# 🧠 Real-Time Gesture Recognition using Mediapipe & OpenCV

### 🔍 Overview

This project implements a **real-time gesture recognition system** using a webcam. It identifies human gestures like waving, pointing, thumbs up/down, and arms crossed using pose estimation from **Mediapipe**, webcam handling with **OpenCV**, and logs gesture data to **SQLite** and **CSV** for analysis.

---

### ⚙️ Dependencies

Install the following Python libraries:

pip install opencv-python mediapipe numpy

*Note:* sqlite3 is part of Python's standard library.

---

### 📁 Project Structure

```
.
├── gesture_recognition.py     # Real-time gesture detection
├── pose_tracking.db           # SQLite database
├── pose_logs.csv              # CSV logs
└── README.md                  # Project documentation
```

---

### 📸 Key Modules and Their Responsibilities

**1. gesture_recognition.py — Core Script**

#### 🔄 Main Pipeline

```
import cv2
import mediapipe as mp
import numpy as np
import sqlite3
import csv
from datetime import datetime
```

- **Webcam input**: Captured using OpenCV.

- **Pose detection**: Handled by Mediapipe's Pose solution.
- **Gesture logic**: Based on angles between key landmarks.
- **Logging**: Simultaneously saves to SQLite and CSV.

---

### 🔢 2. Angle Calculation

Used to differentiate between gestures.

```
def calculate_angle(a, b, c):
    a, b, c = np.array(a), np.array(b), np.array(c)
    radians = np.arctan2(c[1]-b[1], c[0]-b[0]) - np.arctan2(a[1]-b[
    angle = np.abs(radians * 180.0 / np.pi)
    return 360 - angle if angle > 180 else angle
```

**Explanation**:

- Computes the angle at joint b formed by points a, b, and c.
- Used for **elbow**, **shoulder**, and **knee** angles.

---

### ✋ 3. Gesture Detection Logic

```
def detect_gestures(landmarks):
    left_shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.v
                     landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.v
    left_elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].
                  landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].
    left_wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].
                  landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].

    elbow_angle = calculate_angle(left_shoulder, left_elbow, left_w

    if elbow_angle < 90 and left_wrist[1] < left_elbow[1]:
        return "Thumbs Up", elbow_angle
    elif elbow_angle < 90 and left_wrist[1] > left_elbow[1]:
        return "Thumbs Down", elbow_angle
    elif elbow_angle > 160:
        return "Pointing", elbow_angle
```

```
    elif elbow_angle > 160:
        return "Pointing", elbow_angle
    elif left_wrist[1] < left_shoulder[1]:
        return "Wave", elbow_angle
    else:
        return "Unknown", elbow_angle
```

return "Wave", elbow_angle

else:

return "Unknown", elbow_angle

**Explanation**:

- Compares y-coordinates to determine vertical position.
- Uses elbow angles to classify gestures.

---

### 📹 4. Real-Time Pose Tracking & Visualization

```
cap = cv2.VideoCapture(0)
mp_pose = mp.solutions.pose
pose = mp_pose.Pose()
mp_drawing = mp.solutions.drawing_utils
```

Initializes webcam and pose model.

- Begins live capture from webcam.

---

### 📜 5. Logging Detected Gestures

### ✅ SQLite Setup

```
conn = sqlite3.connect('pose_tracking.db')
cursor = conn.cursor()
cursor.execute('''
CREATE TABLE IF NOT EXISTS PoseData (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp TEXT,
    elbow_angle REAL,
    shoulder_angle REAL,
    knee_angle REAL,
    gesture TEXT
)
''')
```

✅ **CSV Setup**

```
csvfile = open('pose_logs.csv', 'a', newline='')
csvwriter = csv.writer(csvfile)
csvwriter.writerow(["Timestamp", "Elbow Angle", "Shoulder Angle", "
```

📝 **Insert Log Entries**

```
timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
cursor.execute("INSERT INTO PoseData (timestamp, elbow_angle, shoul
               (timestamp, elbow_angle, shoulder_angle, knee_angle,
csvwriter.writerow([timestamp, elbow_angle, shoulder_angle, knee_an
conn.commit()
```

---

🖼️ **6. Drawing Results on Video Frame**

```
cv2.putText(image, f"Gesture: {gesture}", (50, 50),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.PO
```

Displays gesture label.

- Draws pose skeleton for visual feedback.

---

✅ **Supported Gestures**

| Gesture | Criteria |
| --- | --- |
| Wave | Wrist above shoulder |
| Pointing | Arm extended (elbow > 160°) |
| Thumbs Up | Wrist below elbow, elbow < 90° |
| Thumbs Down | Wrist above elbow, elbow < 90° |
| Arms Crossed | Wrists overlap on chest |

---

### ▶️ Running the Application

**1. Start Gesture Recognition**

```
python gesture_recognition.py
```

**2. View Database Contents**

```
sqlite3 pose_tracking.db
SELECT * FROM PoseData;
```

**3. Check CSV Logs**

```
cat pose_logs.csv
```

---

### 🍼 Sample Output Log

```
Timestamp,Elbow Angle,Shoulder Angle,Knee Angle,Gesture
2025-05-26 18:00:01,45.2,120.5,178.0,Thumbs Up
2025-05-26 18:00:02,170.3,150.1,176.2,Pointing
```

---

### ❗ Troubleshooting

| Problem | Fix |
| --- | --- |
| cv2.VideoCapture(1) fails | Try index 0 or check webcam permissions |
| No pose landmarks | Adjust lighting and background contrast |
| Unstable detection | Add prediction smoothing using a gesture history queue |

---

### 🔮 Future Work

- Multi-person pose and gesture support

- Use **Mediapipe Hands** for finer gestures
- Gesture-controlled app interfaces (slideshows, games, etc.)
- Add **gesture history queue** with majority voting

---

## 🙏 Credits

- Mediapipe Pose Estimation
- OpenCV
- NumPy