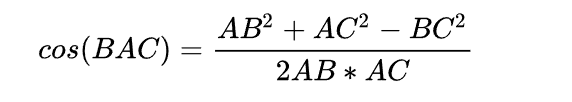
Metoda numarat genoflexiuni:

|  |  |
| --- | --- |
|  |  |

**Algoritm folosit pentru keypoints:**  
-mediapipe

**Algoritmi matematici:**  
-teorema cosinusului: *“Într-un triunghi oarecare pătratul unei laturi este egal cu suma pătratelor celorlalte două laturi minus de două ori [produsul](https://ro.wikipedia.org/wiki/Produs" \o "Produs) lor multiplicat cu cosinusul unghiului dintre ele.” ~https://ro.wikipedia.org/wiki/Teorema\_cosinusului*



**Interpretare unghiuri:**  
 1. cand punctul A se ridica astfel incat unghiul ABC trece de 160 de grade, atunci stage trece din “none” sau “jos” in “sus”. Am adaugat aceasta variabila pentru a incepe genoflexiunile din punctul correct dar si pentru a putea stabili ca genoflexiunea a fost dusa pana la capat.

if angle > 160:  
 stage = "sus"  
if angle < 70 and stage == "sus":  
 stage = "jos"  
 counter += 1

2. cand punctul A, coboara astfel incat unghiul ABC formeaza un unghi mai mic de 70 de grade, atunci variabila stage trece din “sus” in “jos” iar counterul de genoflexiuni creste.

**Vulnerabilitati:**

1. Conteaza unghiul de filmare:

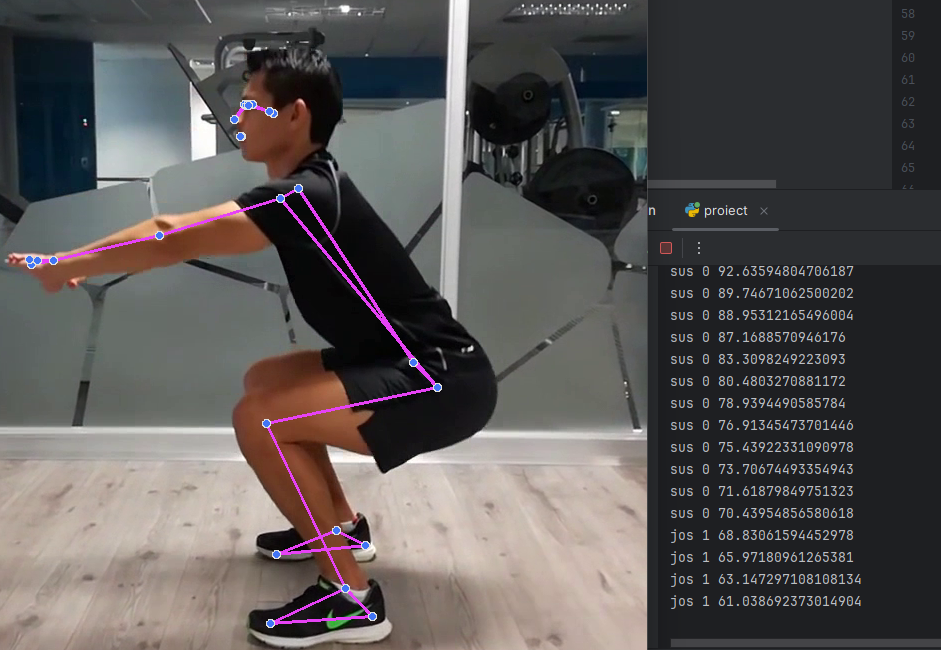
O astfel de filmare cu fata la camera impiedica calcularea corecta a unghiurilor.

Rezolvare:

-impunerea filmarii dintr-o anumita perspectiva  
 -transpunerea intr-o reprezentare 3d a clipului (de documentat!)

**Rezultate:**

1. In momentul flexiei genunghiului <70, variabila stage este modificate in “jos” si counterul pentru genoflexiuni este incrementat

****

ATASEZ CODUL:

import cv2  
import mediapipe as mp  
import numpy as np  
  
mp\_drawing = mp.solutions.drawing\_utils  
mp\_pose = mp.solutions.pose  
  
import numpy as np  
  
  
def calculate\_angle(a, b, c):  
 a = np.array(a)  
 b = np.array(b)  
 c = np.array(c)  
  
 #aici calculez lungimile laturilor triunghiului A B C  
 dist\_ab = np.linalg.norm(a - b)  
 dist\_bc = np.linalg.norm(b - c)  
 dist\_ac = np.linalg.norm(a - c)  
  
 #vf ca AB sau BC sa nu fie 0, ca sa previn impartirea la 0 in legea cosinusului  
 if dist\_ab == 0 or dist\_bc == 0:  
 return 0  
  
 #legea cosinusului si conversie din radiai in grade  
 cos\_angle = (dist\_ab \*\* 2 + dist\_bc \*\* 2 - dist\_ac \*\* 2) / (2 \* dist\_ab \* dist\_bc)  
 #print(cos\_angle)  
 angle = np.degrees(np.arccos(cos\_angle))  
  
 return angle  
  
  
counter = 0  
stage = None  
cap = cv2.VideoCapture('video0.mp4')  
if not cap.isOpened():  
 print("Eroare de la video!")  
 exit()  
  
  
with mp\_pose.Pose(min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as pose:  
 while cap.isOpened():  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  
 image.flags.writeable = False  
  
 results = pose.process(image)  
  
 image.flags.writeable = True  
 image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)  
  
 if results.pose\_landmarks:  
 landmarks = results.pose\_landmarks.landmark  
  
 sold = [landmarks[mp\_pose.PoseLandmark.LEFT\_HIP.value].x, landmarks[mp\_pose.PoseLandmark.LEFT\_HIP.value].y]  
 genunchi = [landmarks[mp\_pose.PoseLandmark.LEFT\_KNEE.value].x, landmarks[mp\_pose.PoseLandmark.LEFT\_KNEE.value].y]  
 glezna = [landmarks[mp\_pose.PoseLandmark.LEFT\_ANKLE.value].x, landmarks[mp\_pose.PoseLandmark.LEFT\_ANKLE.value].y]  
  
 angle = calculate\_angle(sold, genunchi, glezna)  
  
 if angle > 160:  
 stage = "sus"  
 if angle < 70 and stage == "sus":  
 stage = "jos"  
 counter += 1  
 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))  
  
 print(stage, counter, angle)  
 cv2.imshow("Genoflexiuni", image)  
  
 if cv2.waitKey(10) & 0xFF == ord('q'):  
 break  
  
 cap.release()  
 cv2.destroyAllWindows()