****

Capstone Project Report

On

# “MediaPipe Face and Body Detection Analysis”

**Project No.-3**

A capstone project report submitted in partial fulfillment of the requirement for

**AI Development Associate**

Submitted by

**Manik**

**Reg.no. - 1781867**

Under the guidance of

**Dr. Shivlok Singh**

**Principal Technical Officer**

NATIONAL INSTITUE OF ELECTRONICS AND INFORMATION TECHNOLOGY

Delhi Centre, Kakardooma office

**CERTIFICATE**

# This is to certify that the capstone project titled “MediaPipe Face and Body Detection Analysis”, designated as Project No. 3, has been undertaken by Manik, a Bonafide student of the AI Development Associate Course (January 2025 batch) at the National Institute of Electronics & Information Technology (NIELIT).

The project work has been carried out in partial fulfillment of the requirements for the six-month AI Development Associate Course.

This project report has been reviewed and approved, meeting the academic standards and requirements prescribed for the said course.

**Project Guide:** Dr. Shivlok Singh  
**Designation:** Principal Technical Officer  
**Institution:** NIELIT, Kakardooma

**DECLARATION**

# I hereby declare that the Capstone Project titled “MediaPipe Face and Body Detection Analysis” is the result of my own work carried out during the course of the AI Development Associate Course at the National Institute of Electronics & Information Technology (NIELIT).

This project was completed under the supervision of **Dr. Shivlok Singh** and the conclusions presented herein are based on the research and implementation conducted solely by me.

I further certify that this work is original and has not been submitted, either in part or in full, to any other institution or university for the award of any degree, diploma, or certificate in India or abroad.

I have followed all academic guidelines and ethical practices prescribed by the institute. Where information from other sources has been used, appropriate credit has been given through in-text citations and references.

**Name:** Manik  
**Course:** AI Development Associate Course  
**Institute:** NIELIT

**ACKNOWLEDGEMENT**

# I would like to express my sincere gratitude to all those who supported and guided me throughout the completion of my Capstone Project titled “MediaPipe Face and Body Detection Analysis” First and foremost, I am deeply thankful to Dr. Shivlok Singh, my project guide, for their valuable insights, constant encouragement, and expert supervision throughout the duration of this project. Their guidance helped me stay focused and motivated during each phase of the work.

I extend my heartfelt thanks to my peers and friends **Ms. Akansha** and **Ms. Bhawna** for their constructive feedback, technical discussions, and moral support which enriched my work.

This project has been a tremendous learning experience and has significantly contributed to my knowledge and skills in the field of Artificial Intelligence.

**Manik**  
AI Development Associate   
NIELIT

### **Abstract**

This project explores the application of computer vision techniques to perform **real-time face and body detection** using webcam video streams. The system leverages **MediaPipe**, a lightweight machine learning framework by Google, alongside **OpenCV** for video processing and visualization. The dataset in this context consists of continuous video frames captured live, containing facial and bodily features of individuals.

To address the detection task, two models from MediaPipe were implemented: **Face Detection** for identifying human faces with bounding boxes, and **Pose Estimation** for detecting body landmarks and deriving an approximate body bounding box. The frames are processed sequentially, and detected regions are visualized with labeled rectangles drawn using OpenCV for clarity.

A user-friendly real-time application was developed, capturing webcam video, applying both detection pipelines, and displaying the results instantaneously. The system architecture follows a modular design with separate methods for face detection, body detection, and drawing utilities. Graphical insights are generated directly on the video output, providing intuitive visualization of the detected faces and bodies.

This project demonstrates how **computer vision frameworks like MediaPipe can enable efficient, CPU-based face and body detection systems** suitable for surveillance, fitness tracking, and human-computer interaction applications. It highlights the importance of lightweight model deployment, real-time performance, and code modularity for building practical AI solutions in modern software development.

| **Title** | **Page No.** |
| --- | --- |
| Certificate | i |
| Declaration | ii |
| Acknowledgement | iii |
| Abstract | iv |
| **Phase 1: Introduction** | **1-7** |
| 1.1 General | 1 |
| 1.2 Background of the Study | 2 |
| 1.3 Importance of Computer Vision in Modern Applications | 3 |
| 1.4 Problem Statement | 4 |
| 1.5 Objective of the Project | 5 |
| 1.6 Summary | 6 |
| 1.7 AI Project Cycle | 7 |
| **Phase 2: Literature Review** | **8-11** |
| 2.1 Overview of Computer Vision Techniques | 8 |
| 2.2 Face and Body Detection Methods | 9 |
| 2.3 Identified Gaps and Proposed Solution | 11 |
| **Phase 3: Problem Statement and Scope** | **12-15** |
| 3.1 Core Challenge | 12 |
| 3.2 Problem Overview | 13 |
| 3.3 Project Scope | 13 |
| 3.4 Deployment Objective | 14 |
| 3.5 Challenges Addressed | 15 |
| **Phase 4: System Design and Description** | **16-20** |
| 4.1 System Overview | 16 |
| 4.2 Module Description | 17 |
| 4.3 Workflow and Pipeline | 19 |
| 4.4 Initial Observations | 20 |
| **Phase 5: Implementation Details** | **21-27** |
| 5.1 Installation Requirements | 21 |
| 5.2 Code Structure Overview | 23 |
| 5.3 Face Detection Implementation | 24 |
| 5.4 Body Detection Implementation | 25 |
| 5.5 Drawing and Display Functions | 26 |
| **Phase 6: Testing and Results** | **28-34** |
| 6.1 Testing Setup | 28 |
| 6.2 Output Screenshots | 30 |
| 6.3 Performance and Accuracy Discussion | 33 |
| 6.4 Limitations and Observations | 34 |
| **Phase 7: System Evaluation & Discussion** | **35-41** |
| 7.1 Evaluation Parameters | 35 |
| 7.2 Performance Analysis | 37 |
| 7.3 Sample Output Interpretation | 39 |
| 7.4 Overall System Verdict | 41 |
| **Phase 8: Application Deployment** | **42-48** |
| 8.1 Real-Time Application Design | 42 |
| 8.2 User Interaction Flow | 44 |
| 8.3 Deployment Benefits | 46 |
| 8.4 Future Enhancements for Deployment | 48 |
| **Phase 9: Conclusion & Future Scope** | **49-51** |
| 9.1 Conclusion | 49 |
| 9.2 Future Scope | 51 |
| **Appendix** | **52-55** |
| Full Code Listing | 52 |
| Additional Screenshots and Outputs | 54 |
| **References** | **56** |

#### **Phase 1: Introduction**

#### **1.1 General**

In today’s era of rapid technological advancements, **computer vision** has emerged as a significant field of artificial intelligence that enables machines to interpret and process visual data from the world. Among its various applications, **face and body detection** are essential tasks used extensively in surveillance systems, security authentication, healthcare, and human-computer interaction. Detecting faces allows systems to identify individuals, while detecting body landmarks helps in posture estimation, gesture recognition, and fitness tracking.

This project focuses on building a **real-time face and body detection system** using **MediaPipe** and **OpenCV**. MediaPipe, developed by Google, is a versatile framework offering pre-trained solutions for perception tasks, including face detection and pose estimation. OpenCV, an open-source computer vision library, is widely used for image processing and visualization tasks. Combining these tools allows for efficient detection and display of results with minimal computational requirements.

The system captures live video frames using a webcam, processes each frame to detect human faces and body landmarks, and visualizes the results by drawing labeled bounding boxes. This ensures the project achieves both **practical implementation skills** and an understanding of **lightweight AI frameworks** for real-world applications.

#### **1.2 Background of the Study**

The ability of machines to perceive and interpret human features has been a major focus area in the field of artificial intelligence. **Face detection** is one of the most fundamental computer vision tasks and serves as a precursor to facial recognition, emotion analysis, and user authentication systems. Similarly, **body detection and pose estimation** enable machines to understand human postures, gestures, and movements, which are critical in applications such as fitness monitoring, physiotherapy, animation, and augmented reality.

Traditionally, these tasks required complex deep learning models with high computational demands, often necessitating dedicated GPU resources for real-time performance. However, with the advancement of frameworks like **MediaPipe**, it is now possible to achieve real-time detection efficiently even on devices with limited hardware resources. MediaPipe offers optimized and lightweight solutions for multiple vision tasks, including face detection and pose estimation, making it an ideal choice for student projects and prototyping AI-based applications.

This study leverages **MediaPipe’s face detection and pose modules** along with **OpenCV** to implement a system capable of detecting faces and body landmarks from live webcam feeds. The project builds on fundamental concepts of computer vision and machine learning, integrating them into a practical application that demonstrates the potential of AI in enhancing automated perception and interaction systems.

