****

Visually-Based Fall Detection System with IoT Alarm

ZIWEI XU

**3rd Year Project Final Report**

Department of Electronic &  
Electrical Engineering

UCL

Supervisor: Prof. Nathan Gomes

29 March 2023

I have read and understood UCL’s and the Department’s statements and guidelines concerning plagiarism.

I declare that all material described in this report is my own work except where explicitly and individually indicated in the text. This includes ideas described in the text, figures and computer programs.

This report contains 42 pages (excluding this page and the appendices) and 10980 words.

|  |  |
| --- | --- |
| Signed: \_\_\_\_\_\_\_ZIWEI XU\_\_\_\_\_\_\_\_\_ | Date: \_\_\_\_13/4/2023\_\_\_\_\_\_ |

**Visually-Based Fall and Falling Detection**

**with IoT Alarm**

ZIWEI XU

**Abstract**

**This project aims to create a visually-based body fall detector for the elderly, using an RGB camera. The main target audience for this project is elderly people who live alone. Based on numerous research studies, it has been found that elderly individuals who live alone may have difficulty getting up on their own after a fall, and delayed notification of a fall may lead to life-threatening situations that could have been prevented. This project is divided into two parts: a Fall detection system and an IoT alarm notification system. The fall detection system, which is based on Mediapipe pose and OpenCV, can detect in real-time whether a person captured by the camera has fallen. OpenCV is used to extract frames from the real-time video, which are then processed by Mediapipe Pose to extract the person in the frame and define 33 coordinate points on their joints, which are then used to determine whether a fall has occurred. Once the system detects a fall, the PushDeer IoT alarm will quickly transmit the message to a specified device, such as a family member's mobile phone, ensuring that the fallen elderly person receives immediate assistance. Through experiments conducted with three different datasets, the fall detection system achieved an accuracy rate of 97.67% in monitoring environments with clean backgrounds and clear images of people. However, the accuracy of the system may be significantly affected by complex backgrounds, such as in datasets where the accuracy rate can drop to as low as 86.67%. This project can be applied not only in homes, but also in any situation where the safety of the elderly needs to be monitored.**

**Contents**

Abstract3

1 Introduction5

1.1 Project Motivation / Background5

1.2 Literature review of Internet of Things7

1.3 Literature review of Fall detection system7

1.4 Theoretical Background of tools8

1.5 The Target Performance Criteria11

1.5.1 System Evaluating Criteria11

1.5.2 System Target Performance12

1.6 The Report Overview13

2 System and Algorithms13

2.1 The Fall and Falling System14

2.1.1 Video Pre-processing demonstration and explanations15

2.1.2 Video Pre-processing code demonstration and explanations16

2.1.3 Human Body landmarks information processing demonstration and explanations18

2.1.4 Human Body landmarks information processing code demonstration and explanations19

2.1.5 Fall Detection System demonstration and explanations23

2.1.6 Fall Detection System code demonstration and explanations24

2.1.7 Falling Detection System demonstration and explanations26

2.1.8 Falling Detection System code demonstration and explanations27

2.1.9 Whole system with repeat system demonstration and explanations30

2.2 The IoT Alarm33

2.2.1 IoT Alarm Code demonstration and explanations34

3 Datasets & Results35

3.1 Ideal dataset criteria35

3.2 Own Dataset36

3.3 50 ways to fall37

3.4 UR Fall detection dataset38

3.5 Results Evaluation39

4 Conclusion and Future work41

4.1 Project Summary41

3.5 Project Conclusion41

3.5 Project Future Work42

5 Reference43

6 Appendix45

1. **Introduction**

The ageing of the population poses an inescapable and grave challenge for contemporary society. Furthermore, elderly individuals often experience physical debilitation, which can result in sudden falls due to diminished faculties, impaired vision, or abrupt cardiac events. Given the escalating numbers of elderly people and their frailty, a reliable fall detection and notification system has become increasingly imperative. The goal of this project is to avert fatalities resulting from falls that go unnoticed and unattended. This initial section provides an overview of the project's motivation, background, fundamental components, the tools employed, their theoretical foundations, and the envisioned final product.

* 1. **Project Motivation / Background**

Over the past few decades, the world has experienced a significant increase in life expectancy due to advancements in healthcare and medical technology. This has resulted in a growing population of the elderly, with the number of people over the age of 65 expected to double by 2050. As reported by the World Health Organization (WHO), the proportion of individuals aged 60 and above is predicted to rise from 12% to 22% between 2015 and 2050.[1] This indicates that by 2050, one-fifth of the populace will have reached sixty years of age or older.

