A PROJECT REPORT ON

Real-Time Human Pose Estimation: A MediaPipe and Python Approach for 3D Detection and Classification

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING (Bachelor's of Technology)
Submitted by:

Hitesh Maindola	1918025
Manan Garg	1918041
Shashank Negi	1918058

Under the Guidance of
Ms. Vipanshi Kansal
Assistant Professor

Project Team ID: MP22CSEH19.

Department of Computer Science and Engineering
Graphic Era (Deemed to be University)
Dehradun, Uttarakhand
MAY-2023

CANDIDATE’S DECLARATION

We hereby certify that the work which is being presented in the Project Report entitled **“Real-Time Human Pose Estimation: A Mediapipe and Python Approach for 3D Detection and Classification”** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering and submitted in the Department of Computer Science and Engineering of the Graphic Era (Deemed to be University), Dehradun is an authentic record of my own work carried out during a period from **August-2022 to May-2023** under the supervision of **Ms. Vipanshi Kansal**, **Assistant Professor**, Department of Computer Science and Engineering, Graphic Era (Deemed to be University).

The matter presented in this dissertation has not been submitted by me/us for the award of any other degree of this or any other Institute/University.

Hitesh Maindola	1918025	hiteshmaindola
Manan Garg	1918041	manangarg
Shashank Negi	1918058	shashanknegi

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Signature	Signature
Supervisor	**Head of the Department**

External Viva

Name of the Examiners:	**Signature with Date**
1.	
2.	

Abstract

Abstract of the project work : This project uses Mediapipe and Python to create a real-time 3D pose detection and pose categorization system. In several applications, including motion analysis, virtual reality, and human-computer interface, human posture estimation is essential. The suggested system makes use of the flexibility and ease-of-use of Python programming, as well as Mediapipe, a powerful framework for creating multimodal perceptual pipelines.

The system uses OpenCV to manage the incoming video stream, enabling in-the-moment frame capture and analysis. The Mediapipe framework, which incorporates cutting-edge computer vision and machine learning algorithms, is then used to process the collected frames. This makes it possible to extract pose information from the video frames, making it easier to recognise 3D human poses.

The system uses calculating angle heuristics to improve posture analysis. The system learns more about the arrangement and movements of the human body by calculating the angles between important body joints. These angle heuristics enable a more complex study of human poses and offer useful information for pose classification.

The suggested system's accuracy in recognising and classifying human poses in real-time scenarios is shown by experimental findings. The system offers a useful tool by combining the strengths of Python, Mediapipe, OpenCV, and calculating angle heuristics the system offers a useful tool to researchers and developers who are interested in .

Keywords: Pose Classification, Mediapipe, Real-time, 3D pose detection, Angle Heuristics, Python, OpenCV, Machine Learning

Acknowledgement

Any achievement, be in scholastic or otherwise does not depend solely on the individual effort but on the guidance, encouragement and co-operation of intellectuals, elders and friends. A number of personalities in their own capacity have helped me in carrying out this project work.

Our sincere thanks to project guide **Ms.Vipashi Kansal, Assistant Professor,** Department of Computer Science and Engineering, Graphic Era (Deemed to be University), for his valuable guidance and support throughout the course of project work and for being a constant source of inspiration.

We extend our thanks to **Prof. (Dr.) Guru Prasad M.S.**, Project coordinator, Department of Computer Science and Engineering, Graphic Era (Deemed to be University), for his valuable suggestions throughout all the phases of the Project Work.

We are extremely grateful to **Prof. (Dr.) D. P. Singh**, HOD of the Computer Science and Engineering Department, Graphic Era (Deemed to be University), for his moral support and encouragement.

We thank the **management of Graphic Era (Deemed to be University)** for the support throughout the course of our Bachelor's Degree and for all the facilities they have provided.

Last, but certainly not least we thank all teaching and non-teaching staff of Graphic Era (Deemed to be University) for guiding us in the right path. Most importantly we wish to thank our parents for their support and encouragement.

Hitesh Maindola	1918025
Manan Garg	1918041
Shashank Negi	1918058

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

In the following sections, a brief introduction and the problem statement for the work has been included.

1.1 Project Introduction

A "Human Pose estimation" is a Computer Vision task to visualize a person's orientation. This method is frequently used to anticipate the position of a person's body components or joints. Given the wide range of uses for such a system, it is one of the most intriguing fields of computer vision research and has had significant growth.

Due to their numerous applications in areas including augmented reality, motion capture, human-computer interaction, and sports analytics, real-time 3D pose recognition and pose classification have attracted a lot of attention in recent years. It is possible to create a variety of interactive and immersive experiences and gain important insights into human movement patterns thanks to the accurate real-time detection and classification of human positions.

This Report represents the in-depth view of real-time 3D pose detection by using the Mediapipe framework and Python programming language. An extensive selection of pre-built tools and models are provided by the flexible cross-platform Mediapipe framework, which may be used to create multimodal perceptual AI applications. Python offers a flexible and user-friendly environment because it is a well-known and widely used programming language.

In several applications, including motion analysis, virtual reality, and human-computer interface, human posture estimation is essential.

1.2 Problem Statement

The problem statement for the present work can be stated as follows: In order to successfully analyse human movements utilising Python, Mediapipe, OpenCV, and calculating angle heuristics, a real-time 3D pose detection and pose classification system is required. In computer vision, estimating human stance is a difficult issue, and present systems either lack real-time functionality or fall short of offering accurate and thorough pose analysis.

1. Real-Time Performance: Creating a system that can process video frames in real-time would ensure accurate pose detection and classification even in dynamic situations.
2. Accuracy and Robustness: Achieving high accuracy and resilience in pose identification and classification while taking into account variances in human body poses, occlusions, and noisy video inputs.
3. Mediapipe and OpenCV integration: Using the Mediapipe framework and OpenCV library's combined capabilities to manage video input streams, decode pose data, and enable real-time analysis.
4. Angle Heuristics for Pose Analysis: By include the calculation of angle heuristics, extra information about the arrangement and motion of the human body is provided, improving the outcomes of pose identification and classification.
5. Ease of Use and Flexibility: Create a flexible system that is simple to use and can be seamlessly integrated with current tools using Python.

…..

1.3 Objectives

The objective of this project is to develop a human activity and pose recognition system using computer vision techniques. The system should be able to accurately analyze video or image data and identify human activities and poses in real-time.

1. Proper posture: Our project will provide the details for the proper posture during exercise and any workout which will be reducing the chances for the injuries in a very efficient manner.
2. Cost effective: Our project will be helping to train people in the proper posture in a cost-effective manner as it will be replacing personal trainers in gym which costs really high.
3. Flexibility and convenience: Using our project any individual will be able to exercise in a proper manner at anywhere and at any place without going to any gym.
4. Our project will be a handy tool like live trainer for proper yoga postures and poses.

Chapter 2

Literature Survey/ Background

1	2021.	Mukhiddin Toshpulat ov Wookey L eeSuan Le e2 · Arousha H aghighian Roudsari	A Review on Human Pose Estimation	This paper extensively summarizes the current deep learning-based 2D and 3D human pose, hand and mesh estimation methods with a single or multi- person, single or double-stage methodology-based taxonomy.	The PoseFix model refnes the input 2D coordinates of all the persons' human body key points in an input image. It is built based on the top-down pipeline, which processes a cropped human image's tuple and a given pose estimation result of that person.
2	1999	Human Motion Analysis: A Review	J.K.AggarwalQ.Cai	This paper gives an overview of the various tasks involved in motion analysis of the human body.	Tracking human motion from a single view or multiple perspectives focuses on higher-level

					processing, in which moving humans are tracked without identifying their body parts.
3	2007	Vision-based hand pose estimation: A review	AliErola,George Bebisa,Mircea Nicoles	This paper presents a literature review on the latter research direction, which is a very challenging problem in the context of HCI.	CV-based pose estimation, Single frame pose estimation, Partial hand pose estimation
4	2015	A Review on Hand Gesture Recognition System	Jayesh S.Sonkusare Ravindra Sar	This paper reviews the comparative study of various hand gesture recognition techniques which are presented up-till now.	Many Researchers are still developing robust and efficient new hand gesture recognition techniques. (without any peripheral devices)

Table 2.1 Literature Survey

2.1 Introduction to Mediapipe

Google's Mediapipe is a potent open-source platform that makes it easier to create multimodal perceptual AI applications. It offers a number of ready-made, adaptable building components, or "pipelines," for a variety of activities, including audio and video processing, hand tracking, face detection, and position estimation. It is simpler to develop and deploy AI-powered applications on a variety of devices because to these pipelines' seamless platform compatibility.