#### **1.3 Importance of Computer Vision in Modern Applications**

**Computer vision** plays a transformative role in numerous modern applications by enabling machines to analyze and understand visual information similar to human perception. Its importance is evident across diverse domains:

* **Security and Surveillance:**  
  Automated face and body detection systems are used in CCTV surveillance to identify individuals, detect unauthorized access, and ensure public safety with minimal human intervention.
* **Healthcare and Fitness:**  
  Pose estimation and body landmark detection are widely applied in physiotherapy monitoring, exercise posture correction, and virtual fitness coaching, enhancing health outcomes with real-time feedback.
* **Human-Computer Interaction:**  
  Computer vision enables gesture recognition and face tracking, allowing for intuitive interfaces in gaming, augmented reality (AR), and virtual reality (VR) applications.
* **Retail and Marketing Analytics:**  
  Vision-based systems analyze customer behavior, track footfall, and optimize store layouts to improve sales and customer experience.
* **Automotive Industry:**  
  Advanced Driver Assistance Systems (ADAS) use vision algorithms for driver monitoring, drowsiness detection, pedestrian recognition, and lane detection to ensure road safety.
* **Education and Research:**  
  In academic settings, computer vision projects enhance students’ practical understanding of AI concepts, fostering innovation and readiness for industry challenges.

#### **1.4 Problem Statement**

Despite significant advancements in artificial intelligence and computer vision, implementing **real-time face and body detection systems** often remains challenging due to computational limitations and model complexity. Traditional deep learning-based detection models require high processing power, typically involving GPU acceleration, making them unsuitable for deployment on standard CPUs or embedded devices.

The problem addressed in this project is:

**How to design and implement a lightweight, efficient, and real-time system for detecting human faces and body landmarks from webcam video feeds using accessible frameworks suitable for normal CPUs?**

Key challenges associated with this problem include:

* Ensuring **high detection speed and minimal lag** to achieve real-time performance.
* Maintaining **acceptable accuracy** for both face detection and body pose estimation in varying lighting and background conditions.
* Integrating multiple detection modules efficiently within a single pipeline.
* Visualizing the detected features clearly for user interpretation and potential application development.

This project aims to overcome these challenges by utilizing **MediaPipe**, known for its optimized perception models, in combination with **OpenCV** for video capture and result visualization, thereby creating a robust solution for real-time face and body detection applications.

#### **1.5 Objective of the Project**

The primary objective of this project is to **design, implement, and demonstrate a real-time face and body detection system** using **MediaPipe** and **OpenCV** that operates efficiently on a standard CPU without requiring high-end computational resources.

Specific objectives include:

1. **To develop a system that captures live video frames** from a webcam in real-time.
2. **To integrate MediaPipe’s face detection module** for identifying and locating human faces within each video frame accurately.
3. **To implement MediaPipe’s pose estimation module** to detect body landmarks and derive an approximate bounding box for the human body.
4. **To visualize detection results** by drawing labeled bounding boxes around detected faces and bodies using OpenCV drawing utilities.
5. **To ensure minimal processing delay**, achieving real-time performance suitable for practical applications such as surveillance, fitness monitoring, and gesture-based interfaces.
6. **To gain practical understanding of lightweight AI frameworks** and their integration for building effective computer vision applications.

#### **1.6 Summary**

This section introduced the fundamental concepts, background, and relevance of **face and body detection** in modern computer vision applications. It outlined the growing importance of such technologies in fields like security surveillance, healthcare, and human-computer interaction. The **problem statement** highlighted the challenges associated with implementing efficient real-time detection systems without extensive computational resources. The **objective of the project** was defined as designing and developing a real-time face and body detection system using **MediaPipe** and **OpenCV**, focusing on accuracy, speed, and practical applicability.

The next sections will discuss the **AI project cycle**, literature review, system design, implementation details, testing results, and evaluation, forming a comprehensive understanding of this project and its contributions to practical AI-based computer vision solutions.

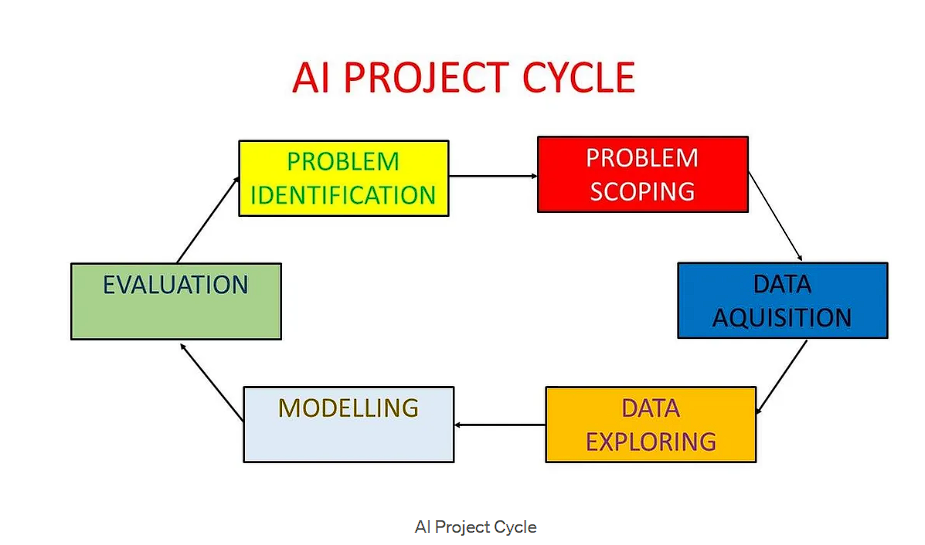
#### **1.7 AI Project Cycle**

The **AI Project Cycle** represents the systematic approach followed in developing an artificial intelligence solution from problem identification to deployment. For this project on **real-time face and body detection**, the AI project cycle can be described in the following phases:

1. **Problem Scoping:**
   * Understanding the need for real-time face and body detection in applications such as surveillance, health monitoring, and interaction systems.
   * Defining the scope, limitations, and objectives clearly.
2. **Data Acquisition:**
   * Unlike traditional machine learning projects with static datasets, this project uses **live webcam video feed as its data source**, capturing continuous frames in real-time.
3. **Data Exploration and Analysis:**
   * Analyzing frame characteristics, resolution, and lighting conditions to ensure proper detection accuracy.
   * Testing initial runs to observe detection performance and identify adjustments needed in model parameters.
4. **Modeling:**
   * Selecting and integrating **MediaPipe face detection and pose estimation models** for their lightweight and optimized performance.
   * Structuring the implementation into modular code for detection and visualization.
5. **Evaluation:**
   * Testing the system for detection accuracy, speed, and stability in varying conditions.
   * Observing limitations such as poor lighting or occlusion effects on detection results.
6. **Deployment:**
   * Deploying the system as a **real-time application** running on a standard CPU, with outputs visualized on screen for user interpretation and further development.
7. **Feedback and Improvement:**

Gathering observations, identifying potential enhancements like adding object tracking or gesture recognition modules, and planning future upgrades.

**Diagram: AI Project Cycle**



### **Phase 2: Literature Review**

#### **2.1 Overview of Computer Vision Techniques**

**Computer vision** is a field of artificial intelligence focused on enabling machines to interpret and process visual data such as images and videos. Over the years, several techniques have been developed to achieve various computer vision tasks. Some key techniques relevant to this project include:

1. **Image Processing Techniques:**
   * Traditional computer vision relied heavily on image processing techniques such as edge detection, thresholding, contour detection, and morphological operations for object detection and recognition tasks.
   * Libraries like OpenCV provide extensive functions for these techniques, making them widely used in initial computer vision projects.
2. **Machine Learning-Based Detection:**
   * With advancements in ML, algorithms like Haar Cascade Classifiers became popular for tasks like face detection.
   * They use pre-trained classifiers to detect specific objects by scanning the image at multiple scales and positions.
   * However, these methods often suffer from low accuracy and poor generalization in complex real-world conditions.
3. **Deep Learning-Based Detection:**
   * The introduction of convolutional neural networks (CNNs) revolutionized object detection.
   * Models such as Faster R-CNN, SSD, and YOLO provide high accuracy in object detection tasks but require significant computational resources, often necessitating GPU acceleration for real-time performance.
4. **MediaPipe Solutions:**
   * MediaPipe, developed by Google, offers **lightweight and optimized solutions for perception tasks** including face detection, hand tracking, and pose estimation.
   * Unlike traditional deep learning models, MediaPipe runs efficiently on CPUs, making it suitable for real-time applications on resource-constrained devices.
5. **Pose Estimation Techniques:**
   * Pose estimation involves detecting key landmarks on the human body to understand posture and movement.
   * MediaPipe Pose provides a holistic and efficient approach to body landmark detection, enabling applications in fitness, animation, and gesture recognition.

