concept; unclear or incorrect relevance to lab topic.	
Code Implementation (6 marks)	Code is well-structured, correct, error-free, and reflects the theoretical concepts; handles edge cases effectively.	Code works with minor errors or inefficiencies; shows partial understanding of the concept.	Code is incorrect, incomplete, or does not align with lab goals.	
Use of Programming Features/Tools (4 marks)	Efficiently applies relevant Python libraries, data processing methods, and ML techniques (e.g., NumPy, Pandas, scikit-learn, Matplotlib) to achieve the desired outcomes.	Uses standard/basic functions and logic; limited or partial use of available features/tools.	Minimal or incorrect use of tools; relies on trivial constructs or hard-coded approaches.	
Results and Report (6 marks)	Produces accurate and well-presented results (visualizations, metrics, comparisons) with clear interpretation and insights.	Results are partially correct or lack clarity in interpretation; report is adequate but lacks depth, formatting, or coherence.	Results are missing, incorrect, or poorly explained.	

LAB # 13

ML × AR -- GESTURE RECOGNITION

OBJECTIVE

To understand how Machine Learning models can recognize hand gestures and trigger Augmented Reality overlays or animations using the Meta Quest Pro and Wizard app.

Equipment & Tools:

- Meta Quest Pro (in AR passthrough mode)
- Wizard app (for gesture capture and AR visualization)
- Laptop with Wizard dashboard access (optional)

LAB TASKS

Task 1 — Introduction to ML in AR (Concept Demo)

- Watch a short demo in Wizard showing how different gestures (e.g., open hand, fist, pointing) trigger various AR overlays.
- Discuss how the ML model behind the scene classifies gestures based on hand keypoints captured by Quest sensors.

Task 2 — Gesture Recording (Data Collection)

- Open Wizard's "Gesture Capture" template.
- Record **three gestures**:
    1. Open hand
    2. Fist
    3. Pointing
- Collect **10 samples** per gesture from different students.
- Observe how Wizard visualizes hand keypoints.

Task 3 — Model Training (Conceptual Overview)

- The Wizard app automatically trains a simple ML classifier on the captured gestures.
- Discuss how ML learns from numeric features (hand coordinates) rather than raw video frames.
- Observe model accuracy or confusion matrix displayed in Wizard.

Task 4 — AR Interaction Demo

- Use the trained model live:
    - Perform "Open hand" → A holographic menu appears.
    - Perform "Fist" → The menu closes.
    - Perform "Pointing" → A virtual arrow highlights a nearby 3D object.
- Note how the app responds in real-time.

Task 5 — Reflection Discussion

- How does the ML model decide which gesture is which?

- What could cause misclassification (lighting, angles, incomplete samples)?
- How could this system be used in healthcare, education, or robotics?

Deliverables:

- Short report or reflection (1 page) including:
  - Screenshots of gestures in Wizard
  - Summary of AR response
  - Explanation of how ML connects to AR behavior

LAB OUTCOMES

By completing this lab, students will:

- Describe how gesture recognition uses ML classification techniques.
- Demonstrate how recognized gestures can control AR elements (e.g., showing text, animation, or object color changes).
- Explain how real-world data (hand positions/poses) are collected and used to train ML models.

SOLVED LAB :

Lab12-13 > main12.py > ...

```python
import cv2
import mediapipe as mp
import numpy as np

mp_hands = mp.solutions.hands
mp_draw = mp.solutions.drawing_utils
hands = mp_hands.Hands(
    static_image_mode=False,
    max_num_hands=1,
    min_detection_confidence=0.7,
    min_tracking_confidence=0.7
)

cap = cv2.VideoCapture(0)

def finger_is_open(tip, pip):
    return tip.y < pip.y

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(rgb)

    gesture = "No Hand"

    if results.multi_hand_landmarks:
        for hand in results.multi_hand_landmarks:
            lm = hand.landmark

            # Finger states
            thumb = lm[4].x > lm[3].x
            index = finger_is_open(lm[8], lm[6])
            middle = finger_is_open(lm[12], lm[10])
            ring = finger_is_open(lm[16], lm[14])
            pinky = finger_is_open(lm[20], lm[18])

            fingers = [thumb, index, middle, ring, pinky]

```

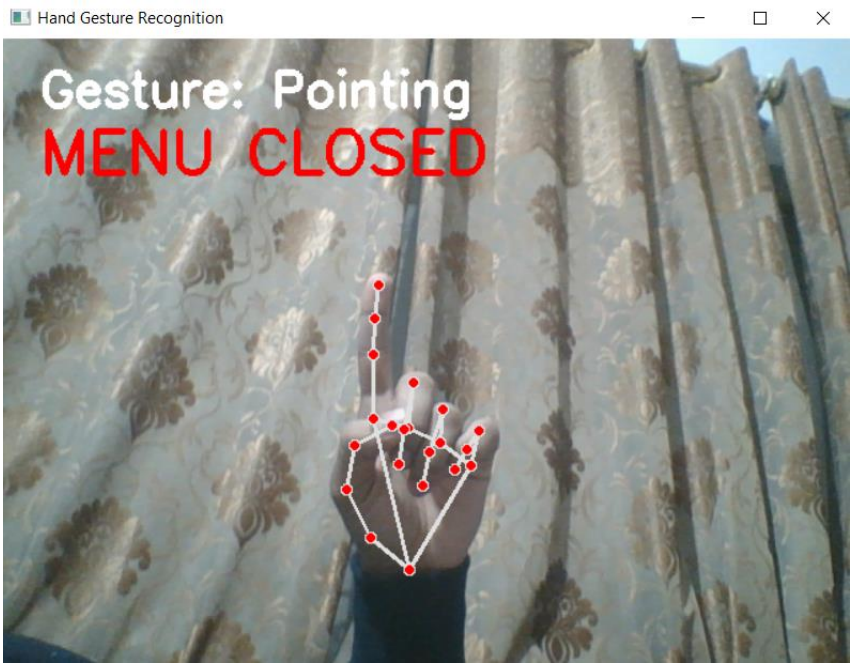
```
            # Gesture logic
            if all(fingers):
                gesture = "Open Hand"
                cv2.rectangle(frame, (50, 100), (400, 200), (0, 255, 0), -1)
                cv2.putText(frame, "MENU OPEN", (80, 160),
                            cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0, 0, 0), 3)

            elif not any(fingers):
                gesture = "Fist"
                cv2.putText(frame, "MENU CLOSED", (80, 160),
                            cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0, 0, 255), 3)

            elif index and not middle and not ring and not pinky:
                gesture = "Pointing"
                cv2.arrowedLine(frame, (300, 300), (500, 200),
                                (255, 0, 0), 5)

            mp_draw.draw_landmarks(frame, hand, mp_hands.HAND_CONNECTIONS)

    cv2.putText(frame, f"Gesture: {gesture}", (30, 50),
                cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 255), 3)

    cv2.imshow("Hand Gesture Recognition (CV)", frame)

    if cv2.waitKey(1) & 0xFF == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

OUTPUT :

1. Open hand

2. Fist

3. Pointing


Hand Gesture Recognition
Gesture: Pointing
MENU CLOSED