Visual computing & image processing

Project Title: Elderly falling detection
Author: Mahmoud M. Shoieb
Description: Developed as part of a university coursework in Artificial Intelligence.
(Full details and code available in this repository.)

Important note: This project is made using Jupiter notebook and not google colab because the project supports life stream from the camera and due to security concerns google colab doesn't support that.

Description:

As our population ages, ensuring the safety and well-being of the elderly becomes increasingly crucial. One significant concern for elderly individuals living independently is the risk of falls, which can lead to serious injuries and complications. To address this concern, lots of technology has been implemented, such as my fall detection software. This software utilizes cutting-edge algorithms to detect falls in real-time, In this article, we explore the importance and benefits of elderly fall detection software in real-life scenarios.

Elderly fall detection software offers rapid response times, ensuring that assistance is provided promptly in the event of a fall. This can be especially critical for individuals living alone who may not be able to seek help themselves.

By continuously monitoring the movements and activities of elderly individuals, fall detection software helps to create a safer living environment. It provides peace of mind to both the elderly and their caregivers, knowing that help will be summoned immediately if a fall occurs.

Prompt intervention following a fall can prevent complications such as prolonged immobilization, dehydration, and secondary injuries. Fall detection software minimizes the time between the occurrence of a fall and the provision of medical assistance, thereby reducing the risk of adverse outcomes.

For many elderly individuals, maintaining independence and autonomy is essential for their quality of life. Fall detection software supports independent living by providing an added layer of security without compromising freedom and mobility.

Elderly fall detection software employs advanced algorithms to accurately detect falls in real-time. This instantaneous detection enables immediate response and intervention, significantly reducing the time it takes to provide assistance.

Unlike traditional monitoring systems that rely on manual checks or wearable devices, fall detection software provides continuous, non-intrusive monitoring. This constant vigilance ensures that falls are detected even during times when the elderly person may not be wearing a monitoring device or anything all is needed is just a camera.

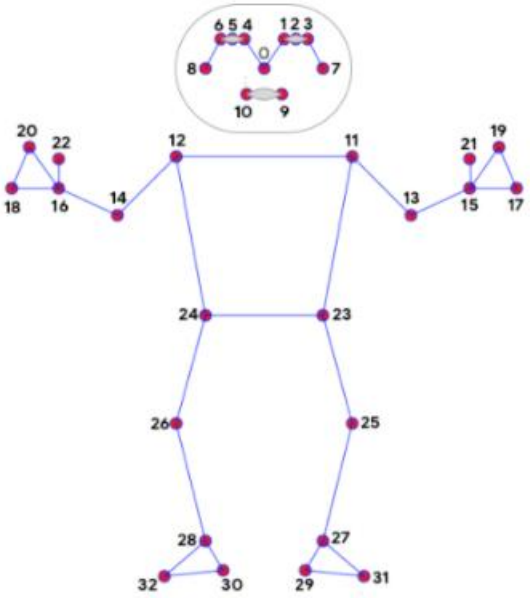
Many fall detection systems incorporate data analytics capabilities, allowing caregivers and healthcare professionals to analyze trends and patterns in the elderly person's movements and behaviors. This data-driven approach can inform personalized care plans and interventions to prevent future falls.

Elderly fall detection software plays a pivotal role in safeguarding the health and well-being of older adults, enabling them to live independently while receiving timely assistance in the event of a fall. With its real-time detection capabilities, and continuous monitoring features, this software offers peace of mind to both elderly individuals and their caregivers. By harnessing the power of technology, we can enhance the safety, autonomy, and quality of life of our aging population, ensuring that they can age gracefully in place.

Algorithms and techniques used:

First I used the matplot which keep track of the human skeleton and this library is essential for determining the center of gravity and keep track of human.

This is a brief for what this library track:



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

Then there's a method for calculating angles using three points in a plane

```
## to calculate the angle formed by three points in a plane
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 int(angle)
```

This method if for geometric calculations and pose estimation tasks

Then after testing the library and getting coordinates from the image

First we extracted all the landmarks for human body parts then we got the needed coordinates and calculated the angle using the user-defined function, then The coordinates for the left foot index and left heel are averaged to determine the point of action for the left foot (Point_of_action_LEFT_X and Point_of_action_LEFT_Y). and so as for the right foot and X,Y are compined into a single array

Then there is three center of gravity detection first for the upper body the second is for the whole body and the third is for the lower body, this can be useful because since the person is moving and this method is using a life stream so calculating the height won't be accurate so we calculate the ratio between two centers of gravity because the scale won't change.

Also visualization implemented to visualize each point.

Determination of a fall technique:

1) Calculate Fall Distance: The code calculates the vertical distance (fall) between a reference point (dot_BODY_X) and another point (Point_of_action_X). This distance likely represents how far the person has moved downwards, suggesting a fall.
2) Determine Falling or Standing: It then determines whether this distance (fall) exceeds a certain threshold (in this case, 50 pixels). If the distance is greater than 50 pixels, it considers the person to be falling (falling = True), otherwise, it considers them to be standing (standing = True).
3) Detect Fall: If the person is detected as falling (falling == True), it checks for specific conditions to confirm a fall. The conditions checked include:

If the person's position is within a specific range (Point_of_action_X < 320 and Point_of_action_X > 100 and Point_of_action_Y > 390 and Point_of_action_Y > y) and they are considered to be standing (standing == True), the code labels it as a fall and updates the stage variable accordingly.

