FITGENIE

PROJECT WORK

Madhav Choudhary

- LinkedIn - Github

PROJECT FILES

DEMO VIDEO

Project Report:

Figenie - Real-Time Bicep Curl Counter

INDEXING :

1. Introduction

The **Figenie** project represents a cutting-edge application in the field of computer vision and fitness technology, focusing on tracking and counting bicep curls in real-time. Fitness applications utilizing computer vision have seen substantial growth, with an increasing focus on harnessing machine learning and pose estimation to enhance the user experience. Figenie serves as a virtual assistant that not only tracks the count of bicep curls but also provides real-time feedback, thereby transforming the workout experience.

By using a webcam to capture live video, Figenie identifies specific body landmarks, calculates angles between key points (like the shoulder, elbow, and wrist), and evaluates arm position throughout each repetition. The system is designed to detect whether a bicep curl has been fully completed, ensuring accurate tracking of each repetition and preventing partial or incomplete curls from being counted.

Objectives of the Project:

- **Real-time Exercise Tracking**: The application aims to count bicep curls accurately, giving users real-time feedback on their performance.
- **Accuracy in Pose Detection**: By focusing on key body landmarks, the program ensures accurate detection of arm position and movement.
- **Enhanced Workout Experience**: Figenie offers users immediate feedback, making workouts more engaging and interactive.

This project illustrates how computer vision can be applied to improve fitness tracking, bridging the gap between technology and physical training. The name “Figenie” combines "fitness" and "genie," symbolizing the app’s ability to act as a “genie” or assistant for fitness enthusiasts by automating exercise tracking.

---

2. Features of the Project

The Figenie project includes several key features that enable it to function as a real-time exercise tracking tool. Each feature works together to create an interactive, accurate, and user-friendly experience.

Feature Highlights

1. **Real-Time Pose Detection**: The application employs real-time pose estimation to recognize body movements and positions. Real-time detection is crucial for fitness tracking applications as it allows users to receive instant feedback on their form and progress. This feature is built upon the MediaPipe library, a powerful framework that enables the system to identify specific landmarks on the body and monitor movements accurately.
2. **Automated Repetition Counting**: Figenie counts each completed bicep curl based on the motion of the arm, specifically detecting when the arm is fully flexed and extended. By automating repetition counting, the program reduces the need for manual input, enhancing the user’s workout focus. This feature is essential for maintaining motivation and helping users achieve their fitness goals without distraction.
3. **Progress Visualization**: The application provides a visual representation of each curl’s progress, including a progress bar and count display, which updates dynamically on the screen. Progress visualization helps users keep track of their performance during each session, allowing them to set targets and work towards completion.
4. **User-Friendly Interface**: Figenie’s interface is minimalistic and user-centric, displaying only relevant information like the curl count and progress. By reducing clutter, it creates an immersive experience, making it easier for users to focus on their exercise without getting distracted by excessive on-screen elements.

Feature Importance

These features not only make the application more functional and interactive but also ensure that users have a smooth experience. Real-time feedback is vital for maintaining proper form, and the automated counting system removes the need for manual tracking. This combination makes Figenie an effective tool for both beginners and experienced fitness enthusiasts.

---

3. Project Structure

Understanding the structure of the Figenie project is crucial to comprehending how each component contributes to the overall functionality. The project is divided into two main

Python files, each serving a specific role. This modular structure enables scalability, making it easy to add new functionalities or modify existing ones without disrupting the entire codebase.

Project Components

1. **app.py**: The main application file that initializes the bicep curl counter and manages the core functionality, including video capture, frame processing, and displaying the count and progress. This file acts as the "brain" of the application, coordinating inputs from the user's webcam and sending data to PoseModule.py for pose detection. It's designed to handle real-time data, ensuring smooth, frame-by-frame processing without lag.
2. **PoseModule.py**: A helper module that defines the poseDetector class, which is responsible for detecting body landmarks and calculating angles between them. This modular approach separates pose detection and analysis from the main program logic, making the code more organized and manageable. The poseDetector class uses MediaPipe for pose detection and provides several methods that simplify the process of detecting specific body landmarks, such as the shoulder, elbow, and wrist.

