Pose Detection and Evaluation System

1. Introduction

The provided code implements a **pose detection system** that uses **MediaPipe** and **OpenCV** to detect and evaluate human poses. The system can analyze the pose of a person either from a **video file** or a **live camera feed**. It compares the detected poses with predefined **ground truth landmarks** and calculates the pose's error, allowing the system to test the accuracy of the detected pose.

---

2. Core Functionality

The system performs the following primary tasks:

1. **Pose Detection**: Detects human body pose in an input frame (from a video or live camera).
2. **Ground Truth Comparison**: Compares detected landmarks with predefined ground truth coordinates.
3. **Pose Calculation**: Calculates pose landmarks in real-time from a given video or webcam.
4. **Error Calculation**: Computes error metrics (Euclidean distance) between detected landmarks and ground truth.
5. **Result Visualization**: Draws annotations on the detected pose, including the comparison with ground truth.
6. **File Output**: Saves the processed video, frame images, and error metrics in CSV format.

---

3. Pose Detection Using MediaPipe

- **MediaPipe** is used to detect human poses in each frame. The code initializes MediaPipe with a **high complexity model**, which offers more precise results but is computationally more expensive.
- `pose = mp_pose.Pose(static_image_mode=False, model_complexity=2, min_detection_confidence=0.7, min_tracking_confidence=0.7)`
- This configuration enables **real-time tracking** of human poses. The `min_detection_confidence` and `min_tracking_confidence` ensure that the pose is detected with a confidence of at least **70%** before proceeding with further calculations.
- The function `process_frame(frame)` converts each frame from the video or webcam feed into an RGB image and then passes it to MediaPipe's pose detector. The detected **pose landmarks** (e.g., shoulders, elbows, knees) are returned by the model.

---

4. Testing Pose Against Ground Truth

The system evaluates the accuracy of the pose detection by comparing detected landmarks with predefined **ground truth landmarks**. Ground truth represents the expected positions of key joints (e.g., shoulders, elbows, knees) in normalized image coordinates (ranging from 0 to 1).

- **GT_LANDMARKS**: This dictionary contains the predefined ground truth coordinates for specific body parts:
- `GT_LANDMARKS = {`
- `    'LEFT_SHOULDER': [0.35, 0.20],`
- `    'RIGHT_SHOULDER': [0.65, 0.20],`
- `    'LEFT_ELBOW': [0.25, 0.40],`
- `    'RIGHT_ELBOW': [0.75, 0.40],`
- `    'LEFT_KNEE': [0.35, 0.75],`
- `    'RIGHT_KNEE': [0.65, 0.75]`
- `}`
- **Ground Truth Comparison**: The system iterates through each landmark in `GT_LANDMARKS`, and for each landmark:
    - Retrieves the detected position from the MediaPipe model.
    - Calculates the **Euclidean distance** between the detected coordinates and the ground truth coordinates.

  This calculation provides an **error metric** for each landmark, which is a measure of how accurate the detected pose is compared to the expected ground truth.

---

5. Pose Calculation from Video and Live Camera Input

The system supports two modes of input:

1. **Video File Input**: A pre-recorded video file can be used to evaluate the poses. The video path is provided as an input, and the video is processed frame by frame.
2. `def process_video_file(input_path=default_input_video, output_path=default_output_video, save_csv=True):`
    - For each frame, the system uses `process_frame()` to detect the pose and compare it to ground truth.
    - The processed frame is annotated with the detected pose and the calculated errors, which is then written to an output video file.
3. **Live Camera Input**: The system can also analyze real-time poses from a webcam feed. It uses OpenCV to capture frames from the camera, then processes each frame using the same `process_frame()` function. The video feed is displayed in a window and optionally recorded.
4. `def process_live_camera(output_path="webcam_output.mp4", duration=30):`

---

6. Error Calculation

For each frame, after detecting the pose, the system computes the **error** (Euclidean distance) for each landmark. The formula used to calculate the Euclidean distance between two points (predicted and ground truth) is:

Error=(xpred−xgt)2+(ypred−ygt)2\text{Error} = \sqrt{(x_{\text{pred}} - x_{\text{gt}})^2 + (y_{\text{pred}} - y_{\text{gt}})^2}

Where:

- xpredx_{\text{pred}} and ypredy_{\text{pred}} are the predicted (detected) coordinates of a landmark.
- xgtx_{\text{gt}} and ygty_{\text{gt}} are the ground truth coordinates of the same landmark.

