Comprehensive Tutorial: Advanced Human Pose Tracking & Gesture Recognition System

1. Introduction
This tutorial explains how to build an advanced human pose tracking and gesture recognition system using MediaPipe Pose, OpenCV, and SQLite. The system detects body landmarks, calculates joint angles, recognizes gestures, and stores data for further analysis or integration into larger human-computer interaction (HCI) applications.

---

2. System Overview
The system performs the following key functions:

1. **Real-Time Pose Tracking: Detects 33 body landmarks using MediaPipe Pose.**
2. **Joint Angle Calculation: Computes angles at key joints (elbows, shoulders, knees) to understand posture.**
3. **Gesture Recognition: Identifies specific static gestures such as wave, pointing, thumbs up/down, and arms crossed.**
4. **Data Storage: Logs pose data and recognized gestures into an SQLite database and CSV file for long-term storage and analysis.**
5. **Visual Feedback: Displays real-time pose landmarks and annotated gestures overlaid on camera feed.**

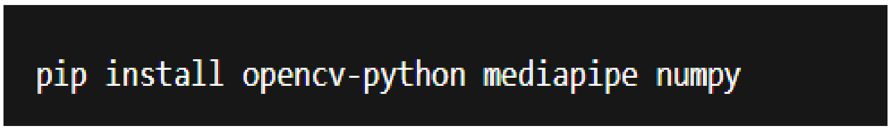
---

3. Prerequisites
3.1 Software & Libraries

- **Python 3.8+: Chosen for its mature ecosystem and compatibility with libraries.**
- **OpenCV (cv2): Used for real-time video frame capture and rendering. Considered alternatives like Pillow and scikit-image, but OpenCV provides unparalleled real-time video support and integration with NumPy arrays.**
- **MediaPipe: Selected for its efficient and lightweight pose estimation pipeline with CPU/GPU support. Alternatives like OpenPose were considered, but MediaPipe is easier to install, faster on limited hardware, and well-integrated with Python.**
- **NumPy: Used for vector math and angle calculations. It is fast and optimized for numerical computations.**
- **SQLite3: Provides lightweight, serverless structured data storage. Chosen over heavier solutions like PostgreSQL or MySQL for its simplicity and local storage capability.**
- **CSV Module: Allows export of pose and gesture logs for use in spreadsheet programs or additional data processing pipelines.**
- **Datetime: Required for accurate timestamping of events and logs.**

3.2 Hardware Requirements

- **Webcam: A standard USB camera or built-in laptop webcam is sufficient.**
- **Optional GPU: Enhances processing speed, especially when dealing with high-resolution input or multiple video streams.**

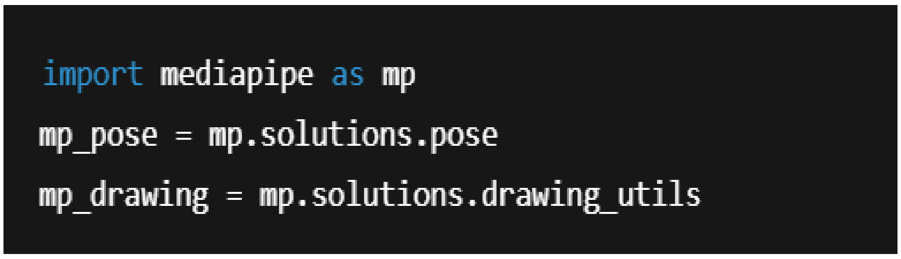


```
pip install opencv-python mediapipe numpy
```

4. Step-by-Step Explanation

4.1 Initial Setup

1. **MediaPipe Initialization:**
    - **mp_pose = mp.solutions.pose: Loads pre-trained pose detection models.**
    - **mp_drawing = mp.solutions.drawing_utils: Enables easy landmark visualization.**



```
import mediapipe as mp
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
```

2. **Database Setup:**
    - **Creates pose_tracking.db and initializes a table to store timestamps, joint angles, and gestures.**
3. **CSV Logging:**
    - **Initializes a pose_logs.csv with headers: Timestamp, Elbow Angle, Shoulder Angle, Knee Angle, Gesture.**
4. **Video Capture:**
    - **cv2.VideoCapture(2): Opens a camera stream. Camera index may need adjustment.**

4.2 Core Functions

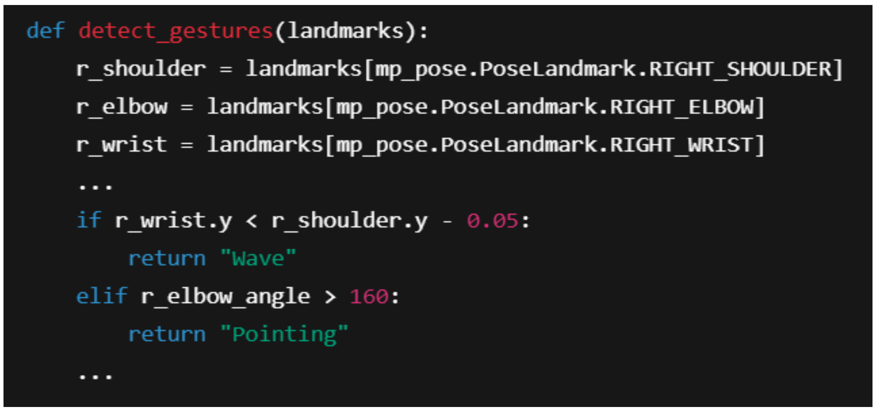
calculate_angle(a, b, c)

- **Purpose: Determines the angle formed by three body landmarks.**
- **Input: MediaPipe landmark objects a, b, and c.**
- **Method:**
    - **Converts points to vectors.**
    - **Uses the dot product formula to calculate the angle between vectors:**

$$angle = arccos\left(\frac{\vec{ab}\cdot\vec{bc}}{\|ab\|\,\|bc\|}\right)$$

detect_gestures(landmarks)

- **Logic:**
    1. **Extracts key landmarks: shoulders, elbows, wrists.**
    2. **Calculates joint angles.**
    3. **Evaluates gesture conditions:**
        - **Wave: Wrists above shoulders.**
        - **Pointing: Extended arm (elbow angle > 160°).**
        - **Thumbs Up/Down: Wrist position relative to elbows.**
        - **Arms Crossed: Wrists cross the body's centerline.**
- **Output: Gesture label.**



```python
def detect_gestures(landmarks):
    r_shoulder = landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER]
    r_elbow = landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW]
    r_wrist = landmarks[mp_pose.PoseLandmark.RIGHT_WRIST]
    ...
    if r_wrist.y < r_shoulder.y - 0.05:
        return "Wave"
    elif r_elbow_angle > 160:
        return "Pointing"
    ...
```

4.3 Main Processing Loop

1. **Frame Capture:**
   - Reads frames from webcam.
   - Converts BGR to RGB (required for MediaPipe).
2. **Pose Detection:**
   - Processes RGB frame to extract body landmarks.
3. **Gesture Recognition:**
   - Calls detect_gestures().
   - Stabilizes results using a 10-frame history buffer and majority vote to reduce jitter.
4. **Data Logging:**
   - Inserts current data (angles + gesture + timestamp) into both SQLite and CSV.
5. **Visualization:**
   - Draws landmarks and pose skeleton.
   - Displays gesture label on the video feed.
6. **Exit Mechanism:**
   - Press Q to terminate the loop and safely close resources.

---

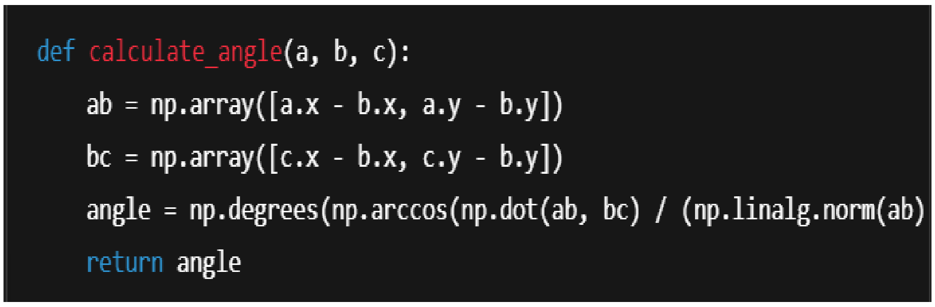
5. Key Concepts Explained

5.1 MediaPipe Pose

- Tracks 33 2D/3D landmarks per person.
- Runs efficiently on CPUs (~30 FPS).
- Core landmarks used include:
  - RIGHT/LEFT SHOULDER, ELBOW, WRIST

5.2 Angle Calculation

- Essential for understanding limb orientation.
- Used in classification logic to infer static gestures.