Developers may employ advanced machine learning and computer vision algorithms without having to design everything from scratch thanks to Mediapipe's broad selection of pre-trained models and user-friendly APIs. Developers can customise and mix various components thanks to the platform's modular architecture to meet their unique needs. The framework also allows real-time processing, allowing for the quick construction of applications that can process data as it comes in.

2.2 Introduction to OpenCV

OpenCV: commonly known as Open Source Computer Vision, a library well-known for open-source library for computer vision and machine learning. It offers a broad range of tools and techniques that let programmers carry out a variety of computer vision, image processing, and machine learning-related activities.

Python, C++, Java, and more are just a few of the many programming languages that OpenCV supports, making it adaptable and useful for many platforms and applications. It offers a selection of models that have already been trained to do tasks like object identification, face recognition, position estimation, and others.

Some key features and functionalities of OpenCV include:

1. Image and Video Processing
2. Feature Detection and Extraction
3. Object Detection and Recognition
4. Machine Learning Integration
5. Real-time Processing and Computer Vision Applications

Chapter 3

Setting up the Environment

Setting up the environment is the crucial part. You will be led through each step required to ensure a successful project execution in this part.

1. **Installing Python:** Start by downloading Python from the official Python website (https://www.python.org), preferably the most recent version. Implementation guidelines unique to your operating system should be followed.

2. **Managing Python Dependencies:** To install and manage third-party libraries, Python offers package management tools like pip or conda. Install the project's necessary dependencies, such as Mediapipe, NumPy, OpenCV, and scikit-learn, using these tools.

3. **Setting Up a Virtual Environment**: To separate the project's dependencies from the overall Python installation, it is advised to set up a virtual environment. This promotes consistency and cleanliness in the surroundings. Virtual environments can be created and managed using programmes like virtualenv or conda.

4. **Installing Mediapipe**: Installing Mediapipe using pip or conda after the virtual environment has been set up. Access to the required elements for 3D pose detection and posture classification will be made possible by this.

5. **Configuring the Development Environment**: The development environment should be configured. To write and run Python code, set up your preferred integrated development environment (IDE) or text editor, such as Visual Studio Code, PyCharm, or Jupyter Notebook. Set up the environment to use the previously built virtual environment.

6. **Importing Necessary Libraries:** To assure access to their capabilities, import Mediapipe, NumPy, OpenCV, and scikit-learn at the beginning of your Python scripts or notebooks.

These instructions will enable you to create a development environment that is correctly setup and has all the dependencies installed. This will give MediaPipe and Python developers a strong foundation for enabling real-time 3D pose detection and pose categorization.

Chapter 4

Requirements and Methodology

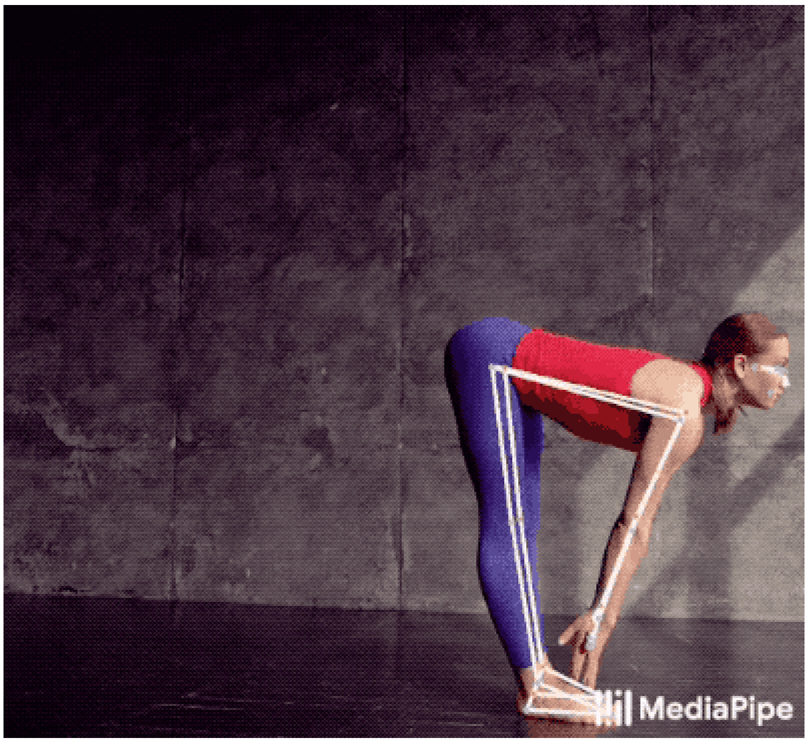


Figure 4.1 Exercise Pose Detection

4.1 Requirements

4.1.1 Hardware Requirements:

1. Processor: For effective calculation and real-time performance, a strong processor is essential. To meet the computational needs of posture estimation methods, a multi-core CPU like the Intel Core i7 or AMD Ryzen 7 is advised.

2. Graphics Processing Unit (GPU): For pose estimation and activity detection GPU can handle data much more quickly when using a dedicated GPU. It is advised to use a high-performance GPU with CUDA or OpenCL capability, such as the NVIDIA GeForce RTX series or AMD Radeon RX series.

3. Memory (RAM): Running sophisticated algorithms and managing massive datasets require a sufficient amount of memory. It is advised to have at least 8 GB of RAM, but higher capacities, such as 16 GB or more, might enhance performance, particularly when working with bigger models or several simultaneous processes.

4. Storage: The dataset, pre-trained models, and intermediate outputs must be kept in sufficient storage. For quicker data access and processing, solid-state drives (SSDs) are preferable.

5. Camera: A high-resolution camera that can record crystal-clear, in-depth photographs or videos is necessary. It will serve as a live input device for our project, feeding data into our programm so that the appropriate results can be produced.

6. Operating System: Based on the particular requirements of the posture estimation and activity recognition software libraries or frameworks being used, confirm compatibility with the selected operating system, such as Windows, macOS, or Linux.

7 Power Supply: To sustain the hardware components and guarantee stable and uninterrupted functioning, a sufficient power supply is required.

4.1.2 Software Requirements:

The following software prerequisites need to be taken into account for human posture estimation and activity recognition to be implemented successfully:

1. Operating System: Based on your experience and compatibility with the necessary software libraries and frameworks, select an appropriate operating system, such as Windows, macOS, or Linux.

2. Use well-known Integrated Development Environments (IDEs), such PyCharm, Visual Studio Code, or Jupyter Notebook, to build up a productive development environment. These environments offer a very user-friendly interface for creating, troubleshooting, and running code for many uses.

3. Select a programming language that provides strong support for computer vision tasks. Python is a popular option because of its rich libraries, which include OpenCV and TensorFlow; for performance-critical components, choose C++.

4. Computer Vision Libraries: Install and make use of thorough libraries for computer vision, such as OpenCV. Pose estimation and simple activity detection are ideal uses of OpenCV's vast variety of tools and methods for image and video processing.

4.2 Methodology

Methodology for Human Pose Detection:

4.2.1 Data Gathering and Processing

1. Data gathering: Compile a variety of photographs or videos that show people in different positions. To ensure resilience, use a variety of backgrounds, lighting settings, and vantage positions.

2. Joint Localization: Use a human joint localization algorithm to find the shoulders, elbows, wrists, hips, knees, and ankles—among other important joints in the human body. Convolutional neural networks (CNN) and pose estimation models like OpenPose are typical methodologies.

4.2.2 Model Building

3. Pose Estimation: After the joints have been localised, calculate the body's pose using pose estimation algorithms. This entails estimating the position and angles of body parts as well as the relationships between joints. The use of deep learning-based techniques like convolutional pose machines or hourglass networks as well as graphical models like pictorial structures are common techniques.