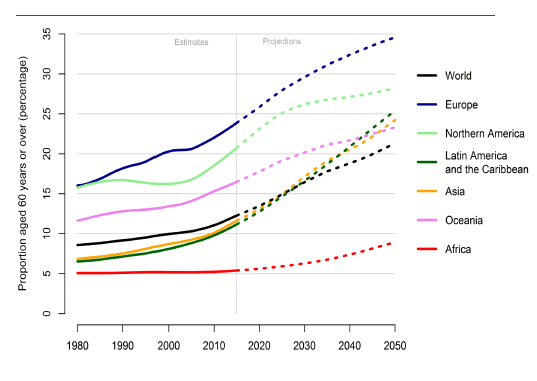


Figure : Percentage of population aged 60 years or more according to region from 1980 to 2050 [2]

Based on the information shown on the Figure1, it indicates the seriousness of the problem of aging population. However, the aging population also brings the consequences like increasing frailty and visual problems. According to the World Health Organization again, for the people over 65 years old, 64% of falls can results in fractures and 32% of hospital admission.[3]

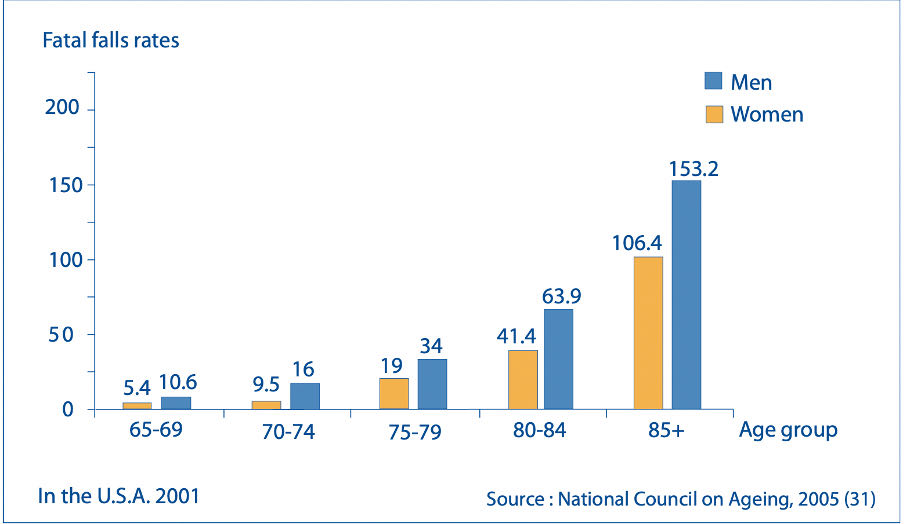


Figure : Fatal falls rate of Men and Women in the U.S.A [4]

From Figure 2, the data shows that the fatal falls rates increase exponentially by the age. The individuals aged 65 or older causes over 36,000 deaths in 2020. It has become the leading cause of injury death among the old people.[5]

After a fall, elderly individuals often cannot get up on their own without assistance. If the fall results in serious injury, such as a fracture, immediate notification is crucial for those who are unable to get up on their own. Prolonged lying on the floor can lead to hypothermia or pneumonia, which can exacerbate the elderly person's injuries. In addition, waiting helplessly for several hours can be extremely distressing for the elderly and may have negative psychological effects.[6]

Every life has extraordinary meaning, and it is regrettable for one's life to end in such a manner. The purpose of this project is to address the issue of delayed notification in falls among the elderly, in order to facilitate timely medical treatment for all elderly individuals in society who experience falls.

* 1. **Literature review of Internet of Things**

In 1985, the term “Internet of things” first appeared in a speech by Peter T. Lewis. According to his speech, he pointed out that IoT is the integration of people, processes and technology with connectable devices and sensors for remote monitoring, manipulation and evaluation of such devices.[7]

Nowadays, based on the definition from IBM, The Internet of Things (IoT) can be defined as a concept of connecting any device (so long as it has an on/off switch) to the Internet and to other connected devices.[8] It is a giant network of connected things and people. This vast network comprising interconnected devices and individuals, enabling the collection and sharing of data on usage patterns and surrounding environments. [8]

Internet of things provides a perfect solution to the problem of fall detection in elderly individuals. As described in Section 1.1, the Project Background, this project aims to address the need for real-time notification to caregivers and emergency services in the event of falls in elderly individuals. Leveraging the IoT's networking capabilities, notifications can be sent within a very short time, thereby preventing fall-related fatalities resulting from delayed notification.