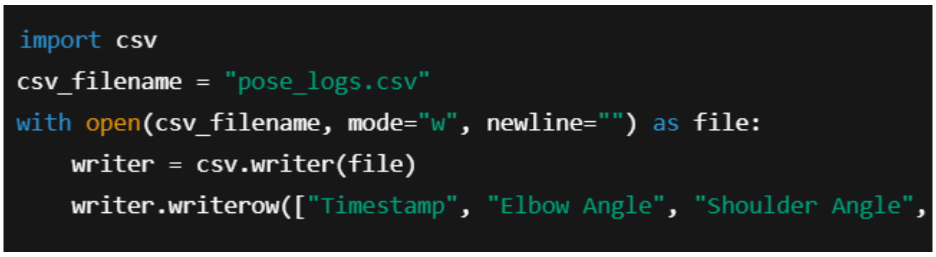
```python
def calculate_angle(a, b, c):
    ab = np.array([a.x - b.x, a.y - b.y])
    bc = np.array([c.x - b.x, c.y - b.y])
    angle = np.degrees(np.arccos(np.dot(ab, bc) / (np.linalg.norm(ab)
    return angle
```

5.3 Gesture Stabilization

- Reduces noisy detection with temporal smoothing.
- Uses majority vote from a 10-frame history buffer for robust results.

5.4 Data Storage

- SQLite: Enables structured queries, such as "Show all instances of 'Thumbs Up'."
- CSV: Ideal for export, visualization, and analysis in tools like Excel or Pandas.



```
import csv
csv_filename = "pose_logs.csv"
with open(csv_filename, mode="w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["Timestamp", "Elbow Angle", "Shoulder Angle",
```

---

6. Potential Improvements

1. More Gestures:
   - Include "Clapping", "Hand Raise", sign language basics.
2. 3D Pose Estimation:
   - Leverage depth coordinates from MediaPipe for enhanced accuracy.
3. Custom ML Model:
   - Use a classifier (e.g., Random Forest or small neural network) trained on angle and position features to recognize complex dynamic gestures.
4. Real-Time Alerts:
   - Integrate sound or visual alerts when specific gestures are detected (e.g., for safety or assistance).
5. Cloud Integration:
   - Sync data across devices using Firebase, AWS, or MQTT brokers.

---

7. Conclusion

This system showcases how pose estimation and gesture recognition technologies can be combined for real-time, interactive applications. It:

- Tracks body movements using lightweight frameworks.
- Recognizes intuitive static gestures.
- Logs data for offline or real-time analysis.

By using a robust pipeline of MediaPipe, OpenCV, and SQLite, this system serves as a powerful base for developing intelligent human-computer interfaces, fitness trackers, surveillance systems, and interactive educational tools..

Final Code Repository: https://github.com/thinkrobotics

Demonstration:

```
    writer.writerow(["Timestamp", "Elbow Angle", "Shoulder Angle", "Knee An

def calculate_angle(a, b, c):
    ab = np.array([a.x - b.x, a.y - b.y])
    bc = np.array([c.x - b.x, c.y - b.y])
    dot_product = np.dot(ab, bc)
    mag_ab = np.linalg.norm(ab)
    mag_bc = np.linalg.norm(bc)
    angle = np.degrees(np.arccos(dot_product / (mag_ab * mag_bc)))
    return angle

def detect_gestures(landmarks):
    r_shoulder = landmarks[mp_pose.PoseLandmark.RIGHT_SH
    r_elbow = landmarks[mp_pose.PoseLandmark.RIGHT_ELBO
    r_wrist = landmarks[mp_pose.PoseLandmark.RIGHT_WRIS
    l_shoulder = landmarks[mp_pose.PoseLandmark.LEFT_SH
    l_elbow = landmarks[mp_pose.PoseLandmark.LEFT_ELBO
    l_wrist = landmarks[mp_pose.PoseLandmark.LEFT_WRIST

    r_elbow_angle = calculate_angle(r_shoulder, r_elbow,
    l_elbow_angle = calculate_angle(l_shoulder, l_elbow,
    wrist_distance = math.sqrt((r_wrist.x - l_wrist.x)

    if r_wrist.y < r_shoulder.y - 0.1 and l_wrist.y < l_
        return "Wave"
    elif r_elbow_angle > 150 and r_wrist.y < r_shoulder.
        return "Pointing"
    elif wrist_distance < 0.05
        return "Clap"
    elif r_wrist.y < r_elbow.y and l_wrist.y < l_elbow.y
        return "Thumbs Up"
    elif r_wrist.x < l_shoulder.x and l_wrist.x > r_shoul
        return "Arms Crossed"
    return "None"

cap = cv2.VideoCapture(0)
```