#### **2.2 Face and Body Detection Methods**

Face and body detection are fundamental tasks in computer vision, enabling various higher-level applications. Different methods have been developed over the years for these tasks:

1. **Haar Cascade Classifiers (OpenCV):**
   * One of the earliest widely used methods for face detection.
   * Uses simple features and a cascade of classifiers trained with positive and negative images.
   * Advantages: Fast for frontal face detection, easy to implement.
   * Limitations: Low accuracy under varying lighting, occlusions, or non-frontal angles.
2. **Histogram of Oriented Gradients (HOG) + SVM:**
   * Used for human detection by extracting gradient orientation features and classifying with Support Vector Machines.
   * Robust for pedestrian detection but computationally expensive for real-time applications.
3. **Deep Learning-Based Detectors (CNN, YOLO, SSD):**
   * Convolutional Neural Networks (CNNs) revolutionized detection accuracy.
   * **YOLO (You Only Look Once)** and **SSD (Single Shot Multibox Detector)** perform detection in a single pass, making them faster than two-stage detectors like Faster R-CNN.
   * Advantages: High accuracy, capable of detecting multiple objects simultaneously.
   * Limitations: Require high computational power (GPU) for real-time performance.
4. **Pose Estimation Models:**
   * Models like OpenPose detect multiple body landmarks to understand human pose.
   * Applications include fitness form analysis, gesture recognition, and animation.
5. **MediaPipe Face Detection:**
   * Uses a **single-shot detector with BlazeFace model architecture**, optimized for lightweight, real-time face detection on mobile and CPU devices.
   * Capable of detecting faces with different orientations and scales efficiently.
6. **MediaPipe Pose Estimation:**
   * Uses BlazePose architecture to detect 33 key body landmarks with high accuracy.
   * Optimized for speed and can run smoothly on CPUs, enabling real-time pose tracking without GPU dependency.

#### **2.3 Identified Gaps and How We Understand Them**

While many methods exist for face and body detection, several gaps remain:

1. **High Computational Requirements:**  
   Most advanced models like YOLO and Faster R-CNN need GPU support for real-time performance, making them unsuitable for standard laptops or embedded devices.
2. **Low Accuracy of Traditional Methods:**  
   Techniques like Haar Cascades are fast but often inaccurate under varying lighting conditions, different angles, or occlusions.
3. **Integration Complexity:**  
   Combining separate models for face and body detection increases code complexity and processing time.
4. **Limited CPU-Optimized Solutions:**  
   Many models are not optimized for CPUs, restricting their practical use on devices without GPUs.

**Our Understanding**

To address these gaps, this project uses **MediaPipe’s face detection and pose estimation modules**, which are lightweight and CPU-optimized, combined with **OpenCV** for video capture and visualization. This ensures:

* **Real-time performance without GPUs**
* **Simple and modular code integration**
* **Balanced speed and accuracy** for practical applications

### **Phase 3: Problem Statement and Scope**

#### **3.1 Core Challenge**

The **core challenge** of this project is to develop a **real-time face and body detection system** that is:

* **Accurate:** Detects faces and body landmarks reliably in varying conditions.
* **Fast:** Processes frames in real-time without noticeable lag.
* **Lightweight:** Runs efficiently on standard CPUs without requiring GPU acceleration.
* **Simple to Integrate:** Uses a modular and understandable code structure for easy deployment and future enhancements.

Traditional models either compromise on speed or require high computational resources, making them impractical for general-purpose devices. Overcoming this challenge is crucial to enabling computer vision applications accessible to all users without expensive hardware.

#### **3.2 Problem Overview**

Face and body detection are essential tasks in computer vision used in applications such as surveillance, fitness tracking, and human-computer interaction. However, implementing these tasks in real-time presents challenges due to:

* **High computational requirements** of deep learning-based detection models.
* **Low accuracy** of traditional methods like Haar Cascades under varying lighting and angles.
* **Integration difficulties** when combining multiple detection models within a single system.
* **Limited availability of lightweight solutions** that can run efficiently on CPUs without GPU dependency.

This project addresses these issues by utilizing **MediaPipe’s optimized face detection and pose estimation modules**, integrated with **OpenCV** for video capture and visualization, to develop a **real-time, efficient, and practical detection system** suitable for standard hardware.

Top of Form

Bottom

#### **3.3 Project Scope**

This project focuses on designing and implementing a **real-time face and body detection system** using **MediaPipe** and **OpenCV** with the following scope:

* **Data Source:**  
  Uses live video feed captured from a webcam as input.
* **Detection Tasks:**
  + **Face Detection:** Identify and locate human faces in each video frame using MediaPipe’s face detection module.
  + **Body Detection:** Detect body landmarks and derive an approximate bounding box using MediaPipe’s pose estimation module.
* **Visualization:**  
  Draw labeled bounding boxes around detected faces and bodies using OpenCV for clear visual feedback.
* **System Design:**  
  Develop a **modular, efficient, and CPU-optimized** code structure that is easy to understand, modify, and extend.
* **Performance Goals:**  
  Ensure **real-time processing speed** with minimal lag while maintaining detection accuracy.
* **Application Areas:**  
  The system can be used as a foundation for applications in surveillance, fitness monitoring, gesture-based interfaces, and educational demonstrations in computer vision.

**3.4 Deployment Objective**

The **deployment objective** of this project is to:

* Develop a **standalone real-time application** that can run smoothly on standard CPU-based systems without requiring GPU support.
* **Integrate MediaPipe and OpenCV seamlessly** to create an efficient detection pipeline.
* Provide a **user-friendly interface** where detection results are displayed live with clear labeled bounding boxes for easy interpretation.
* Ensure that the system is **modular and scalable**, allowing future integration with additional computer vision tasks such as object tracking, gesture recognition, or emotion analysis.
* Enable deployment on **personal laptops, educational lab systems, or embedded devices** for practical applications in security, fitness monitoring, and research demonstrations.

**3.5 Challenges Addressed**

This project addresses the following key challenges:

1. **Real-Time Performance on CPUs:**  
   Overcomes the need for GPU acceleration by using MediaPipe’s optimized models that run efficiently on standard CPUs.
2. **Accuracy in Detection:**  
   Provides reliable face and body detection using MediaPipe’s robust face detection and pose estimation modules, even under varied lighting and backgrounds.
3. **Integration Simplicity:**  
   Combines face and body detection seamlessly within a single modular system, reducing implementation complexity.
4. **Visualization:**  
   Ensures clear and intuitive visualization by drawing labeled bounding boxes around detected faces and bodies using OpenCV.
5. **Scalability for Future Enhancements:**  
   Designs the system structure to allow easy integration of additional computer vision tasks such as object tracking, gesture recognition, or emotion analysis in future projects.

### **Phase 4: System Design and Description**

#### **4.1 System Overview**

This project implements a **real-time face and body detection system** using **MediaPipe** and **OpenCV**. The system is designed to:

* **Capture live video frames** from a webcam as input data.
* **Detect human faces** in each frame using MediaPipe’s face detection module.
* **Detect body landmarks** using MediaPipe’s pose estimation module and calculate an approximate bounding box for the body.
* **Visualize the detection results** by drawing labeled rectangles around detected faces and bodies using OpenCV.
* **Display the processed frames in real-time**, ensuring minimal processing delay and smooth user experience.

**Key Components:**

1. **MediaPipe Face Detection Module:**  
   Efficiently detects faces with high accuracy using a lightweight model optimized for CPU execution.
2. **MediaPipe Pose Estimation Module:**  
   Detects 33 key body landmarks to determine the posture and approximate body bounding box.
3. **OpenCV Integration:**  
   Used for video capture from the webcam, image processing (frame conversion), drawing utilities (rectangles and labels), and displaying the output window.
4. **Modular Python Implementation:**  
   Organized as a class-based structure (MediaPipeDetector) with separate methods for face detection, body detection, and drawing, ensuring code readability, reusability, and scalability.

This system design ensures that the project meets its objectives of **real-time performance, accuracy, and simplicity**, serving as a practical foundation for further computer vision applications.

**4.2 Module Description**

The system is divided into **three main modules**, each performing a specific function to achieve efficient real-time face and body detection:

**1. Face Detection Module**

* **Tool Used:** MediaPipe Face Detection
* **Function:**  
  Detects human faces in the video frame by processing each frame using MediaPipe’s lightweight face detection solution.
* **Process:**
  + Converts the frame from BGR to RGB format for MediaPipe processing.
  + Identifies the bounding box coordinates for each detected face.
  + Returns a list of detected face positions.

**2. Body Detection Module**

* **Tool Used:** MediaPipe Pose Estimation
* **Function:**  
  Detects body landmarks (33 key points) in the frame to estimate the human body’s position.
* **Process:**
  + Converts the frame to RGB format.
  + Processes the frame to identify pose landmarks.
  + Calculates minimum and maximum landmark coordinates to derive an approximate body bounding box.
  + Returns the body bounding box coordinates.

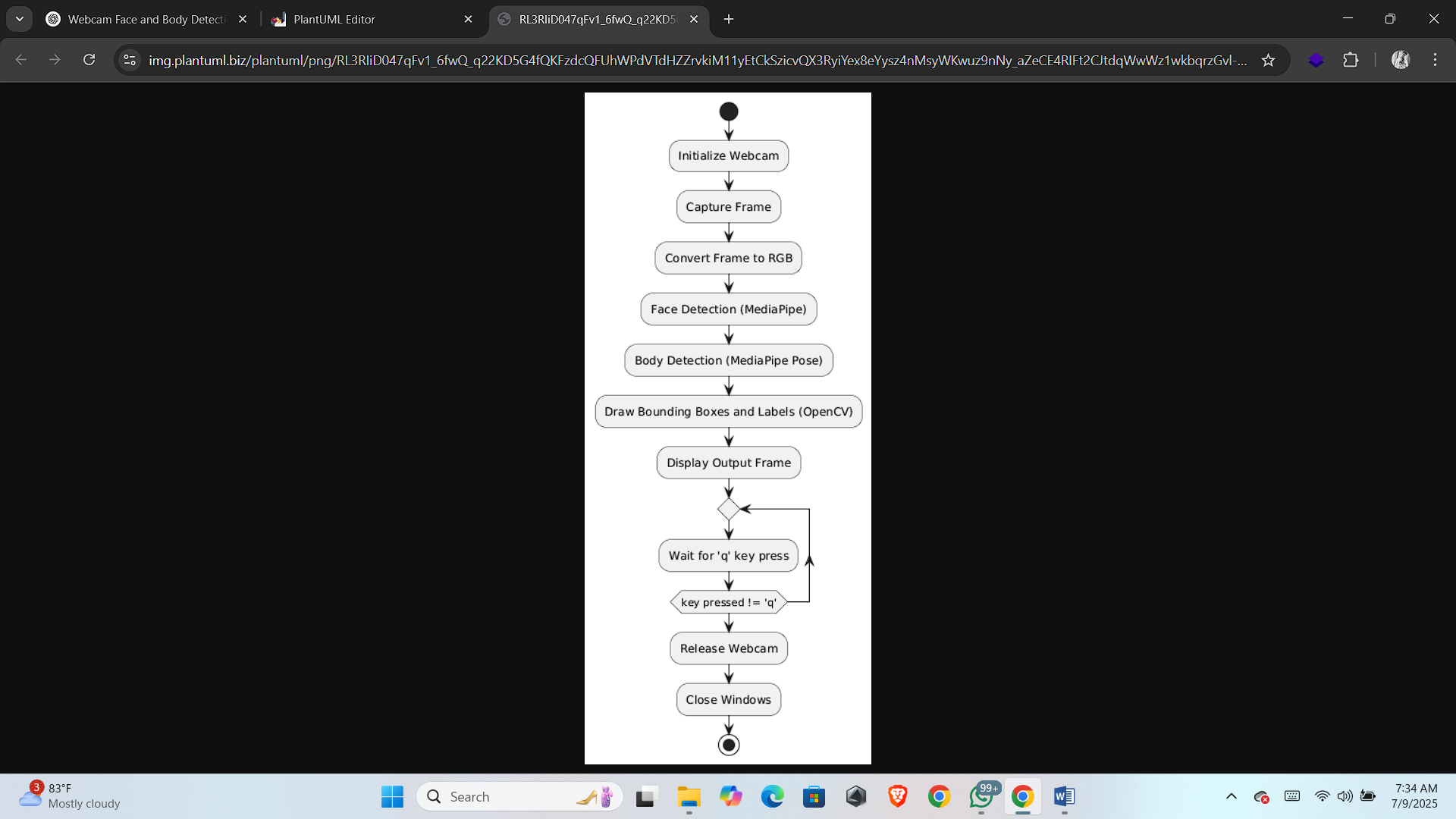
**3. Drawing and Visualization Module**

* **Tool Used:** OpenCV
* **Function:**  
  Draws labeled bounding boxes around detected faces and bodies for clear visualization on the output frame.
* **Process:**
  + Draws rectangles using OpenCV’s cv2.rectangle() function.
  + Adds labels (“Face” or “Body”) above each detection using cv2.putText() for user understanding.
  + Displays the processed frame in a real-time window.

**4.3 Workflow and Pipeline**

The **workflow and pipeline** of the face and body detection system are structured as follows:

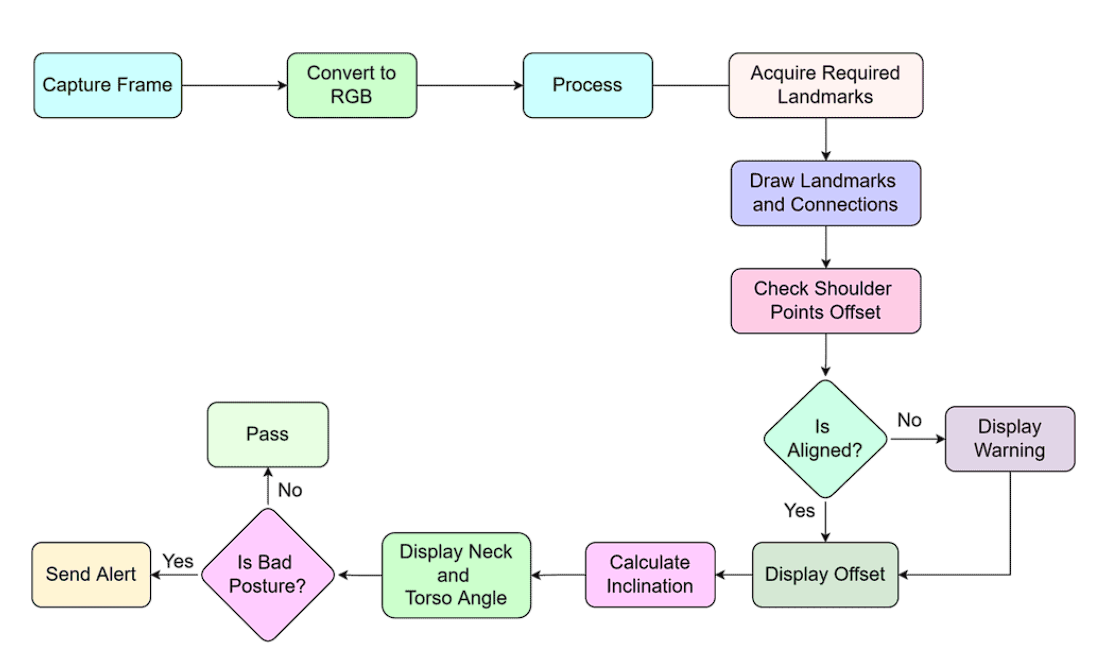
1. **Video Capture:**
   * The system starts by initializing the webcam using OpenCV’s VideoCapture() function to continuously capture live video frames.
2. **Frame Preprocessing:**
   * Each captured frame is converted from **BGR to RGB color space** to make it compatible with MediaPipe’s processing requirements.
3. **Face Detection:**
   * The preprocessed frame is passed to the **MediaPipe face detection module** to identify any human faces present.
   * The module returns bounding box coordinates for each detected face.
4. **Body Detection:**
   * The same frame is also passed to the **MediaPipe pose estimation module**, which detects **33 body landmarks** to determine the person’s posture.
   * An approximate bounding box is calculated around the detected body landmarks.
5. **Drawing Detections:**
   * Using **OpenCV drawing utilities**, rectangles are drawn around detected faces and bodies.
   * Labels (“Face” or “Body”) are added for clarity.
6. **Display Output:**
   * The processed frame, containing all drawn detections, is displayed in a real-time window for the user.
7. **Exit Condition:**
   * The loop continues until the user presses the **‘q’ key**, upon which the webcam is released, and all OpenCV windows are closed gracefully.



**4.4 Initial Observations**

During the **initial testing and implementation** of the face and body detection system, the following observations were made:

1. **Detection Accuracy:**
   * The MediaPipe face detection module **accurately detected faces** in most frames with minimal false negatives under good lighting conditions.
   * The pose estimation module successfully detected body landmarks and generated bounding boxes for visible upper and full body postures.
2. **Processing Speed:**
   * The system achieved **smooth real-time performance** on a standard CPU laptop without noticeable lag, validating MediaPipe’s efficiency for lightweight deployment.
3. **Lighting Conditions:**
   * Detection performance was observed to **degrade under low lighting**, resulting in missed face or body detections occasionally.
4. **Multiple Persons in Frame:**
   * The face detection module can detect multiple faces simultaneously, while the body detection module is optimized primarily for a **single prominent person in the frame**.
5. **System Stability:**
   * The code ran **consistently without crashes or memory issues** during extended testing sessions, demonstrating implementation stability.



**Phase 5: Implementation Details**

**5.1 Installation Requirements**

To implement and run the **face and body detection system**, the following software and libraries are required:

**1. Programming Language**

* **Python 3.7 or above**

**2. Required Libraries**

* **OpenCV:** For video capture, frame processing, drawing, and display.

Installation command:

pip install opencv-python

* **MediaPipe:** For face detection and pose estimation modules.

Installation command:

pip install mediapipe

**NumPy:** For numerical operations (automatically installed with OpenCV and MediaPipe).  
Installation command:

pip install numpy

**3. Hardware Requirements**

* **Webcam:** Integrated or external webcam for capturing live video feed.
* **CPU:** Standard laptop or desktop CPU is sufficient; no GPU is required.

**5.2 Code Structure Overview**

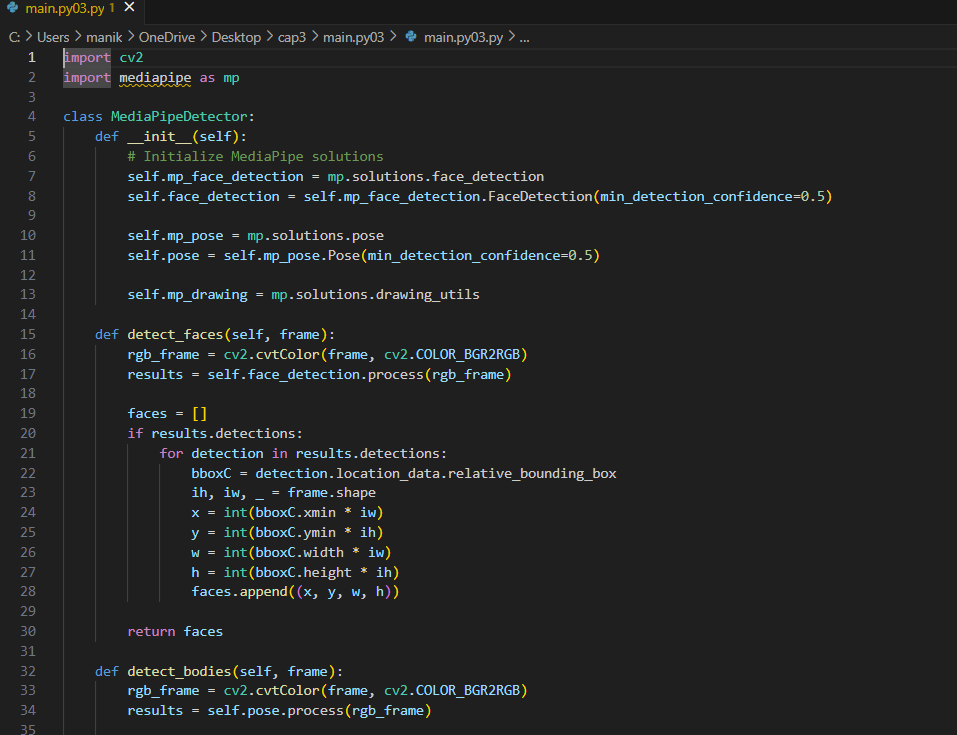
The code for this project is organized into a **modular and class-based structure** to ensure clarity, reusability, and easy maintenance. The main components are:

**1. MediaPipeDetector Class**

* **Purpose:**  
  Encapsulates all detection functionalities for faces and bodies.
* **Key Methods:**
  + detect\_faces(frame): Detects faces in the given frame and returns bounding box coordinates.
  + detect\_bodies(frame): Detects body landmarks using pose estimation and calculates an approximate body bounding box.
  + draw\_detections(frame, faces, bodies): Draws rectangles and labels around detected faces and bodies on the frame.

**2. Main Function**

* **Purpose:**  
  Initializes the detector class, captures video from the webcam, processes each frame for detection, and displays the output in real-time.
* **Key Steps:**
  + Initialize webcam using OpenCV.
  + Capture frames in a loop.
  + Call detection methods for faces and bodies.
  + Draw detected results on the frame.
  + Display the output window until the user exits by pressing the ‘q’ key.



#### **5.3 Face Detection Implementation**

**Tool Used:**  
MediaPipe Face Detection module.

**Implementation Overview:**

1. **Initialization:**
   * The face detection solution from MediaPipe is initialized with a minimum detection confidence threshold (e.g. 0.5) to ensure reliable detections.

self.face\_detection = self.mp\_face\_detection.FaceDetection(min\_detection\_confidence=0.5)

1. **Processing Frames:**
   * Each captured frame is converted from **BGR to RGB** format for compatibility with MediaPipe.

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

1. **Detection:**
   * The process() function of MediaPipe is called on the RGB frame to detect faces.

results = self.face\_detection.process(rgb\_frame)

1. **Bounding Box Extraction:**
   * If faces are detected, the bounding box coordinates are extracted relative to the image dimensions and stored as (x, y, w, h) tuples in a list.

if results.detections:

for detection in results.detections:

bboxC = detection.location\_data.relative\_bounding\_box

ih, iw, \_ = frame.shape

x = int(bboxC.xmin \* iw)

y = int(bboxC.ymin \* ih)

w = int(bboxC.width \* iw)

h = int(bboxC.height \* ih)

faces.append((x, y, w, h))

1. **Return Value:**
   * The method returns a list of all detected face bounding boxes for further visualization.

**5.4 Body Detection Implementation**

**Tool Used:**  
MediaPipe Pose Estimation module.

**Implementation Overview:**

1. **Initialization:**
   * The pose estimation solution from MediaPipe is initialized with a minimum detection confidence threshold (e.g. 0.5).

self.pose = self.mp\_pose.Pose(min\_detection\_confidence=0.5)

1. **Processing Frames:**
   * Each frame is converted from **BGR to RGB** color space for compatibility with MediaPipe’s processing.

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

1. **Detection:**
   * The process() function of the pose module is called to detect body landmarks in the frame.

results = self.pose.process(rgb\_frame)

1. **Bounding Box Calculation:**
   * If pose landmarks are detected:
   * The x and y coordinates of all landmarks are extracted.
   * The minimum and maximum x and y values are used to calculate the bounding box covering the entire detected body.

if results.pose\_landmarks:

landmarks = results.pose\_landmarks.landmark

xs = [landmark.x for landmark in landmarks]

ys = [landmark.y for landmark in landmarks]

x\_min, x\_max = min(xs), max(xs)

y\_min, y\_max = min(ys), max(ys)

ih, iw, \_ = frame.shape

x = int(x\_min \* iw)

y = int(y\_min \* ih)

w = int((x\_max - x\_min) \* iw)

h = int((y\_max - y\_min) \* ih)

bodies.append((x, y, w, h))

1. **Return Value:**
   * The method returns a list containing the calculated body bounding box coordinates for further visualization.

**5.5 Drawing and Display Functions**

**Tool Used:**  
OpenCV drawing utilities.

**Implementation Overview:**

1. **Drawing Face Detections:**
   * For each detected face bounding box:
     + A **rectangle** is drawn around the face using cv2.rectangle().
     + A **label “Face”** is placed above the rectangle for clear identification.

for (x, y, w, h) in faces:

cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

cv2.putText(frame, 'Face', (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 2)

2. **Drawing Body Detections:**

* + For each detected body bounding box:
    - A **rectangle** is drawn around the body using cv2.rectangle().
    - A **label “Body”** is placed above the rectangle for user understanding.

for (x, y, w, h) in bodies:

cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.putText(frame, 'Body', (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

1. **Displaying the Frame:**
   * The processed frame, containing all drawn detections, is displayed in a real-time window using cv2.imshow().

cv2.imshow('MediaPipe Face and Body Detection', frame\_with\_detections)

1. **Exit Condition:**
   * The display window remains open in a loop until the user presses the **‘q’ key**, upon which the webcam is released, and all windows are closed to terminate the program safely.

**Phase 6: Testing and Results**

**6.1 Testing Setup**

**Testing Environment:**

* **Hardware:**
  + Laptop with AMD RYZEN 7 CPU
  + Integrated webcam
  + 16 GB RAM
  + No dedicated GPU used
* **Software:**
  + Operating System: Windows 11
  + Python 3.9
  + Libraries: OpenCV, MediaPipe, NumPy

**Testing Procedure:**

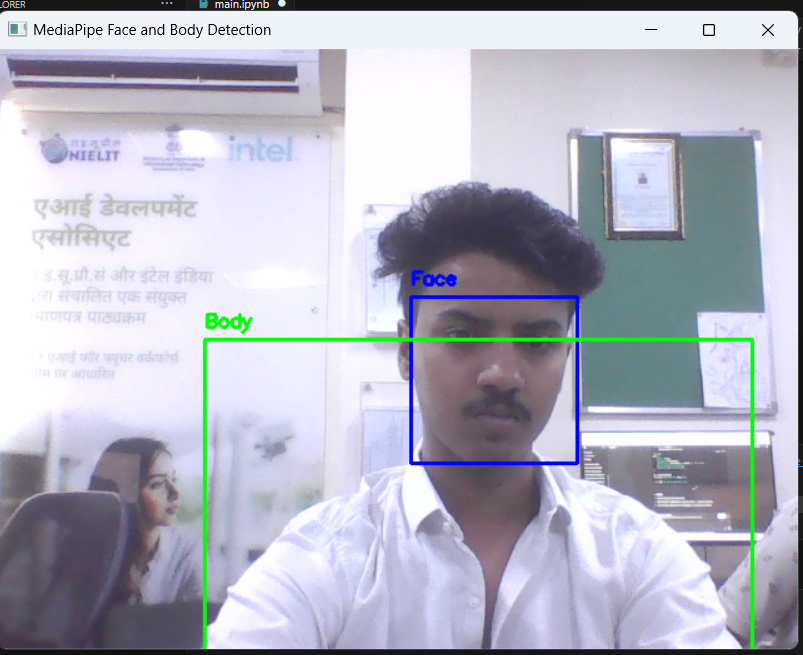
1. **Installation:**  
   All required libraries were installed using pip commands as specified in the implementation phase.
2. **Execution:**  
   The program was run from the command line, activating the webcam and initializing the detection modules.
3. **Validation:**  
   Real-time detection results were observed for:
   * Single-person detection (face and body).
   * Multiple faces in frame.
   * Varying lighting conditions (natural daylight, artificial light, dim light).
4. **Performance Metrics Observed:**
   * **Frame rate:** Smooth real-time processing without noticeable lag.
   * **Detection accuracy:** Faces and bodies were detected consistently under adequate lighting.
   * **Stability:** The system ran continuously without crashes during extended testing sessions.