The errors are then visualized on the video, where:

- **Red circles** represent ground truth landmarks.
- **Green circles** represent detected landmarks.
- **Yellow lines** are drawn to show the error between ground truth and detected landmarks.

---

7. File Output and Visualization

1. **Annotated Video**: The processed video is saved with all the landmarks drawn on the frames. The video is stored in the specified output path.

   Example:

   ```
   C:\Users\laiba mazhar\Downloads\output_video_running.mp4
   ```

2. **Error Metrics**: The error values (Euclidean distances) for each landmark are saved in a CSV file. This allows for further analysis of the pose detection accuracy over time.

   Example:

   ```
   C:\Users\laiba mazhar\Downloads\pose_errors_running.csv
   ```

3. **Frames**: Every 10th frame is saved as an individual image for debugging and visual inspection.
4. **Real-time Video Display**: While processing a video or live camera feed, the annotated frames are shown in a window. The user can press the **Esc** key to stop the live feed.

---

8. Conclusion

This system successfully achieves the following tasks:

1. **Testing Pose Against Ground Truth**: The system compares the detected pose landmarks with predefined ground truth landmarks and calculates the error for each key body part.
2. **Pose Calculation from Video and Live Camera Input**: The system processes both pre-recorded video files and real-time webcam feeds to detect and analyze human poses in real-time.

The combination of **pose detection**, **error calculation**, and **visual feedback** allows for a comprehensive evaluation of pose accuracy. This system could be useful in applications such as sports analysis, physical therapy monitoring, and motion capture for animation.