4. Evaluation and Optimization: Using relevant evaluation measures, such as joint correctness, pose similarity, or mean average precision, evaluate and optimise the pose detection system's performance. Improve accuracy and robustness of the process by iterative adjustments to parameters and algorithms.

4.2.3 Real-time Implementation

5. Real-time Implementation: Boost the pose detection algorithm's efficiency for in-the-moment use. To accomplish low latency processing, this may involve model compression, parallelization, or hardware acceleration approaches.

6. Testing and Validation: To validate the posture identification system's generalizability, test it on a different validation dataset. Examine its robustness to changes in lighting, occlusions, and various body kinds, and validate its performance across various scenarios.

7. Integration and Deployment: Integrate and deploy the pose detection system into the intended framework or application. Make sure the target platform is compatible, then optimise for effectiveness and resource use.

8. Performance Monitoring and Maintenance: Continue to keep an eye on the deployed system's performance and deal with any problems or defects that crop up. Maintain the system's compatibility with new pose detection methods, and as necessary, retrain or fine-tune models using fresh data.

Table 4.1 Documenting the Algorithm used in the Project

```
begin
    1. Importing the necessary Libraries
    2. Initialize Pose detection Model
    3. Load and preprocess input image.
    4. Perform pose detection on each image.
    5. – For each detected pose:
        - Calculate heuristics angles.
        - Classify the pose based on the calculated angles.
        - Mark the classified pose on the image.
    6. Display the resulting images with detected and classified poses.
end
```

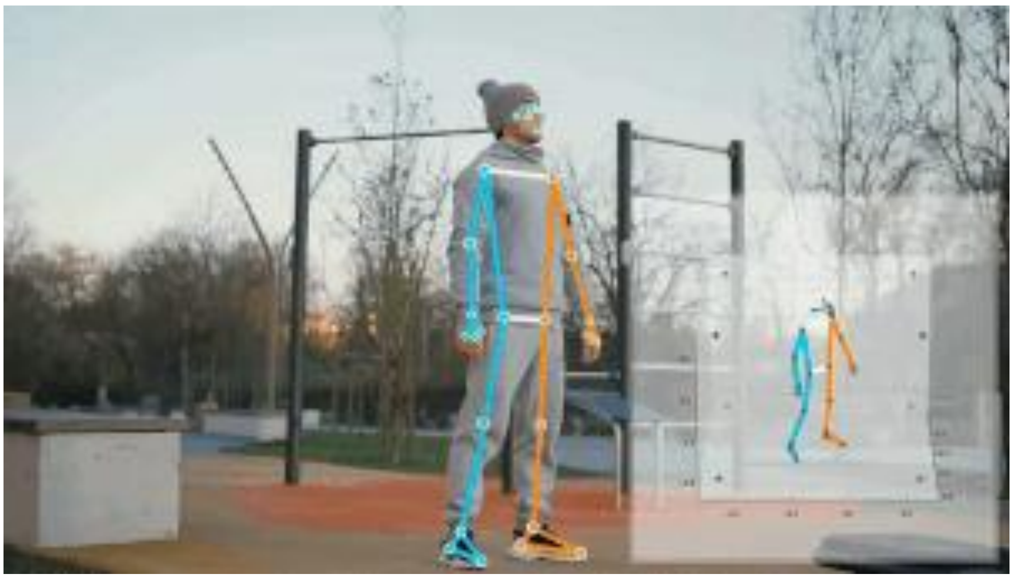
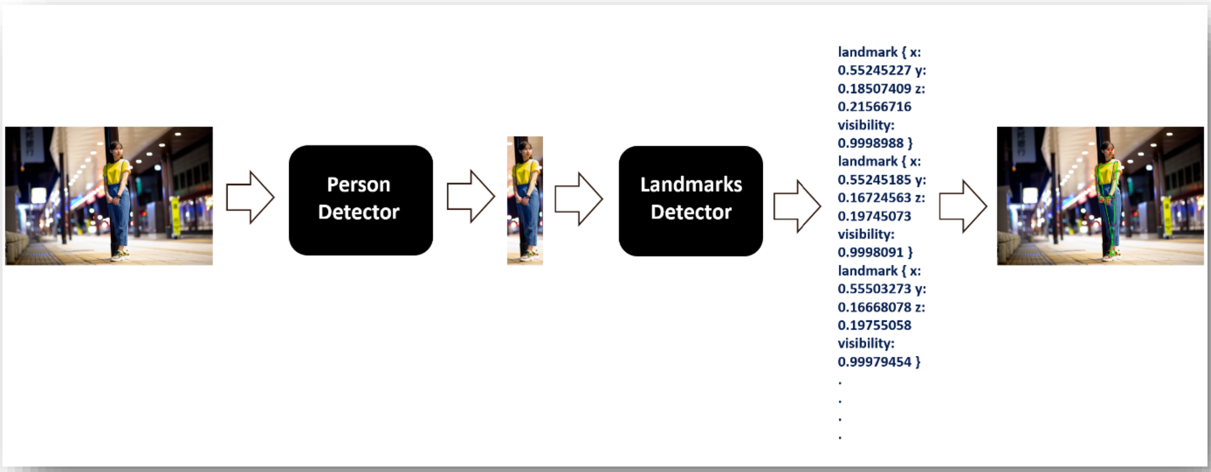


Figure 4.2 Landmark Detection of 2D image and plotting it in 3D Space

Chapter 5

Coding/Code Templates

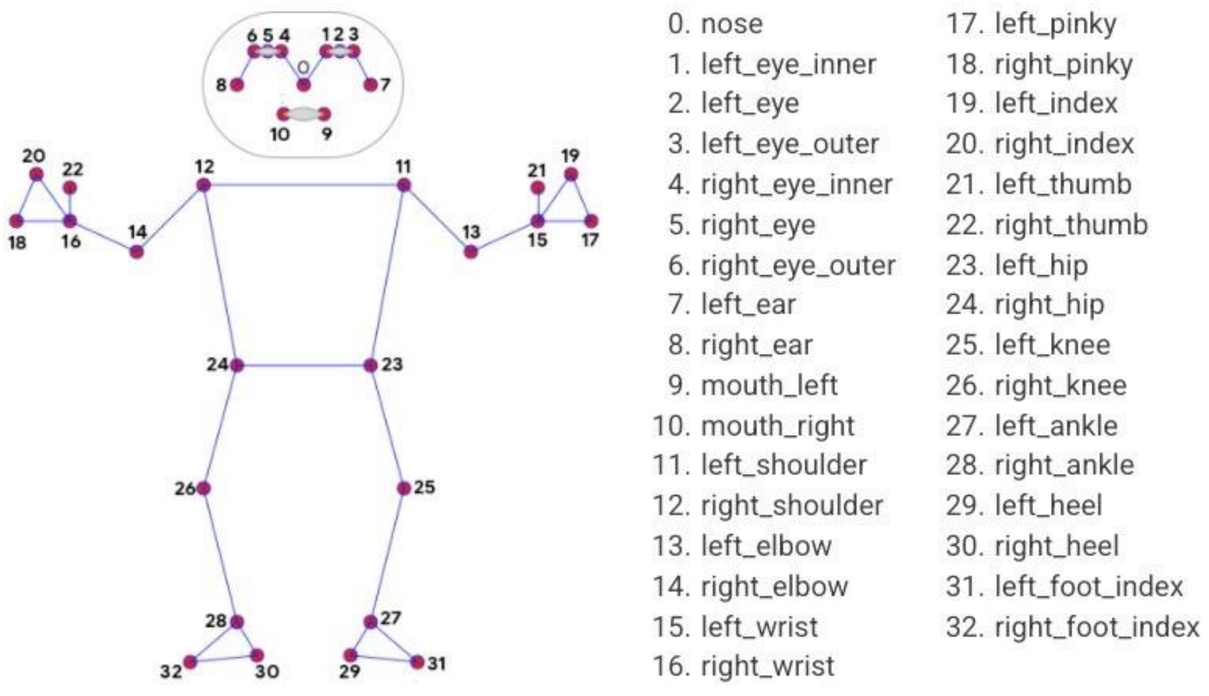
Pose Detection (also known as Pose Estimation) is a widely used computer vision task that enables you to predict humans poses in images or videos by localizing the key body joints (also reffered as landmarks), these are elbows, shoulders, and knees, etc.



```
landmark { x:
0.55245227 y:
0.18507409 z:
0.21566716
visibility:
0.9998988 }
landmark { x:
0.55245185 y:
0.16724563 z:
0.19745073
visibility:
0.9998091 }
landmark { x:
0.55503273 y:
0.16668078 z:
0.19755058
visibility:
0.99979454 }
.
.
.
.
```