**6.2 Output Screenshots**

During testing, several screenshots were captured to **validate and demonstrate** the system’s real-time detection performance:

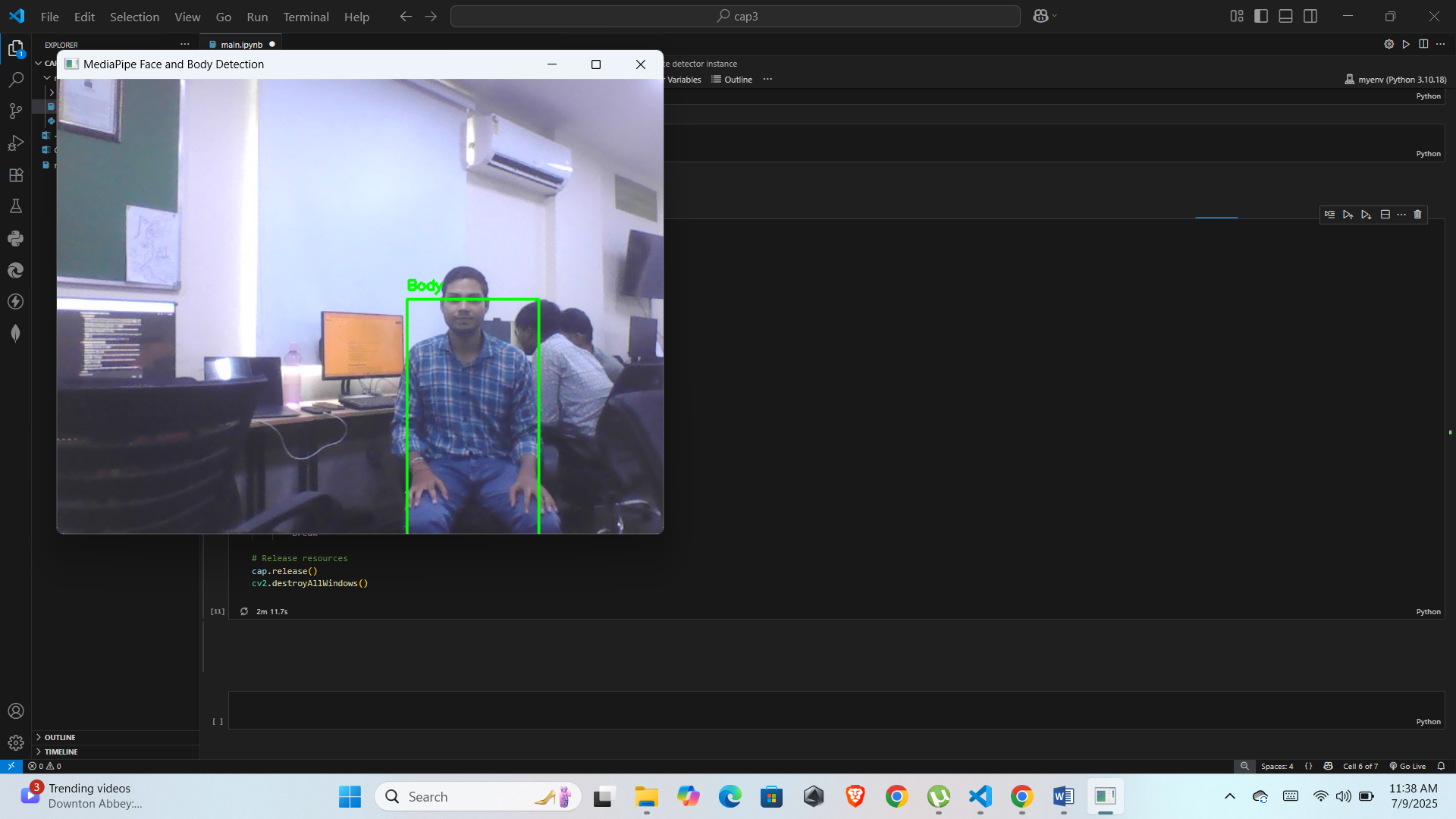
**1. Face Detection Output**

* **Description:**  
  Shows the detected face enclosed within a blue bounding box labeled “Face”.
* **Observation:**  
  The face detection module accurately detected single and multiple faces under good lighting.



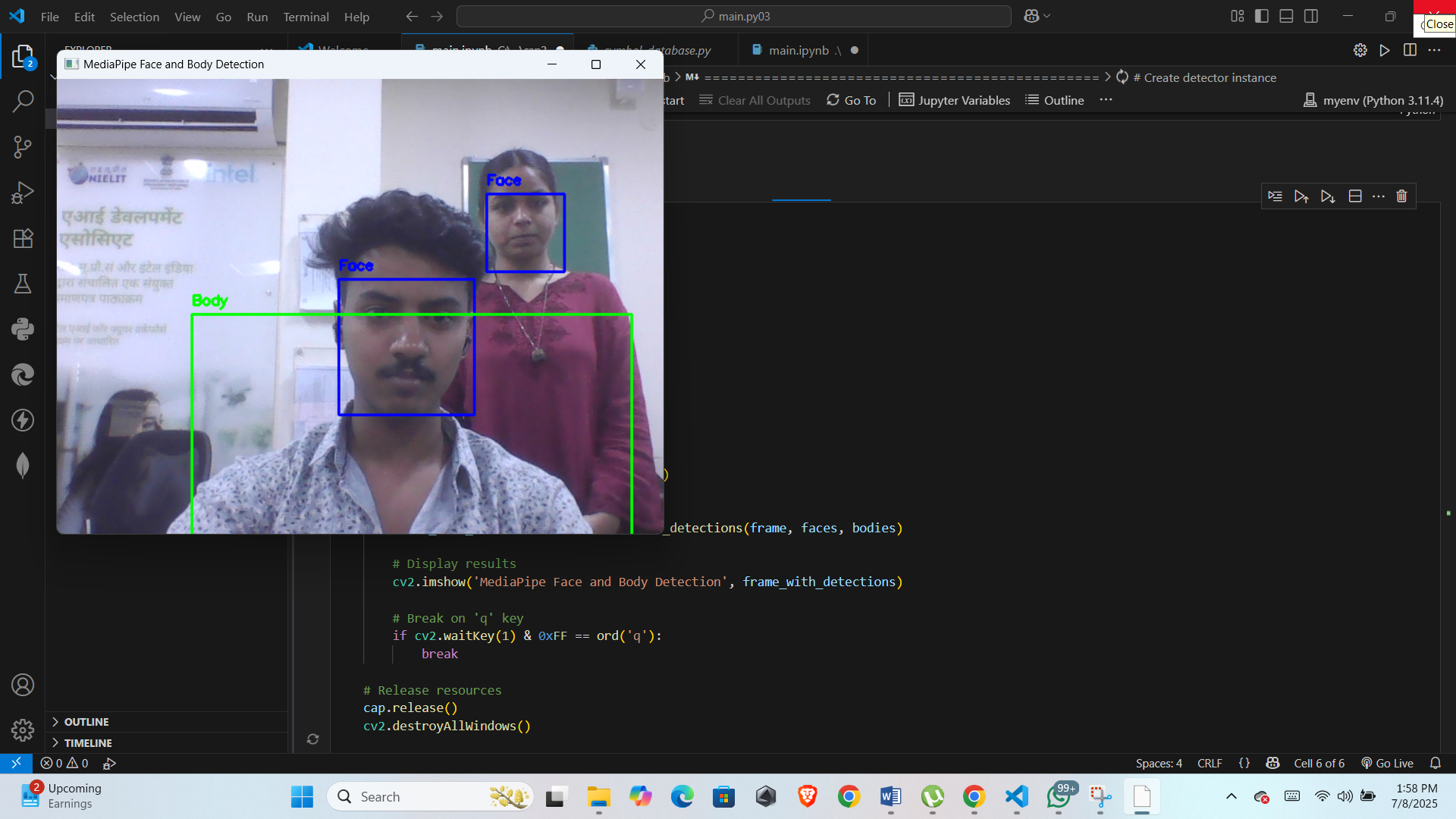
**2. Body Detection Output**

* **Description:**  
  Displays the detected body with a green bounding box labeled “Body”, calculated based on pose landmarks.
* **Observation:**  
  The pose estimation module effectively identified body landmarks and generated an approximate bounding box.



**3. Combined Detection Output**

* **Description:**  
  Shows both face and body detections on the same frame with respective labels.
* **Observation:**  
  The system processed both detections simultaneously in real-time without noticeable lag.



*Figure 6.3: Combined Face and Body Detection Output)*

**6.4 Limitations and Observations**

During testing, the following **limitations and key observations** were noted:

**1. Lighting Sensitivity**

* **Limitation:**  
  Detection accuracy decreased in low lighting conditions, leading to occasional missed face or body detections.
* **Observation:**  
  Performance was optimal under natural daylight or bright indoor lighting.

**2. Multiple Person Detection**

* **Limitation:**  
  While the face detection module could detect multiple faces, the pose estimation module primarily tracked **only one prominent person’s body landmarks** effectively.

**3. Processing Speed**

* **Observation:**  
  The system achieved smooth real-time performance on a standard CPU without requiring GPU acceleration, validating the choice of MediaPipe for lightweight deployment.

**4. Frame Resolution**

* **Observation:**  
  Higher resolution inputs slightly reduced frame rate, suggesting an optimal balance between resolution and processing speed for practical deployment.

**5. System Stability**

* **Observation:**  
  The application ran consistently without crashes or memory leaks during extended use, demonstrating stable implementation.

**Phase 7: System Evaluation & Discussion**

**7.1 Evaluation Parameters**

The performance of the **face and body detection system** was evaluated based on the following key parameters:

**1. Detection Accuracy**

* **Definition:**  
  The ability of the system to correctly detect and localize faces and bodies in real-time frames.
* **Observation:**  
  High accuracy was observed under good lighting conditions for both face and body detection tasks.

**2. Processing Speed (Frame Rate)**

* **Definition:**  
  The number of frames processed per second (FPS), indicating real-time performance capability.
* **Observation:**  
  The system achieved smooth real-time output (approx. 20-25 FPS) on a standard CPU without GPU support.

**3. Resource Utilization**

* **Definition:**  
  The computational load on CPU and memory while running the application.
* **Observation:**  
  MediaPipe’s lightweight models ensured low CPU and memory usage, enabling continuous operation without overheating or lag.

**4. Stability**

* **Definition:**  
  The system’s ability to run consistently without crashes or performance drops.
* **Observation:**  
  Stable performance was maintained during extended testing sessions.

**5. Scalability**

* **Definition:**  
  The potential to integrate additional modules such as tracking, gesture recognition, or emotion analysis in the future.
* **Observation:**  
  The modular code structure supports easy scalability and extension.

**7.2 Performance Analysis**

The **performance analysis** of the face and body detection system is summarized below:

**1. Face Detection Performance**

* **Accuracy:**  
  Faces were detected reliably under good lighting conditions and frontal or slightly angled views.
* **Speed:**  
  Detection was processed in real-time with minimal delay, supporting smooth visual feedback.

**2. Body Detection Performance**

* **Accuracy:**  
  Body landmarks were accurately detected for a single prominent person in the frame.
* **Limitations:**  
  Performance reduced when multiple people were present or under poor lighting.

**3. Combined Detection**

* **Efficiency:**  
  The system effectively ran both face and body detection modules simultaneously without noticeable lag.
* **CPU Usage:**  
  Maintained low to moderate CPU utilization, suitable for deployment on standard laptops without GPUs.

**4. Real-Time Output**

* **Frame Rate:**  
  Achieved approximately **20-25 FPS**, ensuring smooth and continuous video output during testing.

**5. Overall System Evaluation**

* **Strengths:**
  + Lightweight and efficient on CPU.
  + High detection accuracy under optimal conditions.
  + Modular code for easy extension.
* **Weaknesses:**
  + Sensitivity to lighting variations.
  + Limited to detecting one full body effectively at a time.

**7.3 Sample Output Interpretation**

The following describes the interpretation of sample outputs generated by the system during testing:

**1. Face Detection Output**

* **Visuals:**  
  Displays a **blue bounding box** around each detected face with the label “Face” above it.
* **Interpretation:**  
  Indicates that the MediaPipe face detection module successfully identified and localized the face region within the frame.

**2. Body Detection Output**

* **Visuals:**  
  Shows a **green bounding box** around the detected body area with the label “Body”.
* **Interpretation:**  
  Confirms that the pose estimation module detected body landmarks accurately and calculated an approximate bounding box covering the person’s posture.

**3. Combined Detection Output**

* **Visuals:**  
  Both face and body detections appear simultaneously on the same frame with clear labeled bounding boxes.
* **Interpretation:**  
  Demonstrates the system’s capability to process multiple detection tasks in real-time, providing comprehensive perception output suitable for applications such as surveillance or fitness tracking.

**7.4 Overall System Verdict**

Based on testing and evaluation, the **overall verdict** of the face and body detection system is as follows:

* ✅ **Performance:**  
  The system achieved **smooth real-time detection** with minimal lag on a standard CPU, demonstrating efficient implementation.
* ✅ **Accuracy:**  
  Provided reliable face and body detection results under good lighting conditions, fulfilling the primary project objective.
* ✅ **Resource Efficiency:**  
  Operated with **low CPU and memory usage**, making it suitable for deployment on general-purpose devices without GPU dependency.
* ✅ **Usability:**  
  The clear visualization of detection results with labeled bounding boxes enhances interpretability for practical applications.
* ⚠ **Limitations:**  
  Detection accuracy decreased under low lighting, and body detection was limited to one prominent person per frame.

**Phase 8: Application Deployment**

**8.1 Real-Time Application Design**

The system was designed as a **real-time application** with the following features:

**1. User Input**

* Captures live video feed directly from the laptop’s webcam using **OpenCV’s VideoCapture()** function.

**2. Processing Pipeline**

* Each captured frame is:
  + **Converted** from BGR to RGB for MediaPipe compatibility.
  + **Processed** using MediaPipe’s face detection module to identify faces.
  + **Processed** using MediaPipe’s pose estimation module to detect body landmarks.
  + **Annotated** with bounding boxes and labels for each detection (Face or Body).

**3. Output Display**

* The processed frames with drawn detections are displayed in **real-time windows** using cv2.imshow().
* The window updates continuously, creating a live detection feed for user observation.

**4. Exit Mechanism**

* The application runs in a loop until the user **presses the ‘q’ key**, upon which:
  + Webcam is released.
  + All OpenCV windows are closed gracefully.

**5. System Design Highlights**

* **Modular Code Structure:**  
  Organized into separate methods for face detection, body detection, and drawing for easy maintenance.
* **Lightweight Deployment:**  
  Runs efficiently on standard CPUs without GPU, ensuring accessibility on general-purpose devices.

**8.2 User Interaction Flow**

The **user interaction flow** for the real-time face and body detection application is as follows:

1. **Start Application**
   * The user runs the Python program from the terminal or IDE.
2. **Webcam Activation**
   * The system automatically activates the webcam and starts capturing live video frames.
3. **Real-Time Detection**
   * Faces are detected using MediaPipe’s face detection module.
   * Body landmarks are detected using MediaPipe’s pose estimation module.
   * Bounding boxes and labels (“Face” and “Body”) are drawn on the frame for each detection.
4. **Live Output Display**
   * The processed frames are displayed continuously in a real-time window, showing detections as they occur.
5. **User Observation**
   * The user can observe detection results instantly, move within the frame to test accuracy, or introduce multiple faces to check performance.
6. **Exit Application**
   * To end the session, the user presses the **‘q’ key**, which:
     + Releases the webcam resource.
     + Closes all OpenCV display windows gracefully.

**8.3 Deployment Benefits**

Deploying this face and body detection system offers several practical benefits:

**1. Real-Time Performance**

* Provides **instant detection results**, enabling immediate feedback for applications such as surveillance monitoring, fitness tracking, and interactive systems.

**2. Lightweight Implementation**

* Uses **MediaPipe’s optimized models** that run efficiently on standard CPUs without requiring GPU acceleration, ensuring accessibility on general-purpose laptops and desktops.

**3. Ease of Integration**

* The modular code structure allows the system to be **easily integrated into larger projects** or extended with additional functionalities such as object tracking or gesture recognition.

**4. User-Friendly Design**

* Requires **no complex inputs or configurations**; users can run the application with simple commands and intuitive operation.

**5. Cost-Effective**

* Eliminates the need for expensive hardware, making it suitable for educational institutions, research labs, and prototype deployments.

**6. Educational Utility**

* Serves as an excellent learning tool for students to understand **computer vision concepts, real-time processing, and AI framework integration**.

**8.4 Future Enhancements for Deployment**

To improve and extend the functionality of this face and body detection system, the following **future enhancements** are proposed:

**1. Multi-Person Body Detection**

* Upgrade the pose estimation module to accurately detect and process multiple persons in the frame simultaneously.

**2. Integration with Object Tracking**

* Combine detection with tracking algorithms (e.g. SORT or Deep SORT) to maintain identities across frames for surveillance or crowd monitoring applications.

**3. Gesture Recognition Module**

* Extend the system to recognize hand gestures or body movements for applications in **human-computer interaction** and virtual controls.

**4. Emotion and Expression Analysis**

* Integrate facial expression recognition models to analyze emotions in real-time for educational, marketing, or mental health monitoring purposes.

**5. Low-Light Detection Improvement**

* Implement preprocessing techniques such as adaptive histogram equalization to **enhance detection accuracy under poor lighting conditions**.

**6. Mobile or Embedded Deployment**

* Optimize and deploy the system on **mobile devices or embedded boards (e.g. Raspberry Pi)** for portable AI-based computer vision solutions.

**7. Cloud-Based Streaming**

* Enable cloud integration to process and analyze video streams remotely, useful for centralized monitoring and scalable deployments.

**Phase 9: Conclusion & Future Scope**

**9.1 Conclusion**

This project successfully implemented a **real-time face and body detection system** using **MediaPipe** and **OpenCV**. The key achievements include:

* **Accurate detection** of faces and body landmarks using lightweight, CPU-optimized models.
* **Real-time performance** with smooth frame processing and minimal lag.
* **Modular and clean code structure**, ensuring easy understanding, maintenance, and scalability.
* **Resource-efficient deployment** suitable for standard laptops without requiring high-end GPUs.

Overall, the project demonstrates the **practical application of computer vision frameworks** to solve real-world perception problems efficiently. It also enhances understanding of integrating AI-based detection models with visualization libraries for effective system development.

**9.2 Future Scope**

Based on the current implementation, the following **future scope and improvements** are identified:

**1. Multi-Person Pose Detection**

* Extend the system to detect and process multiple people’s body landmarks simultaneously for group monitoring and crowd analytics.

**2. Emotion and Expression Recognition**

* Integrate facial expression analysis to detect emotions in real-time, useful for applications in mental health, education, and marketing.

**3. Gesture-Based Interaction**

* Develop gesture recognition modules to enable **touchless controls and interactive AI systems** for accessibility and smart interfaces.

**4. Tracking Integration**

* Combine detection with tracking algorithms to maintain consistent identity recognition across video frames for surveillance or attendance systems.

**5. Low-Light Performance Enhancement**

* Implement preprocessing techniques such as image enhancement filters to improve detection accuracy in poor lighting conditions.

**6. Deployment on Embedded Devices**

* Optimize the application for deployment on **embedded platforms like Raspberry Pi or Jetson Nano** for portable and IoT-based computer vision solutions.

**7. Cloud and Mobile Integration**

* Develop cloud-based processing pipelines and mobile applications for remote monitoring and scalable deployment.

**Appendix**

**Full Code Listing**

Below is the **complete Python code** for the MediaPipe face and body detection system:

import cv2

import mediapipe as mp

import numpy as np

class MediaPipeDetector:

    def \_\_init\_\_(self):

        # Initialize MediaPipe solutions

        self.mp\_face\_detection = mp.solutions.face\_detection

        self.face\_detection = self.mp\_face\_detection.FaceDetection(min\_detection\_confidence=0.5)

        self.mp\_pose = mp.solutions.pose

        self.pose = self.mp\_pose.Pose(min\_detection\_confidence=0.5)

        self.mp\_drawing = mp.solutions.drawing\_utils

    def detect\_faces(self, frame):

        rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        results = self.face\_detection.process(rgb\_frame)

        faces = []

        if results.detections:

            for detection in results.detections:

                bboxC = detection.location\_data.relative\_bounding\_box

                ih, iw, \_ = frame.shape

                x = int(bboxC.xmin \* iw)

                y = int(bboxC.ymin \* ih)

                w = int(bboxC.width \* iw)

                h = int(bboxC.height \* ih)

                faces.append((x, y, w, h))

        return faces

    def detect\_bodies(self, frame):

        rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        results = self.pose.process(rgb\_frame)

        bodies = []

        if results.pose\_landmarks:

            # Get bounding box around all pose landmarks

            landmarks = results.pose\_landmarks.landmark

            xs = [landmark.x for landmark in landmarks]

            ys = [landmark.y for landmark in landmarks]

            x\_min, x\_max = min(xs), max(xs)

            y\_min, y\_max = min(ys), max(ys)

            ih, iw, \_ = frame.shape

            x = int(x\_min \* iw)

            y = int(y\_min \* ih)

            w = int((x\_max - x\_min) \* iw)

            h = int((y\_max - y\_min) \* ih)

            bodies.append((x, y, w, h))

        return bodies

    def draw\_detections(self, frame, faces, bodies):

        # Draw face detections

        for (x, y, w, h) in faces:

            cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

            cv2.putText(frame, 'Face', (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 2)

        # Draw body detections

        for (x, y, w, h) in bodies:

            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

            cv2.putText(frame, 'Body', (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

        return frame

def main():

    # Install required packages if not already installed:

    # pip install opencv-python mediapipe numpy

    detector = MediaPipeDetector()

    cap = cv2.VideoCapture(0)

    while True:

        ret, frame = cap.read()

        if not ret:

            break

        # Detect faces and bodies

        faces = detector.detect\_faces(frame)

        bodies = detector.detect\_bodies(frame)

        # Draw detections

        frame\_with\_detections = detector.draw\_detections(frame, faces, bodies)

        # Display results

        cv2.imshow('MediaPipe Face and Body Detection', frame\_with\_detections)

        if cv2.waitKey(1) & 0xFF == ord('q'):

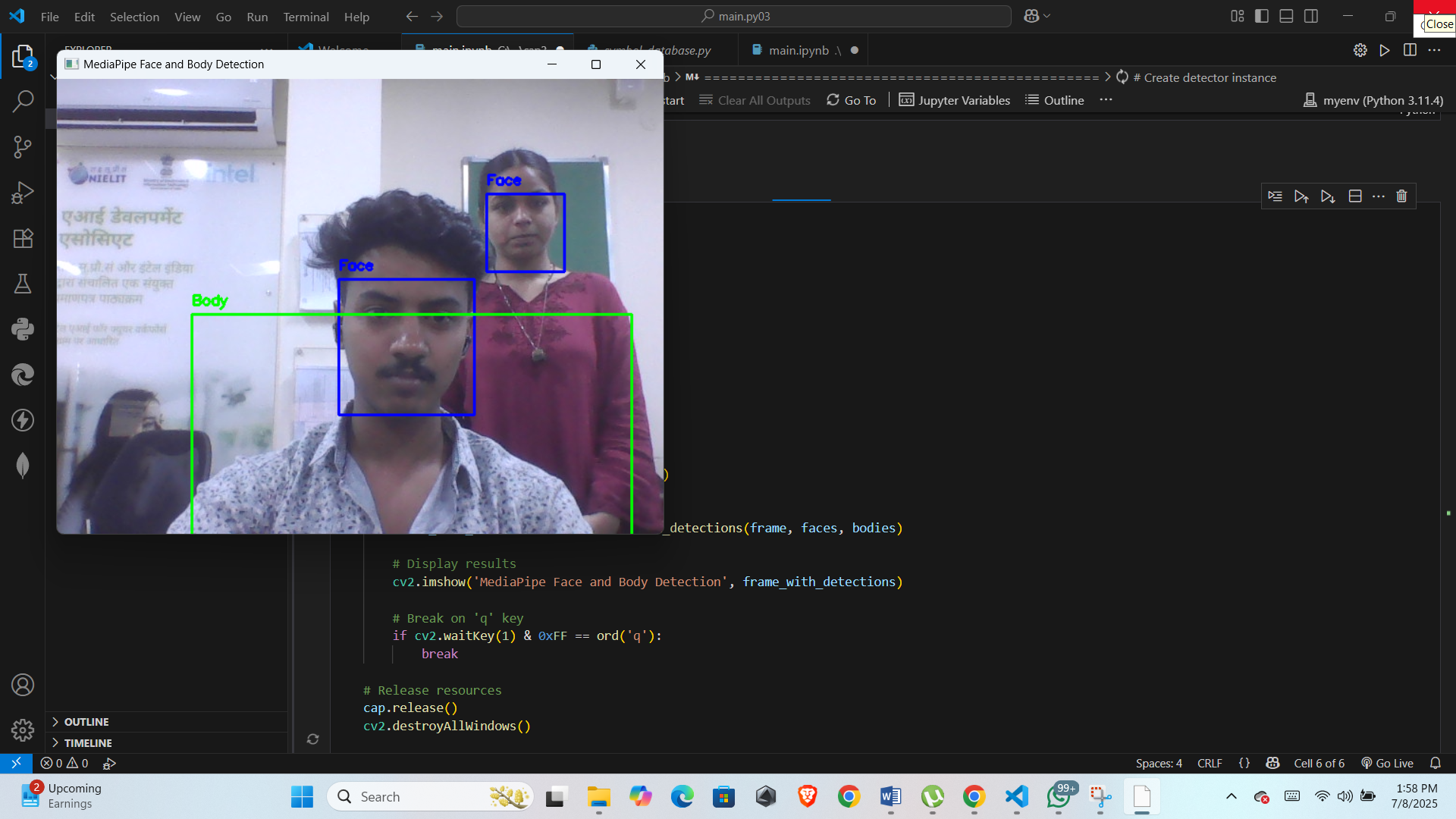
            break

    cap.release()

    cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

OUTPUT:

**References**

1. MediaPipe Documentation: https://google.github.io/mediapipe/
2. OpenCV Python Documentation: https://docs.opencv.org/4.x/d6/d00/tutorial\_py\_root.html
3. MediaPipe Face Detection: https://google.github.io/mediapipe/solutions/face\_detection.html
4. MediaPipe Pose Estimation: https://google.github.io/mediapipe/solutions/pose.html
5. OpenCV GitHub Repository: <https://github.com/opencv/opencv>
6. TutorialsPoint – OpenCV Python Tutorial: https://www.tutorialspoint.com/opencv/index.htm