Project Title	**Human Pose Estimation using MediaPipe**
Course/Module	Computer Networks
Date	November 2025
Submitted By	Ayusi parida(23bhi10004)

2. Introduction

This project implements a **real-time vision** task—**Human Pose Estimation**—using Google's **MediaPipe** framework, specifically the Pose solution. Pose estimation involves identifying and tracking key anatomical joints (landmarks) of a person's body. The goal is to provide a foundational demonstration of a computer vision pipeline capable of processing static images to extract quantifiable body data. The core implementation utilizes Python, OpenCV for image handling, and MediaPipe for machine learning inference.

3. Problem Statement

The problem is to accurately and efficiently detect the **33 key anatomical landmarks** on a human figure within a static image. The solution must then visualize these landmarks and the connections between them, as well as extract and present the numerical pixel coordinates for subsequent analysis (e.g., calculating joint distances).

4. Functional Requirements

- **R1: Image Loading:** Must be able to load an image file (e.g., PNG/JPG) into memory using OpenCV.
- **R2: Pose Detection:** Must utilize the MediaPipe Pose model to detect 33 human body landmarks on the loaded image.
- **R3: Visualization:** Must draw the detected landmarks and their connections (skeleton) onto the original image.
- **R4: Coordinate Extraction:** Must be able to access the normalized coordinates (x, y, z) of specific landmarks (e.g., wrists) and convert them to absolute pixel coordinates.
- **R5: Numerical Analysis:** Must perform a simple calculation based on the extracted coordinates (e.g., distance between two joints).

5. Non-functional Requirements

- **NFR1: Performance:** The processing time for a single static image should be minimal (though the code is not optimized for real-time video, the underlying framework is capable of it).
- **NFR2: Accuracy:** The landmark detection should be robust and accurate for clear, single-person images.
- **NFR3: Portability:** The solution should run reliably in a cloud environment (Google Colab) using specified library versions (mediapipe==0.10.14).
- **NFR4: Extensibility:** The code structure should allow for easy extension to video processing or the integration of custom gesture logic.

6. System Architecture

The system follows a simple **client-side processing architecture** within the execution environment (Google Colab/Local Machine).

- **Input Layer:** A static image file (.png).
- **Processing Layer:**
    - **OpenCV (cv2):** Handles image reading (cv2.imread) and color space conversion (BGR $\to$ RGB).
    - **MediaPipe (mp.solutions.pose):** The core ML framework for pose estimation inference.
    - **NumPy (np):** Used for numerical calculations (e.g., distance formula).
- **Output Layer:**
    - Annotated image displayed via Matplotlib (matplotlib.pyplot).
    - Textual output (coordinates and calculated distance) printed to the console.

7. Design Diagrams

Since this project is a single-script ML inference task without multi-user interaction, external APIs, or a database, a full set of traditional UML diagrams (Use Case, ER) is not strictly necessary. We will focus on the **Workflow** and **Component** diagrams.

Workflow Diagram

This diagram illustrates the step-by-step process flow of the application.

Component Diagram

This diagram shows the main software components and their dependencies.

8. Design Decisions & Rationale

Design Decision	**Rationale**
MediaPipe Pose	**High performance** and **pre-trained accuracy** for body landmark detection, optimized by Google for real-time tasks.
static_image_mode=True	This setting is crucial for a single static image, as it tells the model to run detection on every image, increasing reliability for images where the person might be far from the center, as opposed to video mode which relies on tracking.
BGR to RGB Conversion	OpenCV reads images in **BGR** format, but MediaPipe and Matplotlib often expect **RGB**. The explicit conversion (cv2.cvtColor(image, cv2.COLOR_BGR2RGB)) ensures correct processing and visualization.
Normalized Coordinates	MediaPipe provides coordinates (x, y) between **0.0 and 1.0**. This design choice makes the output resolution-independent. The application

Design Decision	Rationale
	converts them to pixel values by multiplying by image width (W) and height (H) for use in specific image contexts.

---

9. Implementation Details

The implementation is structured into three main blocks:

1. **Setup and Utilities:** Installs necessary libraries (mediapipe, opencv-contrib-python), defines helper functions (display_image), and performs initial imports.
2. **Part 1: Pose Estimation:** Loads the image, initializes mp.solutions.pose and mp.solutions.drawing_utils. The core is the with mp_pose.Pose(...) as pose: context, which ensures resources are managed correctly.
3. **Part 1.2: Processing & Drawing:** The image is processed via results = pose.process(image_rgb). The mp_drawing.draw_landmarks function then handles the complex visualization by using the predefined mp_pose.POSE_CONNECTIONS to connect the 33 points.
4. Part 1.3: Coordinate Access: Landmarks are accessed using their symbolic name (e.g., mp_pose.PoseLandmark.LEFT_WRIST). Pixel coordinates are calculated as:

10. Screenshots / Results

A successful execution would produce two main visual outputs:

1. Original Image: The loaded image of the athlete.

2. Annotated Image: The original image with the 33 landmarks (blue dots) and the skeletal connections (green lines) drawn over the subject.

Textual Results Example:

Left Wrist Pixel Coordinates: (x=..., y=...)

Right Wrist Pixel Coordinates: (x=..., y=...)

Pixel distance between wrists: ... pixels

11. Testing Approach

The project was tested using a single, clear test case:

- **Test Case:** Successful loading and processing of an image with a single human subject.
- **Expected Result:**

1. The annotated image is displayed with all 33 pose landmarks correctly placed on the body's joints.
2. The console output provides the correct pixel coordinates for the Left Wrist and Right Wrist.
3. The calculated pixel distance between the wrists is plausible.

- **Validation:** Visual inspection of the annotated image confirms the accuracy of the landmark detection. Code inspection confirms the correct formula for coordinate conversion and distance calculation is used.

12. Challenges Faced

1. **Coordinate System Conversion:** The necessity to convert from OpenCV's BGR to MediaPipe's/Matplotlib's RGB format was a minor challenge, requiring the explicit cv2.cvtColor.
2. **Normalized Coordinates:** Understanding that MediaPipe's (x, y, z) are normalized (0.0 to 1.0) and require multiplication by image dimensions (W, H) to obtain absolute pixel values was a key learning point.
3. **Library Dependency Management:** Pinning the versions (numpy"<2", mediapipe==0.10.14) was a necessary step to resolve potential compatibility issues in the Colab environment.

13. Learnings & Key Takeaways

- **MediaPipe Efficiency:** MediaPipe is a highly efficient, high-level wrapper for complex computer vision models, significantly simplifying the implementation of pose estimation.
- **Computer Vision Pipeline:** The process involves a standard pipeline: Load $\to$ Pre-process (Color Convert) $\to$ Inference $\to$ Post-process (Draw/Extract).
- **Data Extraction:** The true power of vision models lies not just in visualization, but in the ability to extract quantitative, measurable data (coordinates) for downstream applications (e.g., biomechanics, gesture control).

14. Future Enhancements

1. **Video Processing:** Adapt the code from static_image_mode=True to process live video feed or pre-recorded videos, enabling real-time gesture recognition.
2. **Custom Gesture Recognition:** Implement logic (as hinted by the wrist coordinates example) to recognize complex gestures (e.g., "T-pose", "Hands-up") by analyzing the relative positions and angles of landmarks.
3. **3D Visualization:** Utilize the z-coordinate (depth) to render the pose in 3D for a more comprehensive analysis.

4. **Error Handling:** Implement more robust error handling for images where the subject is partially obscured or not present.

15. References

- Google MediaPipe Documentation for Pose.
- OpenCV Documentation for image handling.