Directory Structure

Organizing the files in a straightforward directory structure enhances readability and modularity:

```
|-- project_directory/
    |-- app.py
    |-- PoseModule.py
```

Benefits of Project Structure

The separation of pose detection (`PoseModule.py`) from the main application (`app.py`) allows each component to be developed, tested, and optimized independently. This structure also facilitates the addition of new exercises or features in the future, as each module is self-contained. For example, if the project were to incorporate squats or push-ups, additional modules could be created to handle these specific exercises without interfering with the existing bicep curl tracking logic.

---

4. Code Analysis

4.1. File 1: app.py

```
import cv2
import numpy as np
import PoseModule as pm
class BicepCurlCounter:
```

```
    def __init__(self):
        self.detector = pm.poseDetector()
        self.count = 0
        self.direction = 0

    def process_frame(self, img):
        maxh, maxw, _ = img.shape

        # Process frame with pose estimation
        img = self.detector.findPose(img, False)
        lmList = self.detector.findPosition(img, False)

        # Check if landmarks are detected
        if len(lmList) != 0:
            # Calculate angle
            angle = self.detector.findAngle(img, 12, 14, 16)

            # Interpolate for percentage and bar level
            per = np.interp(angle, (55, 145), (100, 0))
            bar = np.interp(angle, (55, 145), (400, 50))

            # Check for the dumbbell curls
            color = (255, 0, 255)
            if per == 100:
                color = (0, 255, 0)
                if self.direction == 0:
                    self.count += 0.5
                    self.direction = 1
            if per == 0:
                color = (0, 255, 0)
                if self.direction == 1:
                    self.count += 0.5
                    self.direction = 0

            # Draw bar
            cv2.rectangle(img, (maxw-100, maxh-400), (maxw - 50, maxh - 50), color,
3)
            cv2.rectangle(img, (maxw - 100, maxh - int(bar)), (maxw - 50, maxh -
50), color, cv2.FILLED)
            cv2.putText(img, f'{int(per)} %', (maxw - 105, maxh - 430),
cv2.FONT_HERSHEY_PLAIN, 2, color, 2)

            # Draw Curl Count
            cv2.rectangle(img, (0, 0), (150, 100), (0, 255, 0), cv2.FILLED)
            cv2.putText(img, str(int(self.count)), (40, 70),
cv2.FONT_HERSHEY_PLAIN, 4, (255, 0, 0), 4)

        return img

def main():
    # Initialize video capture from webcam
```

```
    cap = cv2.VideoCapture(0)  # 0 is usually the default camera
    counter = BicepCurlCounter()

    while cap.isOpened():
        success, frame = cap.read()
        if not success:
            break

        # Process the frame and display the results
        frame = counter.process_frame(frame)
        cv2.imshow("Bicep Curl Counter", frame)

        # Press 'q' to quit
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release resources
    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

4.1.1. Overview and Purpose

The `app.py` file is the main program that captures video from the webcam, processes each frame to detect bicep curls, and displays the results. It uses the `poseDetector` class from `PoseModule.py` to detect body landmarks and calculate the arm's angle, allowing it to count each completed bicep curl.

4.1.2. Code Walkthrough of app.py

4.1.2.1. Imports and Class Initialization

```
import cv2
import numpy as np
import PoseModule as pm
```

- Imports the **OpenCV** library (`cv2`) for video capture and display.
- Imports **NumPy** (`np`) for mathematical calculations.
- Imports the custom `PoseModule`, which contains the `poseDetector` class for pose estimation.

4.1.2.2. BicepCurlCounter Class Definition

```
class BicepCurlCounter:
    def __init__(self):
        self.detector = pm.poseDetector()
        self.count = 0
        self.direction = 0
```

- Defines a class called BicepCurlCounter that will be responsible for detecting bicep curls.
- **Attributes**:
    - self.detector: An instance of poseDetector used for pose detection.
    - self.count: Tracks the total number of completed bicep curls.
    - self.direction: Keeps track of the current direction of arm movement, helping to identify full repetitions.

4.1.2.3. Frame Processing with process_frame

```
def process_frame(self, img):
    img = self.detector.findPose(img, False)
    lmList = self.detector.findPosition(img, False)
    if len(lmList) != 0:
        angle = self.detector.findAngle(img, 12, 14, 16)
```

- **findPose** and **findPosition**: These functions are used to detect the pose and find key landmarks, returning a list lmList with coordinates of all detected body points.
- The angle of the arm (shoulder, elbow, wrist) is calculated using findAngle. This angle will be used to determine the position of the bicep curl (whether extended or flexed).

4.1.2.4. Calculating Curl Status with Interpolation

```
per = np.interp(angle, (55, 145), (100, 0))
bar = np.interp(angle, (55, 145), (400, 50))
```

The np.interp function is used to map the angle to a **percentage (per)** and a **progress bar level (bar)**.

- per represents how much of the curl is complete (100 for fully extended and 0 for fully flexed).

- bar is used for the display of a progress bar indicating the curl's current position.

4.1.2.5. Curl Counting Logic

```
if per == 100:
    color = (0, 255, 0)
    if self.direction == 0:
        self.count += 0.5
        self.direction = 1
if per == 0:
    if self.direction == 1:
        self.count += 0.5
        self.direction = 0
```

- When per reaches 100 (indicating the arm is fully extended), the counter checks the movement direction:
- If the previous direction was "down," it increments count by 0.5 and changes direction to 1, marking the arm's transition to an extended position.
- When per reaches 0 (indicating the arm is fully flexed), it completes the repetition and increments count again by 0.5, resulting in a complete curl count.
- This logic ensures that only full up-and-down cycles are counted as complete curls.

4.1.2.6. Displaying Real-Time Output

```
cv2.putText(img, str(int(self.count)), (40, 70), cv2.FONT_HERSHEY_PLAIN, 4, (255,
0, 0), 4)
```

- Adds the current count on the display, allowing the user to see their progress in real time.

4.1.2.7. Main Loop for Video Capture

```
def main():
    cap = cv2.VideoCapture(0)
    counter = BicepCurlCounter()
    while cap.isOpened():
        success, frame = cap.read()
        frame = counter.process_frame(frame)
        cv2.imshow("Bicep Curl Counter", frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

- Sets up the video capture, continuously processing each frame.
- process_frame is called on each frame, and the processed image is displayed in a window titled "Bicep Curl Counter."
- The loop ends if the user presses "q."