Figure 5.1 Landmark Detection Process

There are 33 landmarks which are used to detect the key joints in an human body. For which the detector is used for the very first frame and then key joints are derived from the previous frame's pose by using a tracking method.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Figure 5.2 33 Body Landmarks

Importing the necessary libraries:

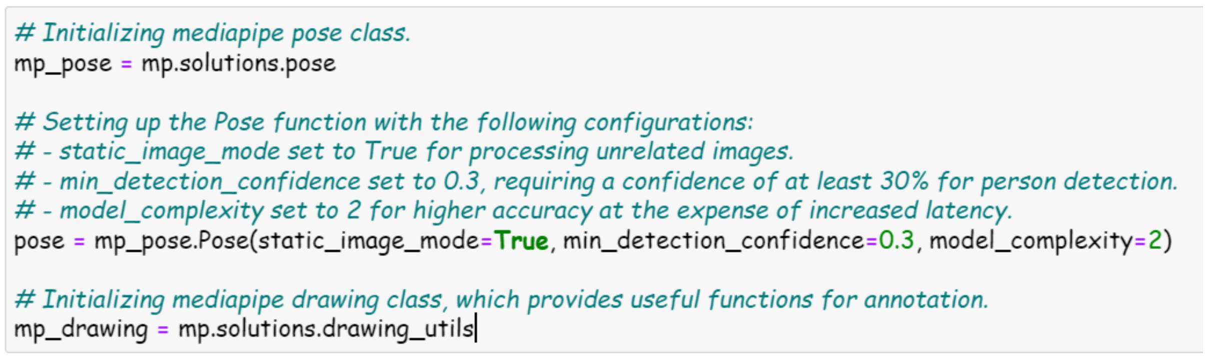
Library	**Description**	**Key Functions**	**Working in the Project**	**Alternative(s)**	**Advantages over Alternatives**
Math	Provides mathematical functions and operations	sin(), cos(), radians(), degrees(), etc.	Used for mathematical calculations	-	-
CV2	OpenCV library for computer vision tasks	imread(), imshow(), resize(), findContours(), etc.	Used for handling video input and image processing	PIL (Python Imaging Library)	High-performance, vast functionality

Library	Description	Key Functions	Working in the Project	Alternative(s)	Advantages over Alternatives
NumPy	Numerical computing library for arrays and math	array(), reshape(), dot(), mean(), etc.	Used for efficient array operations and computations	-	Fast array operations, extensive mathematical functions
Time	Provides time-related functions and operations	sleep(), time(), perf_counter(), etc.	Used for timing and performance measurements	-	Accurate timing, performance measurement
Matplotlib	Data visualization library	plot(), scatter(), imshow(), savefig(), etc.	Used for visualizing pose detection results	Seaborn, Plotly	Extensive plotting options, integration with Python ecosystem

Table 5.1: Details of the Libraries Used

Initializing the Pose detection model:

Utilising the **mp.solutions.position** syntax and call the setup method mp.solutions.pose to initialise the posture detection class.Pose(). The **static_image_mode**, **min_detection_confidence**, **min_tracking_confidence**, **model_complexity**, and **smooth_landmarks** inputs are all passed to the setup function. For landmark visualisation, we also initialise the mp.solutions.drawing_utils class or use OpenCV.

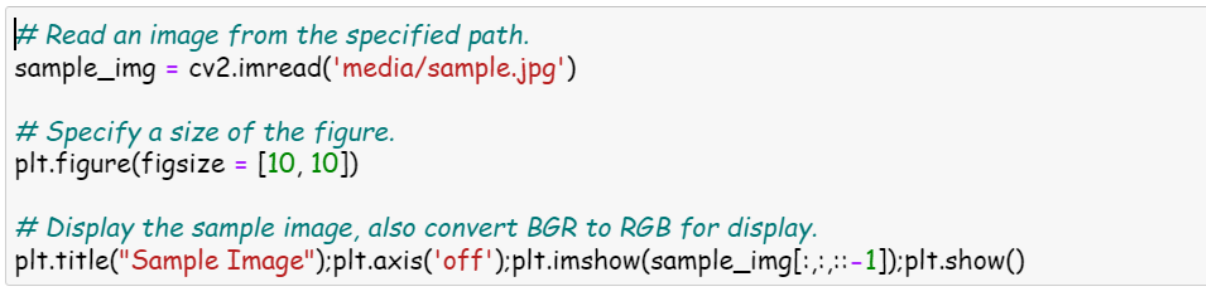


```python
# Initializing mediapipe pose class.
mp_pose = mp.solutions.pose

# Setting up the Pose function with the following configurations:
# - static_image_mode set to True for processing unrelated images.
# - min_detection_confidence set to 0.3, requiring a confidence of at least 30% for person detection.
# - model_complexity set to 2 for higher accuracy at the expense of increased latency.
pose = mp_pose.Pose(static_image_mode=True, min_detection_confidence=0.3, model_complexity=2)

# Initializing mediapipe drawing class, which provides useful functions for annotation.
mp_drawing = mp.solutions.drawing_utils
```

Working on Image Data:

OpenCV: OpenCV is an open-source computer vision and machine learning library which is an acronym for Open Source Computer Vision Library. It offers a broad range of tools and techniques that let programmers carry out a variety of computer vision, image processing, and machine learning-related activities.It provides comprehensive computer vision functionalities.

For the image processing we are using OpenCV library.



```python
# Read an image from the specified path.
sample_img = cv2.imread('media/sample.jpg')

# Specify a size of the figure.
plt.figure(figsize = [10, 10])

# Display the sample image, also convert BGR to RGB for display.
plt.title("Sample Image");plt.axis('off');plt.imshow(sample_img[:,:,::-1]);plt.show()
```

Perform Pose Detection:

To perform pose detection, we use the mp.solutions.pose.Pose().process() function. Before passing the image to the pipeline, we convert it from BGR to RGB using cv2.cvtColor(). The output consists of 33 landmarks with normalized x, y, and z coordinates, along with visibility values. We display the first two landmarks to provide an overview of the model's output.

```python
def detectPose(image, pose, display=True):

    # Create a copy of the input image.
    output_image = image.copy()

    # Convert the image from BGR into RGB format.
    imageRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Perform the Pose Detection.
    results = pose.process(imageRGB)

    # Retrieve the height and width of the input image.
    height, width, _ = image.shape

    # Initialize a list to store the detected landmarks.
    landmarks = []

    # Check if any landmarks are detected.
    if results.pose_landmarks:

        # Draw Pose landmarks on the output image.
        mp_drawing.draw_landmarks(image=output_image, landmark_list=results.pose_landmarks,
                                  connections=mp_pose.POSE_CONNECTIONS)

        # Iterate over the detected landmarks.
        for landmark in results.pose_landmarks.landmark:

            # Append the landmark into the list.
            landmarks.append((int(landmark.x * width), int(landmark.y * height),
                                  (landmark.z * width)))
    # Check if the original input image and the resultant image are specified to be displayed.
    if display:

        # Display the original input image and the resultant image.
        plt.figure(figsize=[22,22])
        plt.subplot(121);plt.imshow(image[:,:,::-1]);plt.title("Original Image");plt.axis('off');
        plt.subplot(122);plt.imshow(output_image[:,:,::-1]);plt.title("Output Image");plt.axis('off');

        # Also Plot the Pose landmarks in 3D.
        mp_drawing.plot_landmarks(results.pose_world_landmarks, mp_pose.POSE_CONNECTIONS)

    # Otherwise
    else:

        # Return the output image and the found landmarks.
        return output_image, landmarks
```

