Aim

To create a real-time bicep curl counter using computer vision techniques to track and count the number of bicep curls performed by a user.

Objective

The primary objective is to leverage the Mediapipe library to detect and track the user's arm movements, specifically to count the number of bicep curls executed. The system should be able to process video input, detect key body landmarks, and calculate angles to determine curl repetitions accurately.

Summary

The project involves using the Mediapipe library for pose estimation to analyze video feed from a webcam. By extracting key landmarks from the user's arm, the system calculates the angle at the elbow to detect the curl movement. The code then counts the number of curls and displays this information along with the stage of the exercise on the video feed.

Tools and Libraries Used

- **OpenCV**: For handling video capture and displaying the video feed.
- **Mediapipe**: For pose detection and landmark extraction.
- **NumPy**: For angle calculations and array manipulations.

Procedure

1. **Setup and Initialization**:
    - Install the required libraries using pip.
    - Import the necessary libraries (cv2, mediapipe, and numpy).
2. **Video Feed Capture**:
    - Capture video from the webcam using OpenCV.
    - Display the video feed in a window.
3. **Pose Detection Setup**:
    - Initialize Mediapipe's Pose module with specified confidence levels.
    - Convert the captured frames to RGB and process them to detect pose landmarks.
4. **Angle Calculation**:
    - Define a function to calculate the angle between three points (shoulder, elbow, wrist) using trigonometry.
    - Extract landmarks and calculate the angle at the elbow to assess the curl movement.
5. **Curl Counting Logic**:

- Define variables to track the number of curls and the current stage (up or down).
- Use angle thresholds to determine if a curl is performed and update the count accordingly.
- Display the curl count and current stage on the video feed.

6. **Display and Termination**:
   - Render the detected landmarks and current exercise data on the video feed.
   - Break the loop and release resources when 'q' is pressed.

CODE:

```
!pip install mediapipe opencv-python

import cv2

import mediapipe as mp

import numpy as np

mp_drawing = mp.solutions.drawing_utils

mp_pose = mp.solutions.pose

# VIDEO FEED

cap = cv2.VideoCapture(0)

while cap.isOpened():

   ret, frame = cap.read()

   cv2.imshow('Mediapipe Feed', frame)


   if cv2.waitKey(10) & 0xFF == ord('q'):

      break


cap.release()

cv2.destroyAllWindows()

cap = cv2.VideoCapture(0)

## Setup mediapipe instance

with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:

   while cap.isOpened():
```

```
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=2),
                                  mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                   )

        cv2.imshow('Mediapipe Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
cap = cv2.VideoCapture(0)
## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
```

```
while cap.isOpened():
    ret, frame = cap.read()

    # Recolor image to RGB
    image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False

    # Make detection
    results = pose.process(image)

    # Recolor back to BGR
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    # Extract landmarks
    try:
        landmarks = results.pose_landmarks.landmark
        print(landmarks)
    except:
        pass



    # Render detections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=2),
                              mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                              )

    cv2.imshow('Mediapipe Feed', image)
```

```
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
len(landmarks)
for lndmrk in mp_pose.PoseLandmark:
    print(lndmrk)
landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].visibility
landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value]
landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value]
def calculate_angle(a,b,c):
    a = np.array(a) # First
    b = np.array(b) # Mid
    c = np.array(c) # End

    radians = np.arctan2(c[1]-b[1], c[0]-b[0]) - np.arctan2(a[1]-b[1], a[0]-b[0])
    angle = np.abs(radians*180.0/np.pi)

    if angle >180.0:
        angle = 360-angle

    return angle
shoulder =
[landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmarks[mp_pose.PoseLandmark.LEFT
_SHOULDER.value].y]
elbow =
[landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmarks[mp_pose.PoseLandmark.LEFT_ELB
OW.value].y]
```

```
wrist =
[landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]

shoulder, elbow, wrist

calculate_angle(shoulder, elbow, wrist)

tuple(np.multiply(elbow, [640, 480]).astype(int))

cap = cv2.VideoCapture(0)

## Setup mediapipe instance

with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:

  while cap.isOpened():

    ret, frame = cap.read()


    # Recolor image to RGB

    image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    image.flags.writeable = False


    # Make detection

    results = pose.process(image)


    # Recolor back to BGR

    image.flags.writeable = True

    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)


    # Extract landmarks

    try:

      landmarks = results.pose_landmarks.landmark


      # Get coordinates
```

```
            shoulder =
[landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmarks[mp_pose.PoseLandmark.LEFT
_SHOULDER.value].y]
            elbow =
[landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmarks[mp_pose.PoseLandmark.LEFT_ELB
OW.value].y]
            wrist =
[landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmarks[mp_pose.PoseLandmark.LEFT_WRI
ST.value].y]


            # Calculate angle
            angle = calculate_angle(shoulder, elbow, wrist)


            # Visualize angle
            cv2.putText(image, str(angle),
                        tuple(np.multiply(elbow, [640, 480]).astype(int)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA
                            )

        except:
            pass



        # Render detections
        mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS,
                                  mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=2),
                                  mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                                   )

        cv2.imshow('Mediapipe Feed', image)
```

```
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
cap = cv2.VideoCapture(0)

# Curl counter variables
counter = 0
stage = None

## Setup mediapipe instance
with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
    while cap.isOpened():
        ret, frame = cap.read()

        # Recolor image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Make detection
        results = pose.process(image)

        # Recolor back to BGR
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        # Extract landmarks
        try:
```

```
        landmarks = results.pose_landmarks.landmark


        # Get coordinates
        shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x,landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y]
        elbow = [landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].x,landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value].y]
        wrist = [landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].x,landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value].y]


        # Calculate angle
        angle = calculate_angle(shoulder, elbow, wrist)


        # Visualize angle
        cv2.putText(image, str(angle),
                    tuple(np.multiply(elbow, [640, 480]).astype(int)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA
                        )


        # Curl counter logic
        if angle > 160:
            stage = "down"
        if angle < 30 and stage =='down':
            stage="up"
            counter +=1
            print(counter)


    except:
```

```
    pass

# Render curl counter
# Setup status box
cv2.rectangle(image, (0,0), (225,73), (245,117,16), -1)

# Rep data
cv2.putText(image, 'REPS', (15,12),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0), 1, cv2.LINE_AA)
cv2.putText(image, str(counter),
        (10,60),
        cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255,255), 2, cv2.LINE_AA)

# Stage data
cv2.putText(image, 'STAGE', (65,12),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,0), 1, cv2.LINE_AA)
cv2.putText(image, stage,
        (60,60),
        cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255,255), 2, cv2.LINE_AA)



# Render detections
mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS,
                mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=2),
                mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                 )

cv2.imshow('Mediapipe Feed', image)
```

```
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Highlights

- **Pose Detection**: Uses Mediapipe for real-time pose detection, which provides accurate landmark tracking for various body parts.
- **Angle Calculation**: Employs trigonometric functions to determine the angle at the elbow, essential for recognizing the curl movement.
- **Dynamic Counting**: Implements a simple yet effective counter for bicep curls based on angular thresholds, providing immediate feedback to the user.

Conclusion

The project successfully demonstrates a real-time bicep curl counter using computer vision. By combining Mediapipe for pose estimation with OpenCV for video handling, the system accurately tracks and counts bicep curls. This approach can be extended to other exercises or adapted for different applications in fitness and motion analysis.