4.2. PoseModule.py

```
import cv2
import mediapipe as mp
import time
import math


class poseDetector():

    def __init__(self, mode=False, upBody=False, smooth=True,
                 detectionCon=0.5, trackCon=0.5):

        self.mode = mode
        self.upBody = upBody
        self.smooth = smooth
        self.detectionCon = detectionCon
        self.trackCon = trackCon

        self.mpDraw = mp.solutions.drawing_utils
        self.mpPose = mp.solutions.pose
        #self.pose = self.mpPose.Pose(self.mode, self.upBody, self.smooth,
self.detectionCon, self.trackCon)
        self.pose = self.mpPose.Pose(self.mode,min_detection_confidence=0.5,
min_tracking_confidence=0.5)

    def findPose(self, img, draw=True):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.pose.process(imgRGB)
```

```
        if self.results.pose_landmarks:
            if draw:
                self.mpDraw.draw_landmarks(img, self.results.pose_landmarks,
self.mpPose.POSE_CONNECTIONS)
        return img

    def findPosition(self, img, draw=True):
        self.lmList = []
        if self.results.pose_landmarks:
            for id, lm in enumerate(self.results.pose_landmarks.landmark):
                h, w, c = img.shape
                # print(id, lm)
                cx, cy = int(lm.x * w), int(lm.y * h)
                self.lmList.append([id, cx, cy])
                if draw:
                    cv2.circle(img, (cx, cy), 5, (255, 0, 0), cv2.FILLED)
        return self.lmList

    def findAngle(self, img, p1, p2, p3, draw=True):

        # Get the landmarks
        x1, y1 = self.lmList[p1][1:]
        x2, y2 = self.lmList[p2][1:]
        x3, y3 = self.lmList[p3][1:]

        # Calculate the Angle
        angle = math.degrees(math.atan2(y3 - y2, x3 - x2) -
                             math.atan2(y1 - y2, x1 - x2))
        if angle < 0:
            angle += 360

        # print(angle)

        # Draw
        if draw:
            cv2.line(img, (x1, y1), (x2, y2), (255, 255, 255), 3)
            cv2.line(img, (x3, y3), (x2, y2), (255, 255, 255), 3)
            cv2.circle(img, (x1, y1), 10, (0, 0, 255), cv2.FILLED)
            cv2.circle(img, (x1, y1), 15, (0, 0, 255), 2)
            cv2.circle(img, (x2, y2), 10, (0, 0, 255), cv2.FILLED)
            cv2.circle(img, (x2, y2), 15, (0, 0, 255), 2)
            cv2.circle(img, (x3, y3), 10, (0, 0, 255), cv2.FILLED)
            cv2.circle(img, (x3, y3), 15, (0, 0, 255), 2)
            cv2.putText(img, str(int(angle)), (x2 - 50, y2 + 50),
                        cv2.FONT_HERSHEY_PLAIN, 2, (0, 0, 255), 2)
        return angle

def main():
    cap = cv2.VideoCapture(0)
    pTime = 0
    detector = poseDetector()
```

```
    while True:
        success, img = cap.read()
        img = detector.findPose(img)
        lmList = detector.findPosition(img, draw=False)
        if len(lmList) != 0:
            print(lmList[14])
            cv2.circle(img, (lmList[14][1], lmList[14][2]), 15, (0, 0, 255),
cv2.FILLED)

        cTime = time.time()
        fps = 1 / (cTime - pTime)
        pTime = cTime

        cv2.putText(img, str(int(fps)), (70, 50), cv2.FONT_HERSHEY_PLAIN, 3,
                    (255, 0, 0), 3)

        cv2.imshow("Image", img)
        cv2.waitKey(1)


if __name__ == "__main__":
    main()
```

4.2.1. Overview and Purpose

The `PoseModule.py` file defines a custom `poseDetector` class, which serves as a helper class for pose detection and angle calculation. This class wraps around Google's MediaPipe framework, providing easy-to-use methods to extract pose information and calculate angles between specific body parts.

4.2.2. Code Walkthrough of PoseModule.py

4.2.2.1. Imports and Initialization

```
import cv2
import mediapipe as mp
import math
```

- Imports **OpenCV** for image processing, **MediaPipe** for pose estimation, and **math** for angle calculations.

4.2.2.2. poseDetector Class Definition

```
class poseDetector:
    def __init__(self, mode=False, upBody=False, smooth=True, detectionCon=0.5,
trackCon=0.5):
```

```
        self.mpDraw = mp.solutions.drawing_utils
        self.mpPose = mp.solutions.pose
        self.pose = self.mpPose.Pose(mode, upBody, smooth, detectionCon, trackCon)
```

Initializes the poseDetector class with MediaPipe's pose detection settings:

- **mpDraw**: MediaPipe drawing utilities used to draw landmarks.
- **mpPose**: MediaPipe pose module used to access the Pose class.
- **self.pose**: The pose detection model with parameters for customization.

4.2.2.3. Pose Detection with findPose

```
def findPose(self, img, draw=True):
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    self.results = self.pose.process(imgRGB)
    if self.results.pose_landmarks:
        if draw:
            self.mpDraw.draw_landmarks(img, self.results.pose_landmarks,
self.mpPose.POSE_CONNECTIONS)
    return img
```

- Converts the input frame to RGB (required by MediaPipe) and runs the pose detection model on the frame.
- If landmarks are detected, it draws them on the image for visual feedback.

4.2.2.4. findPosition to Extract Landmark Positions

```
def findPosition(self, img, draw=True):
    lmList = []
    if self.results.pose_landmarks:
        for id, lm in enumerate(self.results.pose_landmarks.landmark):
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            lmList.append([id, cx, cy])
            if draw:
                cv2.circle(img, (cx, cy), 5, (255, 0, 0), cv2.FILLED)
    return lmList
```

- Loops through detected landmarks to create lmList, a list containing each landmark's ID and its (x, y) coordinates.
- If draw is enabled, it marks the landmark positions on the image.

4.2.2.5. findAngle to Calculate Angle Between Landmarks

```
def findAngle(self, img, p1, p2, p3, draw=True):
    x1, y1 = lmList[p1][1:]
    x2, y2 = lmList[p2][1:]
    x3, y3 = lmList[p3][1:]

    angle = math.degrees(math.atan2(y3 - y2, x3 - x2) - math.atan2(y1 - y2, x1 -
x2))
    if angle < 0:
        angle += 360
    if draw:
        cv2.putText(img, str(int(angle)), (x2 - 50, y2 + 50),
cv2.FONT_HERSHEY_PLAIN, 2, (0, 0, 255), 2)
    return angle
```

- Takes three landmark points (such as shoulder, elbow, wrist) and calculates the angle at the middle point using trigonometry.
- This angle is critical for determining the arm's flexion and extension.
- Optionally, the angle can be displayed on the image for visual feedback.

5. Summary

5.1. Of app.py

The `app.py` file captures live video from the webcam and uses pose detection to recognize arm movement, calculate angles, and count completed bicep curls. This program serves as the user interface for real-time feedback and visualization, showing the curl count and current position on the screen.

5.2. Of PoseModule.py

The `PoseModule.py` file encapsulates the pose estimation functionality using MediaPipe, providing helper methods to detect poses, extract landmarks, and calculate angles. This module simplifies pose detection, allowing the main program (`app.py`) to focus on curl counting logic without dealing directly with complex pose calculations.

5.3. Overall Summary

Together, `app.py` and `PoseModule.py` create a real-time bicep curl counter. `app.py` captures video, processes each frame to detect body movement, and tracks arm position to count bicep curls. `PoseModule.py` provides the core pose detection and angle calculation functions, simplifying the curl counting logic in `app.py`. This setup allows for accurate exercise tracking and real-time feedback, highlighting the power of computer vision for fitness applications.

6. How the Code Works

The Figenie application functions by capturing a live video feed, detecting key body landmarks, calculating angles between joints, and tracking arm movement to determine bicep curls. Below is a breakdown of how each part of the code contributes to this process.

Step-by-Step Code Execution

1. **Capture Video Feed**: The first step is to initialize the webcam and capture each frame. This is done using OpenCV, which allows real-time access to the webcam. Each frame is then processed to detect the user's arm position and calculate repetitions. OpenCV's `VideoCapture` method is instrumental in obtaining the video feed, making it available frame by frame for processing.
2. **Pose Detection**: Using MediaPipe, specific body landmarks are identified, focusing on the shoulder, elbow, and wrist. These landmarks are essential for calculating the arm's angle and determining the range of motion required for a bicep curl. Pose detection is performed in `PoseModule.py`, where each frame is analyzed to identify and label key points on the body. This process relies on the machine learning model provided by MediaPipe, which is pre-trained to recognize human poses accurately.
3. **Angle Calculation**: In order to track the bicep curl motion, the angle between the shoulder, elbow, and wrist is calculated. This angle helps identify when the arm is fully flexed (at the peak of the curl) and when it is fully extended (at the bottom of the curl). By determining these positions, the program can count each completed curl accurately. The angle calculation is implemented using trigonometric functions from the `math` library.
4. **Repetition Counting Logic**: A complete bicep curl is counted when the user's arm moves from a fully extended position to a fully flexed position and back again. To avoid counting partial curls, the program tracks the arm's movement direction and only increments the count when a full cycle is detected. This logic ensures that each counted repetition is a complete bicep curl.
5. **Display Output**: Once the curl count is updated, the application displays the current count and a progress bar on the screen. This visual feedback helps the user keep track of their exercise progress in real-time. The count display and progress bar are created using OpenCV's drawing functions, ensuring the output remains clear and easy to read.