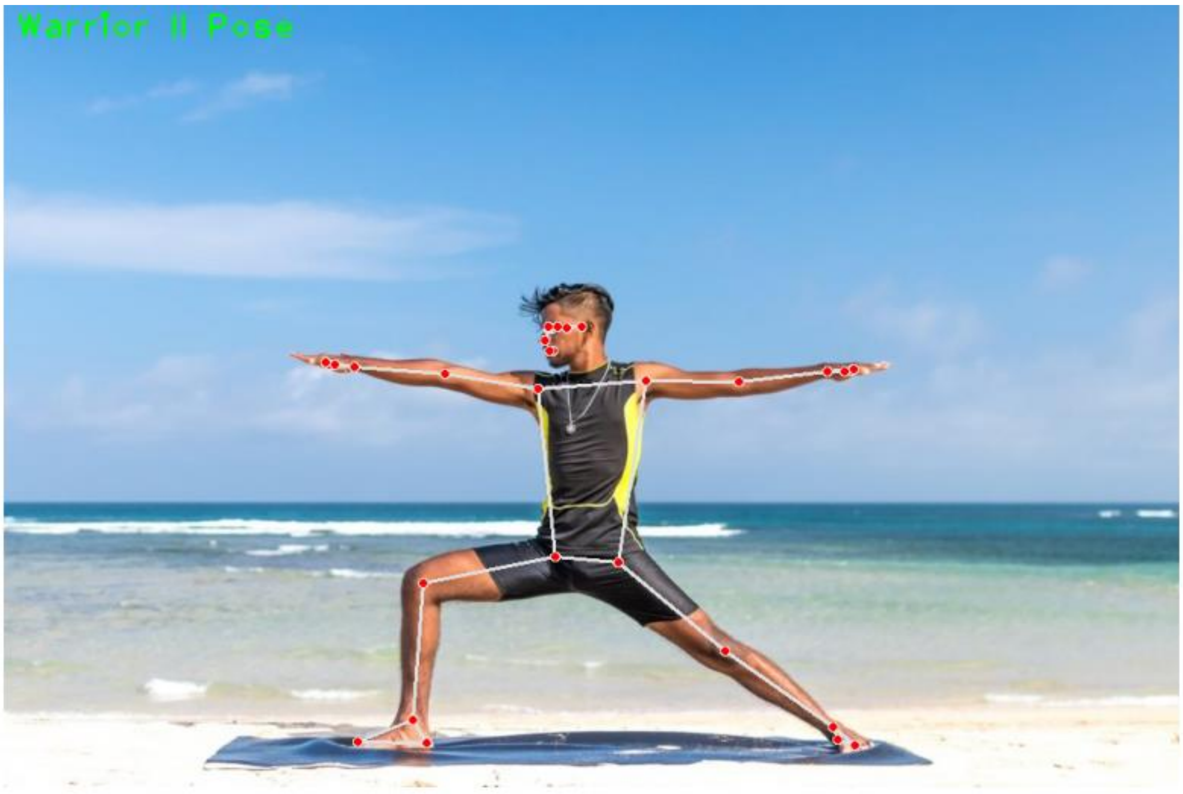


Figure 5.3 Key points detection

Pose classification by calculating the heuristics angles between the landmarks:

Creating a function that will be capable of calculating angles between three landmarks. The angle between landmarks? Do not get confused, as this is the same as calculating the angle between two lines.

The first point (landmark) is considered as the starting point of the first line, the second point (landmark) is considered as the ending point of the first line and the starting point of the second line as well, and the third point (landmark) is considered as the ending point of the second line.

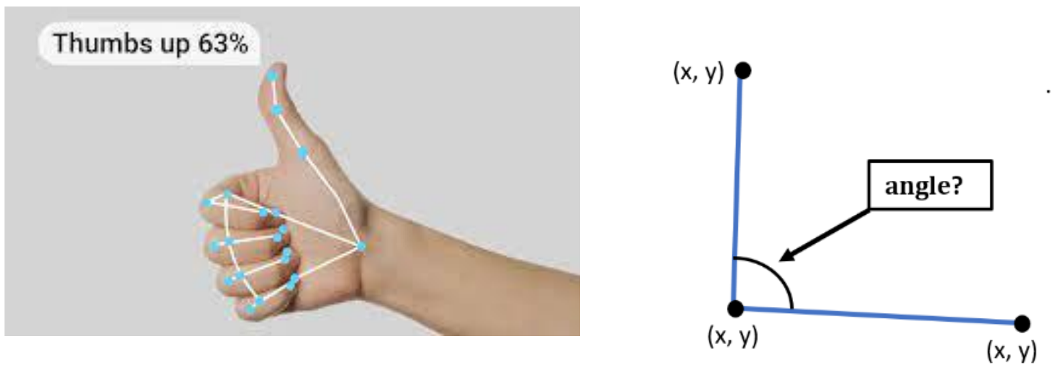
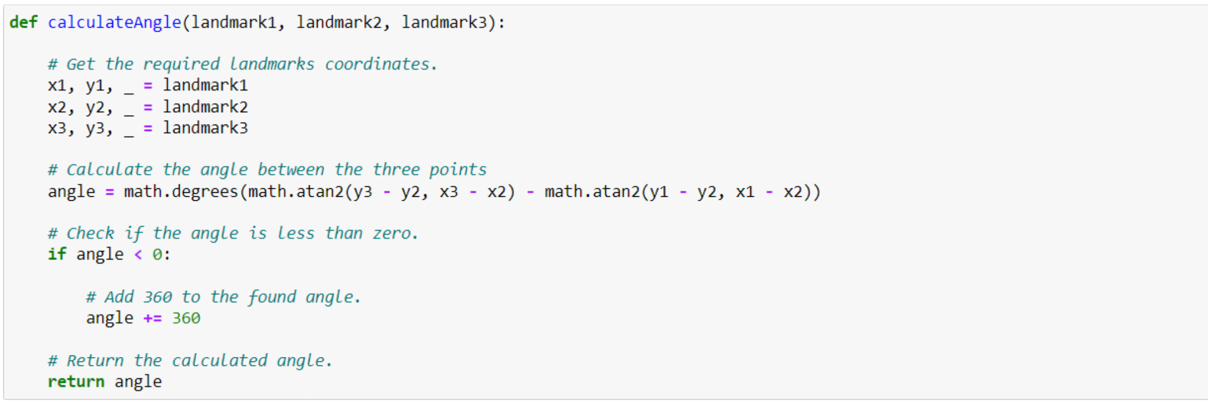