Output is attached with the assignment.

```
"C:\Users\laiba mazhar\PycharmProjects\pythonProject3\.venv\Scripts\python.exe" "C:\Users\laiba mazhar\PycharmProjects\pythonProject3\hi.py"
Pose Detection System
1. Process video file
2. Live camera detection
Enter choice (1/2): INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
1

Processing complete!
Saved processed video to: C:\Users\laiba mazhar\Downloads\output_video_running.mp4
Saved error metrics to: pose_errors.csv

Process finished with exit code 0
```

```
"C:\Users\laiba mazhar\PycharmProjects\pythonProject3\.venv\Scripts\python.exe" "C:\Users\laiba mazhar\PycharmProjects\pythonProject3\hi.py"
Pose Detection System
1. Process video file
2. Live camera detection
Enter choice (1/2): INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
2
Record output? (y/n): y
Recording duration (seconds): 7
Saved webcam recording to webcam_output.mp4

Process finished with exit code 0
```

```
import cv2
import mediapipe as mp
import numpy as np
import time
import csv
```

```
import os

# Initialize MediaPipe
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
pose = mp_pose.Pose(static_image_mode=False,
                    model_complexity=2,
                    min_detection_confidence=0.7,
                    min_tracking_confidence=0.7)


GT_LANDMARKS = {
    'LEFT_SHOULDER': [0.35, 0.20],
    'RIGHT_SHOULDER': [0.65, 0.20],
    'LEFT_ELBOW': [0.25, 0.40],
    'RIGHT_ELBOW': [0.75, 0.40],

    'LEFT_KNEE': [0.35, 0.75],
    'RIGHT_KNEE': [0.65, 0.75]
}


# Hardcoded default paths
default_input_video = r"C:\Users\laiba mazhar\Downloads\running_video.mp4"
default_output_video = r"C:\Users\laiba
mazhar\Downloads\output_video_running.mp4"

def process_frame(frame, ground_truth=None):
    image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = pose.process(image_rgb)

    error_data = {}

    if results.pose_landmarks:
        mp_drawing.draw_landmarks(
            frame,
            results.pose_landmarks,
            mp_pose.POSE_CONNECTIONS,
            mp_drawing.DrawingSpec(color=(0, 255, 0), thickness=2,
circle_radius=2),
            mp_drawing.DrawingSpec(color=(255, 0, 0), thickness=2)
        )

        if ground_truth:
            for landmark_name, gt_coords in ground_truth.items():
                try:
                    landmark = mp_pose.PoseLandmark[landmark_name].value
                    detected = results.pose_landmarks.landmark[landmark]

                    pred_coords = [detected.x, detected.y]
                    error = np.linalg.norm(np.array(pred_coords) -
np.array(gt_coords))
                    error_data[landmark_name] = error

                    h, w = frame.shape[:2]
                    gt_pixel = (int(gt_coords[0] * w), int(gt_coords[1] * h))
                    pred_pixel = (int(detected.x * w), int(detected.y * h))
```

```
                    cv2.circle(frame, gt_pixel, 10, (0, 0, 255), -1)
                    cv2.circle(frame, pred_pixel, 8, (0, 255, 0), -1)
                    cv2.line(frame, gt_pixel, pred_pixel, (255, 255, 0), 2)

                    cv2.putText(frame, f"{landmark_name}: {error:.3f}",
                                (gt_pixel[0] + 15, gt_pixel[1]),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 200, 255),
2)
                except KeyError:
                    print(f"Landmark {landmark_name} not found in MediaPipe")

    return frame, error_data


def save_errors_to_csv(errors, filename=r"C:\Users\laiba
mazhar\Downloads\pose_errors_running.csv"):
    with open(filename, mode='a', newline='') as file:
        writer = csv.writer(file)
        if file.tell() == 0:
            writer.writerow(["timestamp"] + list(errors.keys()))
        writer.writerow([time.time()] + list(errors.values()))


def process_video_file(input_path=default_input_video,
output_path=default_output_video, save_csv=True):
    if not os.path.exists(input_path):
        print(f"Error: File not found at {input_path}")
        return

    cap = cv2.VideoCapture(input_path)
    if not cap.isOpened():
        print(f"Error opening video file {input_path}")
        return

    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)

    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        processed_frame, errors = process_frame(frame, GT_LANDMARKS)
        out.write(processed_frame)

        if save_csv and errors:
            save_errors_to_csv(errors)

        if frame_count % 10 == 0:
            frame_path = f"frame_{frame_count:04d}.jpg"
            cv2.imwrite(frame_path, processed_frame)
```

```
        frame_count += 1

        cv2.imshow('Video Analysis', processed_frame)
        if cv2.waitKey(1) & 0xFF == 27:
            break

    cap.release()
    out.release()
    print(f"\nProcessing complete!")
    print(f"Saved processed video to: {os.path.abspath(output_path)}")
    if save_csv:
        print(f"Saved error metrics to: pose_errors.csv")


def process_live_camera(output_path="webcam_output.mp4", duration=30):
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error opening webcam")
        return

    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    if output_path:
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(output_path, fourcc, 20.0, (width, height))

    start_time = time.time()
    while cap.isOpened() and (time.time() - start_time) < duration:
        ret, frame = cap.read()
        if not ret:
            break

        processed_frame, _ = process_frame(frame)

        if output_path:
            out.write(processed_frame)

        cv2.imshow('Live Pose Detection', processed_frame)
        if cv2.waitKey(1) & 0xFF == 27:
            break

    cap.release()
    if output_path:
        out.release()
        print(f"Saved webcam recording to {output_path}")


if __name__ == "__main__":
    print("Pose Detection System")
    print("1. Process video file")
    print("2. Live camera detection")
    choice = input("Enter choice (1/2): ")

    if choice == "1":
        process_video_file()
```

```
    elif choice == "2":
        record = input("Record output? (y/n): ").lower() == 'y'
        output_name = "webcam_output.mp4" if record else None
        duration = int(input("Recording duration (seconds): ")) if record
else 30
        process_live_camera(output_name, duration)
    else:
        print("Invalid choice")

    cv2.destroyAllWindows()
```