Importance of Real-Time Processing

The application's ability to process video frames in real-time is crucial for providing accurate and timely feedback. Lag or delay in processing could result in inaccurate counts, potentially disrupting the user's workout. By leveraging efficient libraries like OpenCV and MediaPipe, the application ensures smooth and continuous tracking.

7. Frameworks Used

7.1. OpenCV (cv2)

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning library designed to streamline the development of applications in the fields of image processing, computer vision, and machine learning. OpenCV provides various tools and algorithms for image recognition, object detection, and real-time video processing.

Usage in the Project

OpenCV is primarily used for:

- **Capturing Video Frames**: In app.py, cv2.VideoCapture(0) captures real-time video from the webcam.
- **Displaying Frames**: cv2.imshow is used to display the video feed with overlayed text and shapes in real time.
- **Drawing Shapes and Text**: Functions like cv2.rectangle and cv2.putText are used to create visual elements (e.g., count display, progress bar) on the output frame.
- **Converting Image Color Spaces**: In PoseModule.py, cv2.cvtColor is used to convert frames from BGR to RGB color space for compatibility with MediaPipe.

7.2. MediaPipe

MediaPipe is a cross-platform framework developed by Google for building multimodal (e.g., audio, video) and real-time machine learning pipelines. MediaPipe Pose is a solution within MediaPipe that uses machine learning models to recognize human poses by detecting body landmarks in real-time. It's highly optimized for speed and accuracy, making it ideal for real-time applications like gesture recognition and fitness tracking.

Usage in the Project

MediaPipe is central to the project's pose detection functionality:

- **Pose Estimation**: In PoseModule.py, mp.solutions.pose is used to create an instance of the Pose class, which is responsible for detecting human poses from video frames.
- **Drawing Landmarks and Connections**: mp.solutions.drawing_utils allows for drawing detected landmarks and connections between them, helping visualize the detected pose on the video feed.
- **Extracting Pose Landmarks**: The Pose class provides a list of detected body landmarks, each represented by an ID and its (x, y, z) coordinates. These landmarks are used to calculate angles for bicep curl counting.

7.3. NumPy

NumPy is a Python library for numerical and scientific computing. It provides a high-performance array object and tools for working with arrays, enabling efficient operations on large datasets, mathematical computations, and linear algebra functions.

Usage in the Project

NumPy is used in app.py to:

- **Interpolate Values**: np.interp is used to map the angle of the arm to a progress percentage and progress bar value. This interpolation helps in giving users a smoother visual feedback as they perform curls.
- **Efficient Mathematical Calculations**: NumPy's efficient array-based calculations are ideal for processing large volumes of data and performing mathematical transformations without impacting performance.

7.4. Math Library

The **math** module in Python provides mathematical functions, including trigonometric functions, exponential functions, and other useful utilities for mathematical computations.

Usage in the Project

In PoseModule.py, the math library is used to:

- **Calculate Angles Between Points**: The math.atan2 function is used to compute the angle between three landmarks (e.g., shoulder, elbow, and wrist) on the arm. This angle calculation is crucial for determining the position of the arm and counting the bicep curls accurately.
- **Convert Radians to Degrees**: math.degrees converts the angle calculated in radians to degrees, making it easier to interpret and use in the context of the arm's position.

Summary of Framework Usage

Together, these frameworks and libraries enable the Figenie project to capture and process real-time video frames, detect human poses, calculate joint angles, and visually display results to the user. Here's how each library contributes:

- **OpenCV** handles all video-related tasks, including capturing frames, displaying output, and drawing on images.
- **MediaPipe** provides robust and efficient pose detection, making it possible to extract real-time data about body positions and landmarks.
- **NumPy** performs numerical operations, such as interpolation, to enhance visual feedback.
- **Math** helps calculate the precise angles needed to monitor and track the user's movements.

This combination of tools allows for smooth, real-time tracking of bicep curls, providing accurate and timely feedback to users during their workout.

8. Scope of Future Enhancements

The current version of Figenie is focused on counting bicep curls, but there are numerous possibilities for future development. Expanding the application's functionality and improving user experience could significantly enhance its appeal and usability.

Potential Enhancements

1. **Tracking Multiple Exercises**: Currently, the program is designed to count only bicep curls. Adding the capability to recognize and track other exercises, like squats, lunges, or push-ups, could broaden its usefulness. This expansion would involve creating new detection modules in `PoseModule.py` to recognize different body movements and positions.
2. **Enhanced Feedback Mechanisms**: Providing additional feedback, such as vocal cues or form correction suggestions, could make the application more interactive. For example, if the user's form is incorrect (e.g., not reaching full extension), the application could provide voice prompts to encourage proper technique. This would require integrating a text-to-speech system and logic to assess form accuracy.
3. **Increased Accuracy in Pose Detection**: While MediaPipe provides high accuracy, there may be cases where pose detection struggles due to lighting, background, or camera angle. Developing a model that adapts to various environments could improve detection. Adding more robust lighting adjustment features or even creating a specialized version of MediaPipe's pose model for fitness could further improve the system's reliability.
4. **Workout Analytics and Tracking**: Recording workout data, such as total reps, speed, and rest periods, could offer users insights into their performance over time. This data could then be visualized using graphs or charts, allowing users to monitor progress and set goals. Integrating a database to store this data and using a visualization library (like Matplotlib) could help implement this feature.

Value of Future Enhancements

Each proposed enhancement aims to improve the user experience and make Figenie a more versatile fitness tool. By expanding its capabilities beyond bicep curls, Figenie can become a comprehensive exercise tracker. Enhanced feedback and analytics could make it suitable for both beginner and advanced users, allowing it to cater to a broader audience.

---

9. Key Sections of the Code

To understand the Figenie project thoroughly, it's crucial to break down its codebase into logical sections, each contributing to the program's functionality. Both files, `app.py` and `PoseModule.py`, contain code segments that serve specific roles in capturing video input, detecting poses, calculating arm angles, counting repetitions, and displaying the output.

9.1. Initialization and Imports

This section involves setting up the required libraries and dependencies, which are foundational to the entire project.

- **OpenCV (cv2)**: This library is essential for real-time video processing. By importing `cv2`, we enable the program to access the webcam, read frames, and draw graphical elements like lines and circles on the video feed.
- **Mediapipe**: A framework used for pose detection. The `mp.solutions.pose` module provides a pretrained model that recognizes human poses, which is instrumental in detecting body landmarks.
- **Math Module**: The `math` library provides trigonometric functions for angle calculation. By using methods like `atan2` and `degrees`, the program can calculate the angle between landmarks.

These imports ensure the project has the necessary tools for real-time pose detection and analysis.

9.2. Pose Detection and Landmark Identification (PoseModule.py)

The pose detection logic is encapsulated in PoseModule.py within a class named poseDetector. This module leverages MediaPipe to detect and draw body landmarks, making it a critical component in accurately identifying arm positions.

- **poseDetector Class**: This class initializes a pose detection model and includes functions to detect landmarks, such as the shoulder, elbow, and wrist.
    - **findPose()**: A method that takes a video frame as input and applies pose detection on it, returning a modified frame with visual landmarks drawn.
    - **findPosition()**: A method that extracts the x and y coordinates of each detected landmark, which are then used to calculate the angle between the shoulder, elbow, and wrist. This is essential for determining arm movement and curl completion.

By centralizing pose detection in PoseModule.py, the code achieves modularity, making it easy to maintain and extend.

9.3. Angle Calculation (PoseModule.py)

This component of the code calculates the angle between the shoulder, elbow, and wrist using trigonometric functions from the `math` library. Angle calculation is vital because it helps determine whether a full curl has been completed.

- **findAngle()**: This function receives the x and y coordinates of the shoulder, elbow, and wrist, calculates the angle between them, and returns this angle. The angle determines whether the arm is in a flexed or extended position.
- **Trigonometric Calculations**: By applying `atan2` and converting the result to degrees, the program calculates the angle accurately, ensuring the bicep curl count is reliable.

The code's accuracy in angle calculation directly influences the effectiveness of repetition counting.

9.4. Repetition Counting Logic (app.py)

The repetition counting logic in `app.py` ensures that only full curls are counted by tracking the arm's movement direction and detecting when a full curl cycle (from extension to flexion and back) is completed.

- **Counter Variable**: This variable increments only when a complete curl cycle is detected, ensuring that partial curls aren't mistakenly counted.
- **Curl Detection Logic**: The program monitors the arm's direction of movement and counts a curl only after a full extension and full flexion cycle. For instance, the arm must reach below a certain angle (full extension) before the program counts the next repetition.
- **Directional State Tracking**: The logic tracks whether the arm is moving up or down, which helps prevent double-counting of partial repetitions.

This section plays a critical role in the program's functionality by making the curl count accurate and reliable.

9.5. Displaying Output and Real-Time Feedback (app.py)

This section involves drawing visual elements on the frame, such as the repetition count and a progress bar, to provide real-time feedback to the user.

- **cv2.putText()**: This function from OpenCV adds text on the video feed to show the number of completed repetitions.
- **Progress Bar**: The progress bar visually indicates the completion level of each curl, using a rectangle that dynamically adjusts its size based on the arm's angle.
- **Display Logic**: The progress bar and repetition count update in real-time, providing instant feedback on the user's performance and enhancing the interactive experience.

The display elements make the user interface engaging and intuitive, allowing users to stay focused on their workout.

9.6. Real-Time Processing and Looping (app.py)

This section manages the main application loop, which processes each frame from the webcam and runs it through the pose detection and counting logic.

- **Video Capture Loop**: Using a `while` loop, the program continuously captures frames from the webcam and processes each frame individually. This real-time processing loop ensures smooth frame-by-frame analysis.
- **Frame-by-Frame Analysis**: Each frame is passed to the `poseDetector` object for landmark detection, angle calculation, and repetition counting.
- **Exit Condition**: The loop runs continuously until the user manually stops the program (e.g., by pressing a key).

This loop is critical for ensuring the program operates smoothly and updates in real-time.

10. Conclusion

The Figenie project showcases the powerful intersection of computer vision and fitness tracking, providing users with an automated, interactive experience for tracking bicep curls. By using a webcam, Figenie captures live video, detects body landmarks, calculates angles, and counts each curl, providing real-time feedback on workout performance. With its modular code structure and efficient use of MediaPipe and OpenCV, the project achieves a high level of accuracy in tracking arm movement and counting repetitions, making it a reliable virtual fitness assistant.

Key Achievements of the Project

- **Real-Time Exercise Tracking**: Figenie provides real-time tracking and feedback on bicep curls, a feature essential for users who want immediate insights into their form and progress.
- **Automated Repetition Counting**: The program's logic ensures that each repetition is accurately counted, eliminating the need for manual tracking and enhancing workout focus.
- **Modular and Scalable Design**: By dividing functionality between `app.py` and `PoseModule.py`, the project is designed with scalability in mind. This structure allows for future additions of new exercises or features without requiring a complete overhaul of the code.

In conclusion, Figenie exemplifies the integration of computer vision with fitness tracking, offering a real-time, accurate, and interactive experience. The project not only achieves its objective of counting bicep curls effectively but also opens doors for further advancements in personal fitness innovation. With additional development, Figenie could transform into a full-featured fitness assistant, benefiting users in their journey toward improved physical health and performance.

***