Figure 5.4 Angle Heuristics



```
def calculateAngle(landmark1, landmark2, landmark3):

    # Get the required landmarks coordinates.
    x1, y1, _ = landmark1
    x2, y2, _ = landmark2
    x3, y3, _ = landmark3

    # Calculate the angle between the three points
    angle = math.degrees(math.atan2(y3 - y2, x3 - x2) - math.atan2(y1 - y2, x1 - x2))

    # Check if the angle is less than zero.
    if angle < 0:

        # Add 360 to the found angle.
        angle += 360

    # Return the calculated angle.
    return angle
```

```python
def classifyPose(landmarks, output_image, display=False):

    # Initialize the label of the pose. It is not known at this stage.
    label = 'Unknown Pose'

    # Specify the color (Red) with which the label will be written on the image.
    color = (0, 0, 255)

    # Calculate the required angles.
    #----------------------------------------------------------------------------------------------------------------

    # Get the angle between the left shoulder, elbow and wrist points.
    left_elbow_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value],
                                      landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value],
                                      landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value])

    # Get the angle between the right shoulder, elbow and wrist points.
    right_elbow_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value],
                                       landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value],
                                       landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value])

    # Get the angle between the left elbow, shoulder and hip points.
    left_shoulder_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value],
                                         landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value],
                                         landmarks[mp_pose.PoseLandmark.LEFT_HIP.value])

    # Get the angle between the right hip, shoulder and elbow points.
    right_shoulder_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                                          landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value],
                                          landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value])

    # Get the angle between the left hip, knee and ankle points.
    left_knee_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.LEFT_HIP.value],
                                     landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value],
                                     landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value])

    # Get the angle between the right hip, knee and ankle points
    right_knee_angle = calculateAngle(landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                                      landmarks[mp_pose.PoseLandmark.RIGHT_KNEE.value],
                                      landmarks[mp_pose.PoseLandmark.RIGHT_ANKLE.value])

    #----------------------------------------------------------------------------------------------------------------

    # Check if it is the warrior II pose or the T pose.
    # As for both of them, both arms should be straight and shoulders should be at the specific angle.
    #----------------------------------------------------------------------------------------------------------------

    # Check if the both arms are straight.
    if left_elbow_angle > 165 and left_elbow_angle < 195 and right_elbow_angle > 165 and right_elbow_angle < 195:

        # Check if shoulders are at the required angle.
        if left_shoulder_angle > 80 and left_shoulder_angle < 110 and right_shoulder_angle > 80 and right_shoulder_angle < 110:

    # Check if it is the warrior II pose.
    #----------------------------------------------------------------------------------------------------------------

            # Check if one leg is straight.
            if left_knee_angle > 165 and left_knee_angle < 195 or right_knee_angle > 165 and right_knee_angle < 195:

                # Check if the other leg is bended at the required angle.
                if left_knee_angle > 90 and left_knee_angle < 120 or right_knee_angle > 90 and right_knee_angle < 120:

                    # Specify the label of the pose that is Warrior II pose.
                    label = 'Warrior II Pose'
```

```
#----------------------------------------------------------------------------------------------------------------

# Check if it is the T pose.
#----------------------------------------------------------------------------------------------------------------

        # Check if both legs are straight
        if left_knee_angle > 160 and left_knee_angle < 195 and right_knee_angle > 160 and right_knee_angle < 195:

            # Specify the label of the pose that is tree pose.
            label = 'T Pose'

#----------------------------------------------------------------------------------------------------------------

# Check if it is the tree pose.
#----------------------------------------------------------------------------------------------------------------

# Check if one leg is straight
if left_knee_angle > 165 and left_knee_angle < 195 or right_knee_angle > 165 and right_knee_angle < 195:

    # Check if the other leg is bended at the required angle.
    if left_knee_angle > 315 and left_knee_angle < 335 or right_knee_angle > 25 and right_knee_angle < 45:

        # Specify the label of the pose that is tree pose.
        label = 'Tree Pose'

#----------------------------------------------------------------------------------------------------------------

# Check if the pose is classified successfully
if label != 'Unknown Pose':

    # Update the color (to green) with which the label will be written on the image.
    color = (0, 255, 0)

# Write the label on the output image.
cv2.putText(output_image, label, (10, 30),cv2.FONT_HERSHEY_PLAIN, 2, color, 2)

# Check if the resultant image is specified to be displayed.
if display:

    # Display the resultant image.
    plt.figure(figsize=[10,10])
    plt.imshow(output_image[:,:,::-1]);plt.title("Output Image");plt.axis('off');

else:

    # Return the output image and the classified label.
    return output_image, label
```

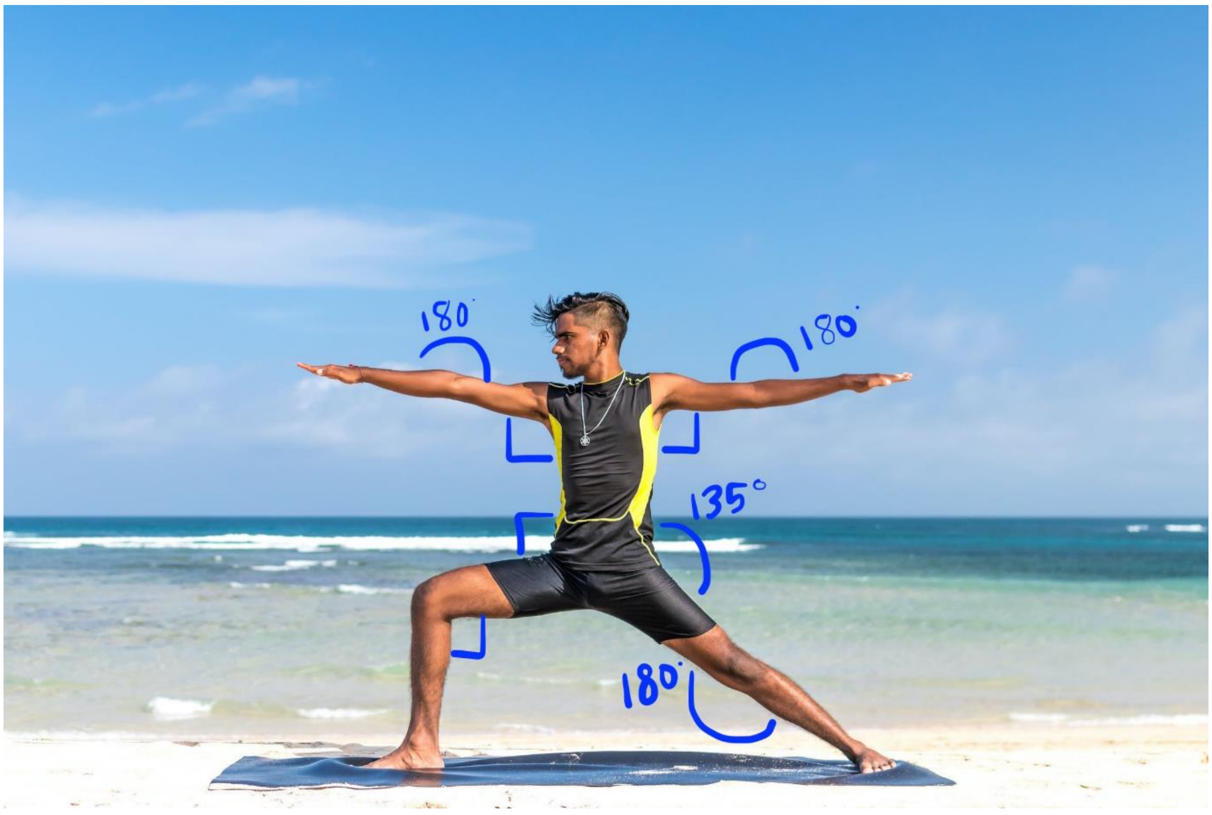


Figure 5.5 Angle Heuristics II

Performing the Pose detection on real time using the real camera feed:

```
import cv2

# Open the default camera (index 0)
cap = cv2.VideoCapture(0)

# Check if the camera is opened successfully
if not cap.isOpened():
    raise IOError("Cannot open Webcam")

# Read frames from the camera and display them
while True:
    ret, frame = cap.read()

    # Resize the frame to 50% of its original size
    frame = cv2.resize(frame, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_AREA)

    # Display the frame in a window named ‘Input’
    cv2.imshow('Input', frame)

    # Wait for a key press
    c = cv2.waitKey()

    # Break the loop if the Esc key is pressed
    if c==27:
        break

# Release the camera and close all windows
cap.release()
cv2.destroyAllWindows()
```