(these numbers are corresponds to a specific area in the frame where a person might typically be located before falling.)

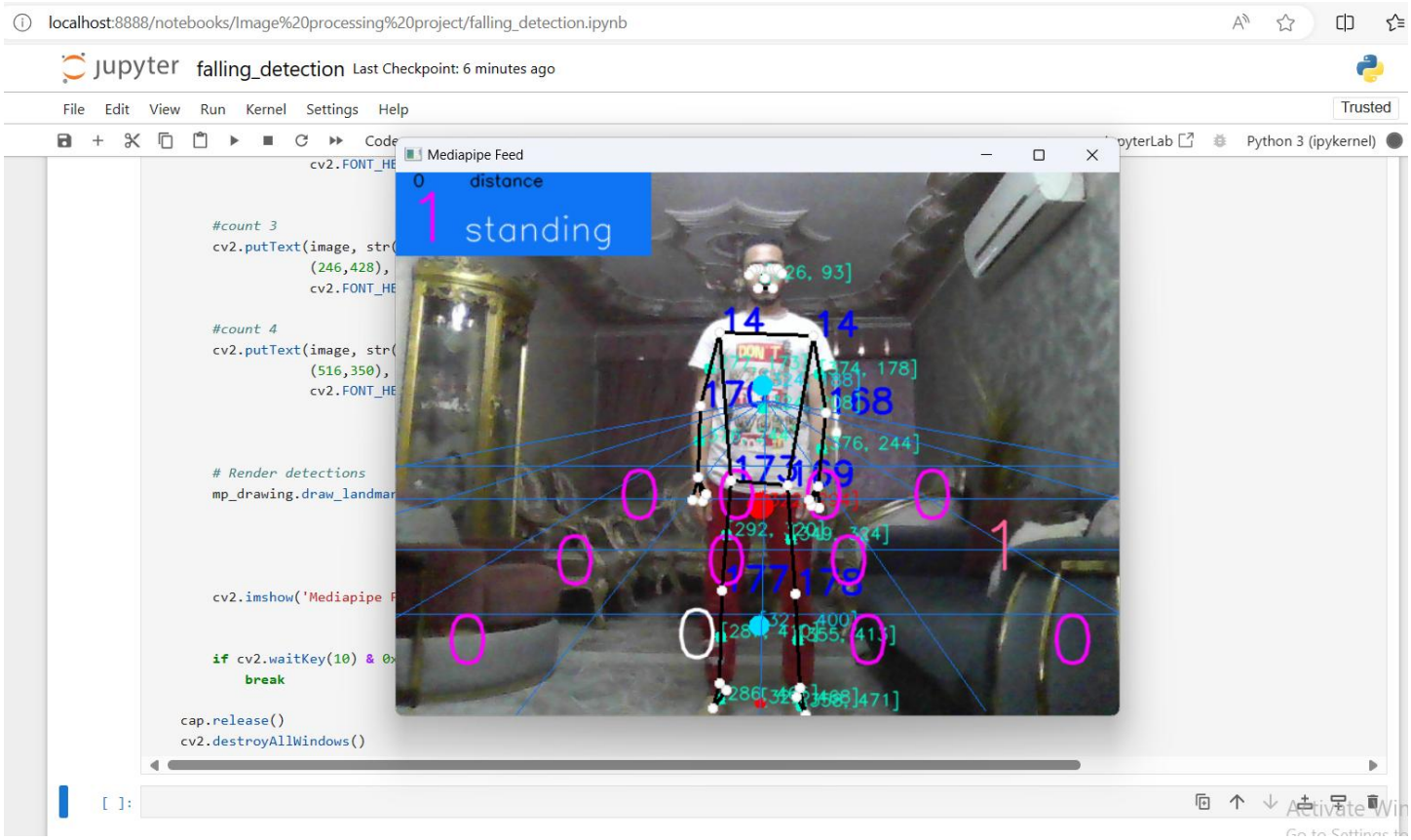
(Point_of_action_Y > 390: This condition checks if the vertical position of the person (Point_of_action_Y) is above 390 pixels. This likely indicates that the person has not yet reached the bottom of the frame, suggesting that they are still in the falling motion rather than standing up.)

If the person's position is within another specific range (Point_of_action_X >= 320 and Point_of_action_Y > 240 and standing), it also labels it as a fall and updates the stage variable.

4) Update Counters: Throughout the code, there are counters (counter_three and counter_four) that are updated based on specific conditions. These counters keep track of the number of falls detected under different scenarios.
5) Visual Feedback: The code provides visual feedback by drawing lines and text on the image (cv2.putText and cv2.line). It also updates the status box with relevant information such as the fall distance, the current stage (falling or standing), and the counters for different fall scenarios.

Screenshots of the output:

In the standing state:



In the falling state: