**MediaPipe Pipeline for Fall Detection**

**1. Introduction**

There are multiple approaches to human fall detection, of those the most popular ones are sensor based (for example, using smart watches with accelerometers) and visual based (for example, using video cameras). This pipeline looks towards a visual based approach to detecting falls in humans by utilising MediaPipe to calculate speeds that body key points move at that could indicate a fall event and body poses in a fall.

**2. Related Works**

This speed aspect of this pipeline is based on “*Vision-based Fall Detection System: Novel Methodology and Comprehensive Experiments*” by Chi-Tam Nguyen et al [1]. It proposed a better method at detecting falls than the optical flow method proposed in “*Deep learning for vision-based fall detection system: Enhanced optical dynamic flow*” by Sagar Chhetri et al [2].

In order to do this, they used MediaPipe pose estimation which can detect one person at a time and extract a skeleton of key points of their body. In videos with multiple people the problem of not detecting a fall may arise since only one person at a time is focused on. To solve the problem of not detecting multiple people, they combined MediaPipe with YOLOv4 (which can detect multiple people) and gave each person detected by YOLOv4 a unique I.D. which MediaPipe then would now be able to extract key points from as it now sees each person individually.

The paper then focuses on 4 key points from the human skeleton extracted were four essential key points in the body, including the head, abdomen (mid-point of the hip), left knee, and right knee. It chose these key points as a result of the research undertaken by Bourke et al [3] which investigated the relationship between a fall and body posture and positioning. The findings of Bourke et al had 100% specificity for non-falls (e.g., videos of someone sitting on a chair) even if the exact activity was not always correctly guessed it was never a false fall alarm (e.g., a video of someone sitting on a toilet confused for sitting on a stool). This suggests that a fall has a unique body pose that cannot be mistaken for other activities.

The optical flow method proposed in [2] used a technique that encodes the temporal data of optical flow videos by the method of rank pooling, which thereby improves the processing time of fall detection and improves the classification accuracy in dynamic lighting condition. It resulted in highly accurate detections however in lower lit conditions the accuracy was seen to fall by as much as 8%.

Nyugen et al’s model looked at combining three speed calculations in order to accurately determine falls.

The three calculations used were velocity and two different types of acceleration; average acceleration which is the mean acceleration of the fall event from start to end. The other acceleration is instantaneous acceleration which is the rate of acceleration from one frame to the next. Each speed calculation is given a threshold which they did not mention in their paper.

Velocity is calculated by the squared difference of the x and y positions at different times. In this research they used the following formulas where:

*V= velocity; s = speed; T = time; x = x co-ordinate; y = y co-ordinate*

Instantaneous acceleration is formulated using the following formula where using the Velocity result the rate of change in speed from frame to frame is obtained:

A math equation with a number of letters

Description automatically generated with medium confidence

A math equation with black text

Description automatically generated with medium confidenceAverage acceleration is a measure of the average rate of change of speed of the key points being tracked from when the fall event is first detected to when it has ended:

The conclusion of this research found that their vision based method performed better than the optical flow research they compared to and the accuracy results below show that even in low lit conditions it performed better:

A table with numbers and text

Description automatically generated

**3. Method**

**3.1 Version 1**

The pipeline I initially implemented varied from the research undertaken in [1] as due to the computationally intensive nature of their approach I explored a similar technique which could potentially hold similarly accurate results for fall detection.

The key component I left out was the use of YOLOv4 due to the amount of processing required to run it without the presence of a graphics processing unit (GPU), so the pipeline implemented focuses solely on being able to detect one person’s fall.

Using MediaPipe’s library I develop a class **PoseDetector**.

This class has multiple functions that I call to perform actions needed to detect the fall such as:

* ***findPose()*** which uses MediaPipe to find key points of a human in the frame and using these key points found, it will return them as dots on the human body
* ***getPosition()*** which gets the x co-ordinate and y co-ordinate of the key points of the human body parts that can be seen and stores it in array so they can be tracked.

Using this **PoseDetector** class now I call it onto a video input. It will then simultaneously show me the visible human key points on the person in frame and keep a record of the positions of the key points by appending them into a list as the video goes on.

A person standing in a room with a drawing of a human body

Description automatically generated

While coding I came across an issue where videos of greater resolutions (width and height sizes) were more sensitive to changes in acceleration since more pixels were being used. To minimise this variation, I gave all videos a standard resolution of 1280-pixel width and 720-pixel height. This was done using ***cv2.resize()*** function from the OpenCV library. Therefore, the threshold values I used would not need to be altered for different videos.

The key point targeted was the hip. This is because research of Gait agrees of the hip as one of the key body parts when analysing gait. Gait is a study of the movement of the human body, including aspects such as stride length, strides per minute and time taken to complete a stride. Since this research shows that the hip is an important part of human movement then a fall event would show a big change in the movement of the hip.

The velocity calculation used was change in hip x co-ordinate and y co-ordinate every 10 frames. Thus, frame 1 compared to frame 10, frame 11 compared to frame 20 and so on throughout the video.

The average acceleration calculation used was the change in speed in the same time intervals. For example, the rate of change in speed from frame 1 to frame 10.

These calculations were combined to determine if a movement that occurred was potentially a fall. Trial and error was used to tune the threshold value to indicate a fall event.

**3.2 Version 2**

Having trialled a simplistic version of the proposed pipeline from [1], there were successes and areas for improvement.

This new version of the pipeline looked at five key points; nose, left hip, right hip, left knee and right knee. The reason behind these key points being selected for the pipeline is the same reasoning used in the related works [1] and [3] in respect to body part positions in fall events.

Each key part x and y position was saved in its own list at every frame of the input video. Then three rules are combined to determine a fall.

**Rule 1:** Velocity every ten-frame interval. This is a broader look at the movements of the human as since the video runs at around 20 frames per second, this looks at every half second of movement which is a realistic timeframe for a fall event. For example, if the body points have moved an average of over 400 pixels in half a second this is a big jump and indicates a fall.

**Rule 2:** Acceleration every five-frame interval. This looks closer at the different velocities in the build up to the fall. If at frame 1 the velocity is at 20 pixels a frame and at frame 5 the velocity is at 70 pixels a frame and then at frame 10 the velocity is at 170 pixels frame this shows that the person is picking speed at an increasing rate. This is common in a fall before hitting the ground so it can be an additional indication of a fall event.

**4. Results and Findings**

**4.1 Version: Velocity (1 Rule)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Accuracy** | **Sensitivity** | **Specificity** | **Precision** |
| 91% | 6% | 99% | 33% |

|  |  |  |
| --- | --- | --- |
|  | **Predicted No Fall** | **Predicted Fall** |
| **Actual No Fall** | **14311** | **173** |
| **Actual Fall** | **1313** | **87** |

**Accuracy** measures the overall correctness of the fall detection system in detecting fall frames and detecting non-fall frames.

This was calculated by:

**Sensitivity** measures the ability of the fall detection system to correctly identify frames of falls.

This was calculated by:

**Specificity** measures the ability of the fall detection system to correctly identify frames where no fall has happened.

This was calculated by:

**Precision** measures the accuracy of positive predictions made by the fall detection system (how many of the fall alarms are really falls).

This was calculated by:

True Positives – Predicted Fall that was Actual Fall

True Negatives – Predicted No Fall that was Actual No Fall

False Positives – Predicted Fall that was Actual No Fall

False Negatives – Predicted No Fall that was Actual Fall

The model I used was successful in detecting true falls for three fall videos tested. There were, however, false alarms which occurred. The reason was that for some frames the MediaPipe algorithm would not detect the person and when it had found the person again the key points had skipped some frames so the change in velocity was higher. This is because the formula is distance divided by frame, so a greater distance travelled per frame results in a greater value for velocity.

In total 260 fall alarms were raised, 87 of these were True Falls. Since there are 1400 frames total of fall events that means only 6% of all fall frames were resulted in a detection however 96% of all fall events (not frames but fall actions) had at least one fall alarm. 1% of all non-fall frames had a false fall alarm. This method is not highly sensitive so very few fall alarms are triggered which could mean slow falls are not detected e.g., a fall where the person attempts to break their fall by holding onto objects/furniture.

The image below shows one of the videos where the background affected the detection of the person. This is due to a combination of the person facing away from the camera reducing the ability for MediaPipe to detect a human and the position of the cushions being perceived as a person lying down sideways.

A person standing in a room

Description automatically generated

**4.2 Version 2: Velocity and Acceleration (2 Rules)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Accuracy** | **Sensitivity** | **Specificity** | **Precision** |
| 91% | 23% | 97% | 41% |

|  |  |  |
| --- | --- | --- |
|  | **Predicted No Fall** | **Predicted Fall** |
| **Actual No Fall** | **14028** | **456** |
| **Actual Fall** | **1078** | **322** |

Using a labelled dataset for the “*Le2i Fall Detection Dataset*” I was able to compare which frames my method detects a fall compared to the labelled frames of the dataset which state when the fall even begins and ends. Of the small sample group of fifty videos I explored it had an accuracy of 96% of detecting a fall event. However, only 36% of fall videos had zero false alarms for a fall. This occurred mainly at the start of the when the MediaPipe key points first attach to the person so there was a jump in the speed of the movement of key points.

Additionally, to the method being able to detect a fall event, I examined how many frames indicated a fall out of the entire fall sequence. I found that 23% of the video frames labelled as “fall” were detected by the algorithm. These tended to be near the start of the fall until the peak speed was reached.

A person lying on a mattress in a room

Description automatically generatedBelow is an example of the output frame of a correctly detected fall:

**4.3 Version 3: Velocity and Acceleration with Background Subtraction (2 Rules)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Accuracy** | **Sensitivity** | **Specificity** | **Precision** |
| 90% | 24% | 97% | 41% |

|  |  |  |
| --- | --- | --- |
|  | **Predicted No Fall** | **Predicted Fall** |
| **Actual No Fall** | **13992** | **492** |
| **Actual Fall** | **1059** | **341** |

To minimise the problems arising from lack of human detection by the MediaPipe model, background subtraction was used to preprocess the frames. This was achieved through the mixture of Gaussian model 2 background subtraction algorithm (MOG2) which works by calculating the appropriate number of gaussian distributions for each pixel (the original MOG algorithm had a fixed k number of gaussian distributions for each pixel predetermined so was less adaptable). These distributions are compared frame by frame to determine what is not moving and can be considered background pixels and removed.

As a result of using MOG2, MediaPipe was able to detect a person in every video and in some cases was able to detect the person earlier on in the video. This lowered the amount of videos with no falls detected however there was one video which even though the person was detected throughout, no fall was detected which may be to do with the fall threshold speed values.

Below is an image example of a detected fall using MOG2 and the two speed rules:

A person sitting on a chair

Description automatically generated

**4.4 Version 4: Velocity, Acceleration and Pose (4 Rules)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Accuracy** | **Sensitivity** | **Specificity** | **Precision** |
| 92% | 9% | 99.96% | 95% |

|  |  |  |
| --- | --- | --- |
|  | **Predicted No Fall** | **Predicted Fall** |
| **Actual No Fall** | **14478** | **6** |
| **Actual Fall** | **1281** | **119** |

Speed is not the only indication of a fall. Research suggests that there are poses and body shape features that occur during a fall. Two additional rules were added to the Version 2 model without applying any preprocessing to compare how considering both speed and body poses would affect the performance of model.

The first rule added was that the height of the person would be less than half their max height (their height when fully stood straight) in event of a fall.

The second rule is that the width of the person would become greater than their height. This is due to the body crumpling up as they fall.

The results of combining speed and pose resulting in numeral new results.

Firstly, falls were no longer detected at their early stages and were now detected more closely toward the final stage of the fall.

Secondly, the amount of false fall alarms decreased heavily since the requirements were so specific to a fall, no other activities could easily replicate a fall and trigger a false fall alarm.

Thirdly, fewer frames in total were flagged as fall frames so the sensitivity of the model dropped more than double. This resulted in more falls not being detected however, falling from 96% of videos detecting a fall in previous models to 72% of videos detecting a fall. This may be solved my changing the threshold values.

**4.5 Comparisons**

The initial simplistic algorithm differs from the refined second version mainly by the number of detections. As Version 1 looks at speed per ten frames and does not consider acceleration within the ten-frame period, as Version 2 does, it was able to detect much fewer fall frames however it still had the same accuracy in fall detection of 96% due to it being able to detect the peak of the fall. However, we the dataset tested on was only 50 videos it is likely that on a greater volume of videos it would fall in accuracy.

Accuracy remained the same between models as the main issue affecting accuracy was lack of data preprocessing and not the actual rules for determining a fall. In Version 2 Sensitivity is almost four times greater as the addition of a new rule helps to detect more features of the fall. Specificity however, fell by 2% due to more false alarms occurring since the newer version looks at the speed of the person in more detail resulting in more false alarms from movements of MediaPipe’s key-points. Precision overall improved as well even though more false alarms occurred it was outweighed by the true falls detected.

**4.6 Results Comparison Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Dataset** | **Accuracy** | **Sensitivity** | **Specificity** | **Precision** |
| Nyugen et al’s “Vision-based Fall Detection System” | Fall Detection Dataset (FDD) | 99.99% | N/A | N/A | N/A |
| [Ours] Velocity: 1 Rule | Le2i Fall Detection Dataset | 91% | 6% | 99% | 33% |
| [Ours] Velocity and Acceleration: 2 Rules | Le2i Fall Detection Dataset | 91% | 23% | 97% | 41% |
| [Ours] Velocity and Acceleration with Background Subtraction: 2 Rules | Le2i Fall Detection Dataset | 90% | 24% | 97% | 41% |
| [Ours] Velocity, Acceleration and Pose: 4 Rules | Le2i Fall Detection Dataset | 92% | 9% | 99.96% | 95% |

**4.7 Results Comparison Graphs**

Graph

The Graph 1 above shows the comparison of the models. accuracy between all models was similar with the highest accuracy belonging to the model with Velocity, Acceleration and Pose combined at 92%. Models with Acceleration and Velocity only had the highest sensitivity. This may be because acceleration looks at the rate of change in velocity so was good for working with labelled frames for fall. However, these two models performed slightly worse for specificity which indicates it may be more prone to false alarms. However, when it came to precision, the model with Velocity, Acceleration and Pose combined performed more than twice as high as any other model. This shows that the rules used for this model work really well for classifying a fall and potentially with different threshold values it may perform better overall.

**5. Improvements**

**5.1 Version 1**

Following the outcome of my model areas of improvement were considered. The approach of comparing key points in 10 frame intervals could be altered as I compared frame 1 to frame 10. This could potentially leave a small chance of a fall not being detected if the fall event overlaps the interval. One way to overcome this would be adjust the threshold to account for this possibility. Another solution would be to compare frames in a continuous way, for example, frame 1 would still compare to frame 10 but now the next comparison would be frame 2 and frame 11. As there are more constant observations for speed this way it would be less possible for a fall not to be detected.

Another problem that arose was false fall alarms when the MediaPipe failed to detect a key point momentarily. This means for that frame the key point will have co-ordinates of (0,0) and when it is detected again it then jumps up back giving a big difference in distance being added to the speed calculation. To minimise this false alarm occurrence more conditions could be applied to the code to ignore these zero values. For example, the code could be written to ignore frames when the key point has disappeared and to have a maximum change in distance per frame value allowed.

**5.2 Version 2**

Having improved the detection of fall events the use of more body parts and more rules showed a great improvement to the algorithm. To further improve this pipeline, one approach is to preprocess the data for example, background subtraction or blurring could be used to solve the issue of the MediaPipe key points sometimes not focusing on the person in the video. Another way to further improve this pipeline is to combine it with another fall detection method, e.g., the angle of key points to each other in a fall event. This could help detect slow falls which the speed algorithm might not catch.

**5.3 Data Preprocessing**

Having evaluated the outcome of the research undertaken, various data preprocessing technique were implemented. The first preprocessing technique used was MOG2 (Mixture of Gaussians).

This works by obtaining pixels intensity values which are gained from Gaussian distributions (also known as Normal distributions) which will be used to determine if that pixel is more likely to belong to background or foreground. As each new frame is processed the pixel intensity value at each co-ordinate of the frame is updated and judged as background or foreground based on how it has changed. MOG2 has its own predefined thresholds for pixel intensity values for background and foreground which it is compared against.

With the OpenCV function *createBackgroundSubtractorMOG2()* it uses the MOG2 preprocessing technique and allows the classification of pixels into background or foreground. Once this classification has been done all the pixels which were classified as foreground were retained using the *bitwise\_and()* function. This function in essence collects the pixels within the determined foreground area and sets the rest of the pixels in the background to black as shown in the image below:

A person on a snowboard

Description automatically generated

The use of MOG2 reduced the risk of the MediaPipe confusing another object as a person which increased the accuracy in videos which previously struggled due to the background. However due to some pixels sometimes not being retained (e.g., the edges of their body/clothes) the ability of MediaPipe’s model to detect the person may decrease.

The other preprocess technique explored was utilised MOG2 as well but instead of following and retaining the moving pixels in the frame, this technique placed a bounding box around all areas where there were pixel intensity values that met the threshold and everything outside the bounding boxes are not retained and left black. This approach is to minimise any reduction in ability of MediaPipe of detecting the person while simultaneously removing background noise. Below is an image example of this technique:

A person kneeling on a bed

Description automatically generated

As you can see this technique does not have the issue of losing some pixels which may not be reaching the MOG2 predetermined thresholds so more of the human is shown. However, it does allow more of the background into the image which could allow more risk for background disturbance as a trade-off.

The third technique used was a combination of Canny Edge detection and MOG2. The pixels that would be retained were determined by the canny edges. Canny Edge was very successful at not picking up background noise, however as it was mainly the pixels that were found near the edges a lot of the pixels of the person were black. As MediaPipe is trained on human images from the COCO dataset, having these black pixels may lower the ability of MediaPipe to detect a person’s key-points.

Below is an example of Canny Edge combined with MOG2:

A person sitting on a chair

Description automatically generated

MediaPipe was still able to find person but for some frames it struggled to find body parts that the other two techniques did not. To improve this maybe use of other edge detection techniques may provide better results.

**References**

[1] - Nguyen, Chi-Tam & Phan, Thanh-Danh & Duong, Minh-Thien & Nguyen, Van-Binh & Pham, Huynh-The & Le, My-Ha. (2023). Vision-based Fall Detection System: Novel Methodology and Comprehensive Experiments. <https://www.researchgate.net/publication/373629478_Vision-based_Fall_Detection_System_Novel_Methodology_and_Comprehensive_Experiments>

[2] - S. Chhetri, A. Alsadoon, T. Al-Dala’in, P. Prasad, T. A. Rashid,and A. Maag, “Deep learning for vision-based fall detection system : Enhanced optical dynamic flow,” Comput. Intell., vol. 37, no. 1, pp. 578–595, 2021. <https://arxiv.org/ftp/arxiv/papers/2104/2104.05744.pdf>

[3] - Bourke, Alan & O'Brien, J.V. & ÓLaighin, Gearóid. (2006). Evaluation of A threshold-based tri-axial accelerometer. J Gait and Posture. 26. 194-199. <https://www.researchgate.net/publication/291990647_Evaluation_of_A_threshold-based_tri-axial_accelerometer>