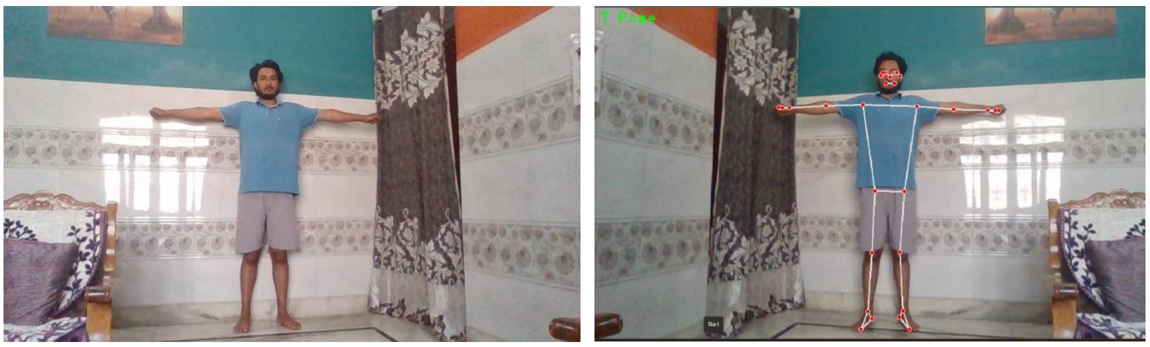
Table 5.2 Testing Camera Mode

```
: # Setup Pose function for video.
pose_video = mp_pose.Pose(static_image_mode=False, min_detection_confidence=0.5, model_complexity=1)

# Initialize the VideoCapture object to read from the webcam.
camera_video = cv2.VideoCapture(0)
camera_video.set(3,1280)
camera_video.set(4,960)

# Initialize a resizable window.
cv2.namedWindow('Pose Classification', cv2.WINDOW_NORMAL)

# Iterate until the webcam is accessed successfully.
while camera_video.isOpened():

    # Read a frame.
    ok, frame = camera_video.read()

    # Check if frame is not read properly.
    if not ok:

        # Continue to the next iteration to read the next frame and ignore the empty camera frame.
        continue

    # Flip the frame horizontally for natural (selfie-view) visualization.
    frame = cv2.flip(frame, 1)

    # Get the width and height of the frame
    frame_height, frame_width, _ =  frame.shape

    # Resize the frame while keeping the aspect ratio.
    frame = cv2.resize(frame, (int(frame_width * (640 / frame_height)), 640))

    # Perform Pose landmark detection.
    frame, landmarks = detectPose(frame, pose_video, display=False)

    # Check if the landmarks are detected.
    if landmarks:

        # Perform the Pose Classification.
        frame, _ = classifyPose(landmarks, frame, display=False)

    # Display the frame.
    cv2.imshow('Pose Classification', frame)

    # Wait until a key is pressed.
    # Retreive the ASCII code of the key pressed
    k = cv2.waitKey(1) & 0xFF

    # Check if 'ESC' is pressed.
    if(k == 27):

        # Break the loop.
        break

# Release the VideoCapture object and close the windows.
camera_video.release()
cv2.destroyAllWindows()
```

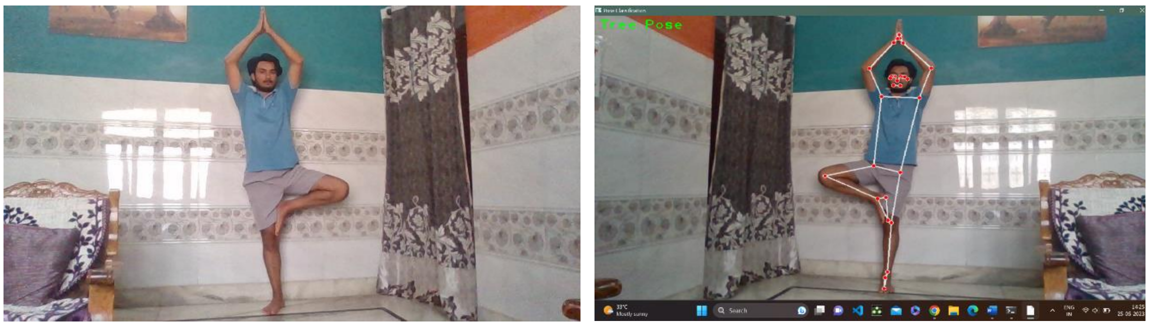
Chapter 6

Testing

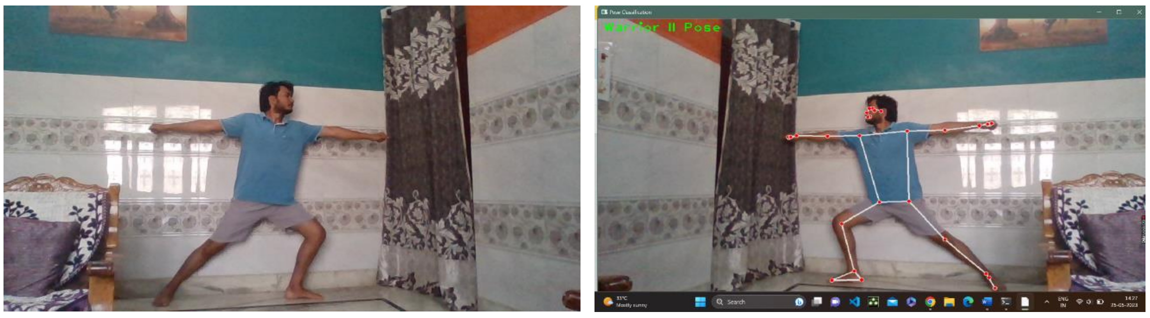
1. **Diverse Pose Scenarios**: Gathering a variety of pictures or videos that illustrate different poses in different situations. Give instances of the standard human positions, such as standing, sitting, walking, jogging, bending, and leaping. Make sure the test materials reflect a range of body types, fashion choices, and social backgrounds.
2. **Challenging Environment**: The system's resilience is tested by testing in different lighting settings, such as those with low light or great contrast, may be involved. Additionally, introducing the cluttered backgrounds, occlusions (partially hidden bodies), and other elements that might influence pose estimation's accuracy.
3. **Multi-person Pose Estimation**: Evaluating the system's ability to handle the multiple persons in the frame. Include pictures or videos of several people, making sure they are doing different things or striking different positions. This will put the system's capacity to recognise and calculate postures for each person accurately to the test.
4. **Tracking and temporal consistency**: Analyse the system's capacity to monitor human postures over time with consistency in terms of both tracking and temporal consistency. Check movies of people moving continuously, for as when dancing or strolling, to see if the posture estimation stays correct and consistent. Test the system's ability to manage posture changes and maintain consistency between successive frames.
5. **Real-time Performance**: Measuring the system’s real-time performance by detecting the video frames by using real-time camera feed. Evaluating the accuracy and stability of pose estimation of the person.



T Pose



Tree Pose



Warrior II Pose

Figure 6.1 Real – time Pose Detection

Chapter 7

Results and Discusssion

A common computer vision problem is human posture identification, which analyses photos or videos to predict the positions of important body joints in a person's pose.

7.1 Results

7.1.1 Pose Detection Output Structure

A. The Pose Detection algorithm using OpenPose and MediaPipe provided a structured output containing the pixel positions and coordinates of the various body landmarks, such as eyes, neck, shoulders, elbows etc

B. The output consisted of a list of body key points, where each key point was identified by a unique label, accompanied by a confidence score and pixel coordinates (x,y) indicating the position of that specific point.

7.1.2 Pose Recognition and Warning System

A. The developed project focused on identifying and recognizing specific poses, namely the T-pose, Tree-pose, and Warrior pose.

B. During the live camera feed, the project accurately detected the landmark positions and compared them with the predefined poses.

C. If the detected pose did not match any of the predefined poses, the system displayed an unknown pose warning signal, indicating that the person's posture was incorrect.

7.1.3 Performance evaluation

A. The performance of the developed project varied depending on the environmental conditions and hardware limitations.

B. In optimal lighting conditions and with sufficient hardware resources, the project worked effectively, accurately recognizing, and classifying the predefined poses.

However, in poor lighting conditions, the project exhibited some lag and inconsistencies in performance.

C. Certain hardware limitations also affected the overall performance, resulting in occasional lag during pose detection and recognition.

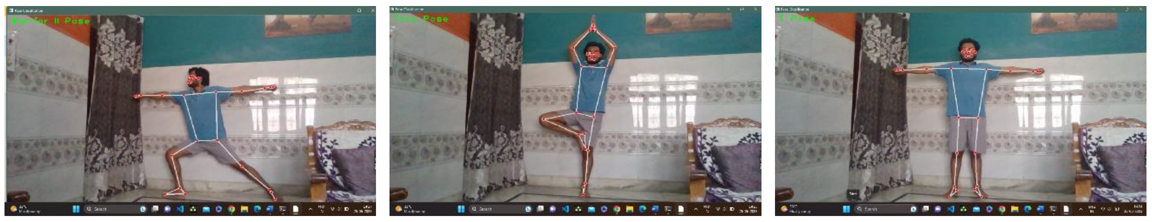


Figure 7.1 Results

7.2 Discussion

The system's viability and efficiency were shown by the selection of the proper libraries and frameworks and the heuristic-based technique for pose categorization.

7.2.1 Choice of Libraries and Frameworks

The success of a project is greatly influenced by the choice of proper libraries and frameworks. The use of Mediapipe, Python, and OpenCV together in this project turned out to be a reliable and effective decision.

- **MediaPipe: Simplifying Development with Pre-trained Models**

A. Pre-trained posture detection model from Mediapipe eliminated the requirement for manual model training. As a result, significant time and money were saved during the development process.

B. Because a pre-trained model was readily available, testing and algorithm improvement could be completed more quickly.

- **Python: Versatile and User-friendly language**

A. Python was the best language for constructing the posture detection and classification algorithm due to its simplicity of use and rich library ecosystem.

B. Python's abundance of libraries for data manipulation, computing, and visualisation sped up the development process and promoted effective coding techniques.

- **OpenCV: Powerful Image Processing and Visualization**

A. For the project's image processing and visualisation needs, OpenCV was a useful tool.
B. Due to the posture detection pipeline's requirements, it was able to efficiently convert between several colour formats, such as BGR to RGB, thanks to its extensive range of functions and algorithms.
C. Additionally, OpenCV included tools for visually representing the identified landmarks, which improved the data' capacity to be understood and analysed.

7.2.2 Heuristic-Based Approach for Pose Classification

In this study, the posture classification approach made use of heuristic angles that were computed from the identified landmarks. This strategy comes with a number of benefits and things to think about.

- **Lightweight and Interpretable Methodology**

A. The posture categorization technique remained lightweight by deriving heuristic angles from the landmark placements, obviating the requirement for intricate machine learning models.
B. Heuristic angles' capacity to be interpreted provided for a clear knowledge of the underlying elements influencing each pose classification.

- **Real-time performance potential**

A. Due to its simplicity and effectiveness, the heuristic-based method has the potential for real-time performance.
B. In real-time applications where speed is essential, calculating heuristic angles straight from the recognised landmarks can be computationally light.

7.2.3 Evaluation and Future Improvements

Several phases of the posture detection and categorization system were assessed throughout the testing phase. The system has positive outcomes, but there is still room for development and more effort.

- **Performance Evaluation and Robustness**

A. The system's precision, dependability, and robustness were evaluated using a variety of measures, such as detection speed, tracking stability, and capacity to handle diverse stances and camera angles.
B. The speed and resilience of the system might be improved by addressing issues like occlusions, different lighting situations, and background clutter.

- **Integration of Deep Learning Techniques**

A. The incorporation of deep learning methods for pose detection and classification may be the subject of future research.
B. Improved accuracy and generalizability, especially in complicated settings, may result from training a customised deep learning model specifically for the posture analysis job.

In conclusion, the project successfully implemented real-time 3D pose detection and pose classification using Mediapipe, Python, and OpenCV. Further improvements, such as addressing challenges and exploring deep learning techniques, hold potential for enhancing the system's performance and expanding its capabilities.

Chapter 8

Conclusion and Future Work

Conclusion:

In this Project, we used MediaPipe and OpenPose to tackle the computer vision issue of identifying human posture. These frameworks worked together to enable us to precisely forecast the locations of key bodily joints during a stance.

A list of body key points, each with an identity, a confidence score, and pixel coordinates indicating the location of the joint, were supplied by OpenPose using MediaPipe's output structure.

The T-position, the Tree-pose, and the Warrior posture were among the unique stances that we successfully produced in our project. The project successfully identified the landmark positions during the live camera feed and gave visual feedback by showing an unknown pose warning signal when the detected pose did not correspond to any predefined poses.

Depending on the hardware restrictions and the environment, the produced project's performance changed. The project successfully recognised and correctly classified positions in ideal lighting conditions and with adequate hardware resources. However, there were sporadic discrepancies and delays in performance due to poor illumination and technology restrictions.

Despite these drawbacks, our experiment proved that utilising OpenPose and MediaPipe to identify and recognise poses was practical and successful. It might be useful for a variety of applications that need to identify human posture, such fitness tracking, virtual reality, and rehabilitation. This project has the potential to become an important resource for encouraging proper posture and offering in-the-moment feedback in a variety of fields with additional enhancements.

Future Work:

1. Multi Model Fusion: Investigate techniques to incorporate additional modalities such as depth information, infrared sensors, or skeleton data from depth cameras to enhance the accuracy and robustness of the pose detection system. Fusion of many modalities can deliver complementing data and enhance performance, particularly in difficult situations like occlusions or dim lighting.
2. Generalization to Unseen Poses and Activities: Enhance the system's capacity for generalisation to previously unseen stances and activities. To do this, several datasets that include a variety of position variations and activities can be gathered or combined.
3. Advancement and Research: The accuracy and robustness of pose detection and classification models are being improved by ongoing research in the field. Deep learning, neural network, and sensor technology advancements may result in pose estimation and categorization techniques that are more accurate and dependable.

Details of Research Publication

References

[1] Abdul Hannan and Faisal Hussan, "Portable Smart Fitness Suite for Real-Time Exercise monitoring and Posture Correction" National Library of Medicine., 08 Oct 2021.

[2] Tucan. PoseEstimation – CoreML, https://github.com/tucan9389/PoseEstimation-CoreML/blob/master/README.md

[3] Sik-Ho Tsang. Review: Cpm — convolutional pose machines (human pose estimation).https://sh-tsang.medium.com/review-cpm-convolutional-pose-machines-human-pose- estimation
224cfeb70aac,2019.

[4] CNN–Graphical Visualization, https://www.analyticssteps.com/blogs/convolutional-neural-network-cnn-graphical- visualization-code-explanation

[5] k-Ho Tsang. Review: Cpm — convolutional pose machines (human pose estimation).https://sh-tsang.medium.com/review-cpm-convolutional-pose-machines-human- pose- estimation-224cfeb70aac,2019.

[6] Rohit Josyula and Sarah Ostadabbas "Review on Human Pose Estimation" Reseacrh Gate., Licencse no. CC-BY-NC-SA 4.0, October 2021.

[7] Vollkorn01. Fitness-Exercise-correction-using-ANN, https://github.com/Vollkorn01/Deep-Learning-Fitness-Exercise-Correction-Keras.

[8]Cao, Z., Simon, T., Wei, S., & Sheikh, Y. (2017). Realtime multi-person 2D pose estimation using part affinity fields. In CVPR (pp. 7291-7299). https://arxiv.org/abs/1611.08050

[9]Kocabas, M., Karagoz, S., & Akbas, E. (2020). OpenPose2ONNX: Towards OpenPose deployment on heterogeneous systems. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (pp. 452-453). https://arxiv.org/abs/2003.10583

[10] Hung, W. C., & Lin, W. Y. (2020). OpenPose for Human Pose Estimation and Tracking in Edge . In 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC) (pp. 632-637). https://ieeexplore.ieee.org/abstract/document/9045745

[11] Yasin, H., Kannala, J., & Rahtu, E. (2020). Monocular 3D human pose estimation by generation and ordinal ranking. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43(2), 550-565. https://ieeexplore.ieee.org/abstract/document/8968130

[12]Prabhu, V., & Yap, P. (2020). Performance Evaluation of OpenPose on Mobile GPU. In 2020 3rd International Conference on Electrical, Computer, Control and Mechatronics Engineering (ECCE 2020) (pp. 1-5). https://ieeexplore.ieee.org/abstract/document/9402542

[13]Haque, S. M., Islam, M. R., Salekin, S., & Haque, M. N. (2020). Efficient Pose Estimation for Robust Human Activity Recognition. In 2020 5th International Conference on Systems, Methodology, Automation and Control (pp. 1-6). https://ieeexplore.ieee.org/abstract/document/9310865

[14] Chen, B., Zhang, D., Huang, Z., & Liang, L. (2021). JugglingPose: A Multi-Person Dataset for Juggling Pose Estimation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (pp. 3776-3785).

[15] Yasin, H., Kannala, J., & Rahtu, E. (2020). Monocular 3D human pose estimation by generation and ordinal ranking. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43(2), 550-565.