* 1. **Literature review of fall detection systems**

Currently, mainstream fall detection systems can be roughly divided into two categories: wearable and visual-based systems. Wearable fall detection was the earliest research direction in fall detection. In the 1990s, Lord and Colvin embedded micro-accelerometers and microcomputer chips in badges to detect whether people had fallen.[9] Since then, most research has generally used accelerometers as sensors. For example, in Bourke's study in 2007, they investigated the most suitable location to place the sensor on the body, achieving 100% specificity under specific conditions.[10]

Wearable fall detection is based on the physiological changes that occur in the body during a fall. Various micro-sensors such as accelerometers, pressure sensors, gyroscopes, and electrocardiograms are used to monitor the abnormal movements of the subject during a fall. The fall detection criteria are then created by studying the correlations between the data. Due to the convenience and low cost of wearable devices, this method has been the main research object in fall detection from the beginning until now.[11]

However, compared to visual-based fall detection systems, wearable fall detection remains invasive for the elderly. The battery life of wearable devices is also a problem for the elderly. Elderly individuals' forgetfulness can lead to forgetting to charge the device or forgetting to wear it. In contrast, visually-based fall detection, can solve these problems.

Visual-based fall detection research emerged at a later stage. In 2011, Rougier et al. proposed a shape-matching technique that tracks a person's silhouette through a video sequence and analyses their behaviour based on the silhouette shape.[12] In 2012, Charfi demonstrated the feasibility of using RGB surveillance cameras to detect falls.[13] Subsequently, with the release of the Kinect device by Microsoft, a large number of studies emerged based on the built-in RGB-D depth camera of the Kinect.

2011, Rougier et al. were the first to utilize depth cameras in fall detection research. The depth information was used to extract two features of the human body, centre of mass height and motion speed, for fall detection. They employed a threshold-based fall detection algorithm and achieved a success rate of 98.7%.[14]

In recent years, visual-based fall detection systems based on RGB cameras have made significant progress. For example, Z.Huang proposes a video-based fall detection system by using a convolutional neural network to extract features from the images and estimate the human pose.[15] Z.Dong proposed a framework that extract skeletal key points from video data by OpenPose and input the key points data to LSTM model to detect falls. [16] There are many papers that extract human skeleton point information from videos and use machine learning classifiers to identify common features of falls.[17-20] These papers provide direction and inspiration for this project. However, these studies rarely incorporate an IoT alarm, and most of them rely on machine learning classifiers to develop fall detection algorithms. Their advantage lies in achieving high accuracy rates with a sufficiently large amount of data. However, the large amount of data required makes it challenging for individual researchers to obtain, and it is also difficult for machines to handle the processing of such massive amounts of data.

This project aims to develop a low-cost solution to fall detection by minimizing the data requirements and reducing the computational load while maintaining a reasonable level of accuracy.

* 1. **Theoretical Background of Tools**

In this section, tools used in the project and the theoretical background of these tools will be introduced. The project requires the use of Mediapipe and OpenCV for the fall detection system, and is based on PushDeer for the IoT alarm.

**1) The Fall and falling system**

**Mediapipe：**

Mediapipe is an open-source, cross-platform framework for building multimodal machine learning applications.[26] It provides a set of pre-built and customizable building blocks for processing audio, video data, as well as for creating machine learning pipelines. In this project, Mediapipe Pose, which is one of the modules of Mediapipe. Mediapipe pose is a real-time pose estimation model that can detect human poses from video or camera input.[27] Mediapipe Pose is built on top of two key technologies: BlazePose and MobileNetv2.

BlazePose is a real-time pose estimation algorithm developed by Google. It is capable of detecting human poses and compute x, y, z coordinates and visibility of 33 different skeletal key points.[26] As shown in Figure 3, BlazePose marks 33 points on the human body in the photo, including the nose, eyes, shoulders, elbows, hips, and feet. Almost all key points of the human body are defined by a coordinate. The x, y, z values and visibility values represent the position of the point and whether it is clearly visible in the photo. These values are normalized based on the width and height of the video, and are in the range of 0 to 1. Specifically, the (x, y) values correspond to the 2D pixel coordinates of the keypoint in the image plane, while the z value represents the estimated depth of the keypoint in 3D space, relative to the camera. The camera coordinate system is a 3D coordinate system cantered at the camera's optical centre, with the x-axis pointing to the right, the y-axis pointing down, and the z-axis pointing along the camera's optical axis, away from the camera. [26]

图表, 雷达图

描述已自动生成

Figure 3: 33 Pose landmarks[21]

图片包含 室内, 桌子, 笔记本, 椅子

描述已自动生成

Figure : Example of landmarks extraction by Mediapipe Pose [22]

Convolutional Neural Network (CNN) which is a popular algorithm for deep learning. In this project, the CNN model architecture is MobileNetv2-like block.[28] It contains 3 layers which are convolution layer, pooling layer and FC layer.[27] It enables efficient inference on low-power devices by reducing the number of parameters and computations required.

BlazePose provides the basic architecture and training data for the model, while MobileNetv2 provides the lightweight and efficient architecture that enables real-time inference on low power device. The CNN architecture of Mediapipe Pose has been optimized for pose estimation, with layers that are specifically designed to extract features from the input image that are relevant to human poses.

**OpenCV：**

OpenCV is a popular computer vision library used for real-time image processing and computer vision applications. [23]

In this project, OpenCV was used to read video streams from camera or prepared video files, which is essential for obtaining the input video feed for the fall detection system. Secondly, OpenCV was also used to provides functions to perform various pre-processing tasks on the video frames, such as resizing, cropping, rotating, and converting the colour space. This is for reducing the noise and improving the overall quality of the input data.

**2) The IoT Alarm**

**PushDeer：**

PushDeer is a Wi-Fi-based notification push software. It is designed to deliver push notification to users on a variety of devices, such as smartphones or desktop computers.[24] The Push notifications are the messages that are sent from a server or application to a user’s device.

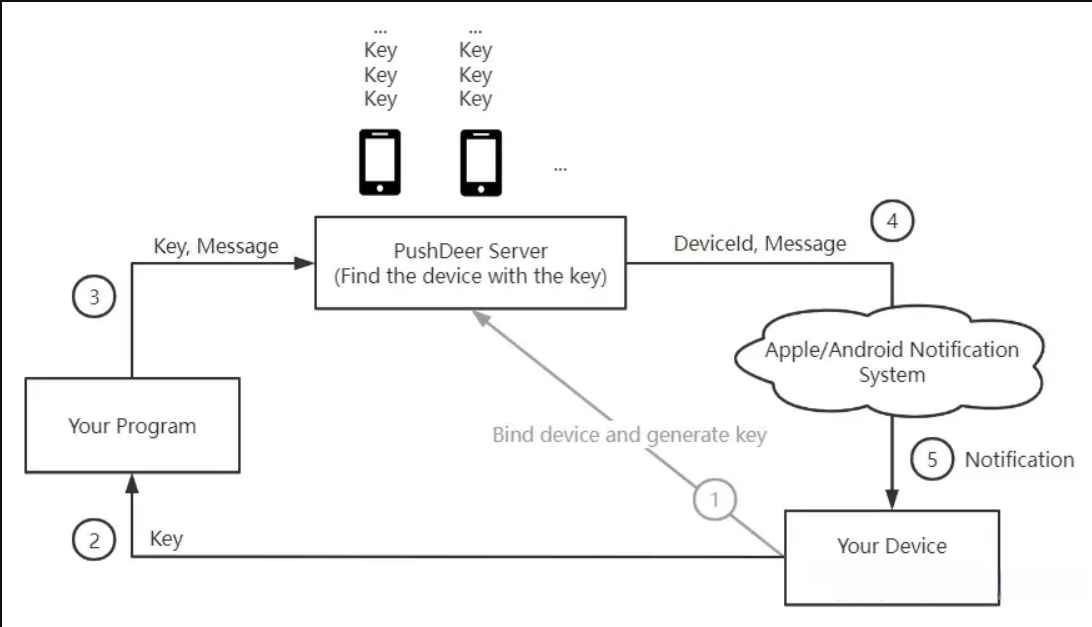


Figure : PushDeer software schematic [25]

Figure 5 is the PushDeer software schematic. User need to install a mobile app on a device which is capable of receiving push notifications. The app then register with the PushDeer service, generate a key which allows it to receive notifications from the server. When a notification needs to be sent to the user, the server sends the notification to the push notification service, which in turn delivers it to the user's device. The notification is typically displayed on the user's lock screen and in their notification centre, and may include text, images, or other interactive elements.

* 1. **The Target Performance Criteria**

This chapter consists of two parts: a background theoretical introduction to the system evaluation Criteria, as well as the target performance for the system.

* + 1. **System Evaluating Criteria**

To evaluate the accuracy of the algorithm, it is necessary to test four types of data: the number of successful non-fall detections, failed non-fall detections, successful fall detections, and failed fall detections.

**Confusion matrix:**

A confusion matrix is a table layout that allows visualization of the performance of an algorithm.[30] It uses four kinds of results along with positive and negative classification to show the difference between the actual and predicted values.

表格

描述已自动生成

Figure 6:Confusion matrix Flowchart [31]

In this project, the input data contains both fallen and non-fallen poses, so the algorithm performance evaluation need to be taken for both of them separately.

In Figure 6,

True positive (TP) for Successful fall number

True negative (TN) for Successful non-fall number

False positive (FP) for failed fall number

False negative (FN) for failed non-fall number

**Accuracy measurement method:**

Accuracy of an algorithm is obtained by dividing the successful fall & non-fall numbers by the sum of all the tests. It gives the proportion of correct predictions.

**Precision:**

Precision of an algorithm is obtained by dividing the correct prediction by the total prediction number for fall and non-fall test separately.

**Sensitivity:**

Sensitivity which is also called true positive rate, recall. It is obtained by dividing the true positive number by positive number.

**F1 Score:**

The F1 score is calculated as the harmonic mean of precision and recall, where precision is the ratio of true positive predictions to the total number of positive predictions, and recall is the ratio of true positive predictions to the total number of actual positive instances. The F1 score ranges from 0 to 1, with higher values indicating better performance.

* + 1. **System Target Performance**

The anticipated performance of a fall detection system needs to be comprehensive and consider multiple aspects. A satisfactory fall detection system requires high sensitivity, meaning the system can detect as many falls as possible while minimizing the false-negative rate. Minimizing the false negative rate means the system can avoid mistaking other non-fall behaviours as falls. Additionally, the system needs to have high accuracy and high F1 score. As mentioned in the previous chapter, high accuracy means the system can accurately detect various types of falls and distinguish between falls and non-fall movements. The F1 score combines the system's precision and sensitivity, and a high F1 score indicates higher performance of the system.

Overall, a good fall detection system needs to balance sensitivity and accuracy, minimizing false negatives while avoiding false positives. The system's performance should be evaluated using multiple metrics, including sensitivity, specificity, accuracy, and F1 score.

* 1. **The Report Overview**

This report discusses a fall detection and alert notification scheme for the elderly. By using Mediapipe pose to extract 3D skeletal coordinates of the human body, it is possible to determine whether the monitored individual has fallen based on the relative position between their body coordinates. In the event of a fall, the IoT's networking capabilities are used to promptly transmit the fall information to a designated device, allowing the elderly person to receive timely medical treatment. This project aims to achieve fall detection and notification using a low-data and low-cost method while maintaining a certain level of accuracy, based on an RGB camera. If this scheme were to be applied, such as by connecting to monitoring cameras within a city or household, the mortality rate caused by falls in society would be greatly reduced.

1. **Systems and Algorithms**

In this chapter, a detailed description of two different systems, the Fall and Falling Detection System and the IoT Alarm, will be presented. Firstly, the overall flowchart of each system, along with the corresponding code for environment preparation, will be explained to provide a general understanding of their functionalities. Some sections will be briefly summarized, such as the Fall Detection Algorithm, to provide an overview of the system's components. Once the system's overall flowchart is established, detailed explanations of the Fall and Falling Detection Algorithms will be provided, along with their corresponding flowcharts and code.

The Fall and Falling detection system consists of five parts: video information pre-processing, human body keypoint information extraction, fall detection algorithm, falling detection algorithm, and system loop condition. Fall detection is mainly achieved by combining the fall detection algorithm and the falling detection algorithm. The falling detection algorithm serves as an auxiliary to fall detection, aiming to eliminate some movements that are easily confused with falls, such as squatting, by determining the falling speed.

In contrast, the IoT alarm only has one component, which explains how to achieve real-time alarm notifications within the system by using the PushDeer message push software.

* 1. **The Fall and Falling system**

**Fall and Falling System Flowchart:**

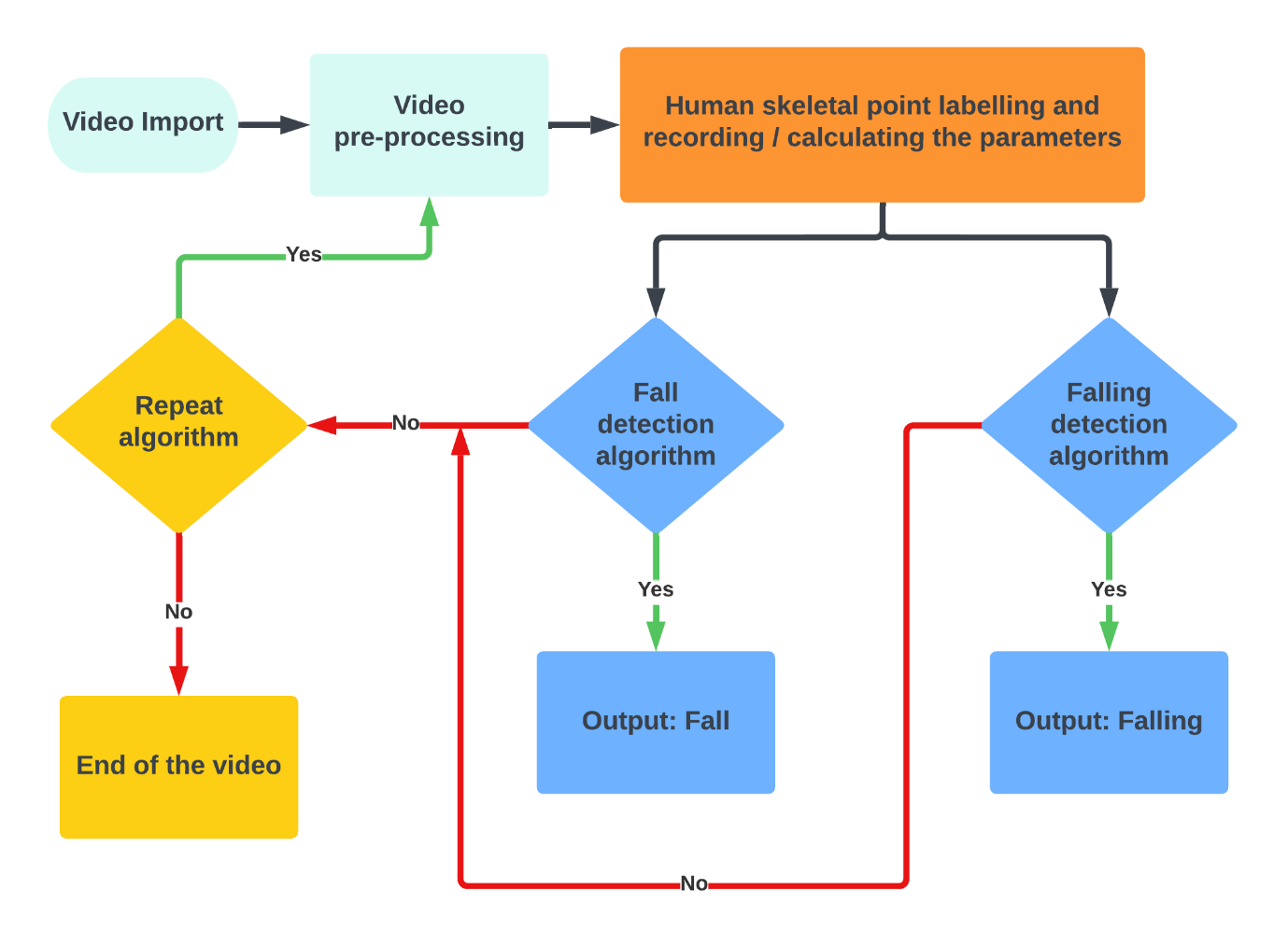


Figure 7: Fall and Falling Detection System Flowchart

Figure 7 is the flowchart of the whole fall and falling detection system. This system is separated into different sections by colours. The cyan colour part shows the video pre-processing stage which include video import and video pre-processing. The orange colour part shows the landmark extraction and parameter calculation stage. The blue colour part shows the fall and falling detection algorithms and the yellow part shows the repeat algorithm. The next section goes into much more detail about each of these panels.

* + 1. **Video Pre-processing demonstration and explanations**

This chapter elucidates the video information pre-processing module in the fall detection system, encompassing the extraction of frames from transmitted video information. A comprehensive flowchart is presented in the chapter, providing a detailed explanation of each keyword in the flowchart and illustrating how the module operates. The video information pre-processing module plays a vital role in the fall detection system, as it converts raw video data into usable frames for further analysis. The chapter's flowchart and detailed explanation offer a clear understanding of the module's workings.

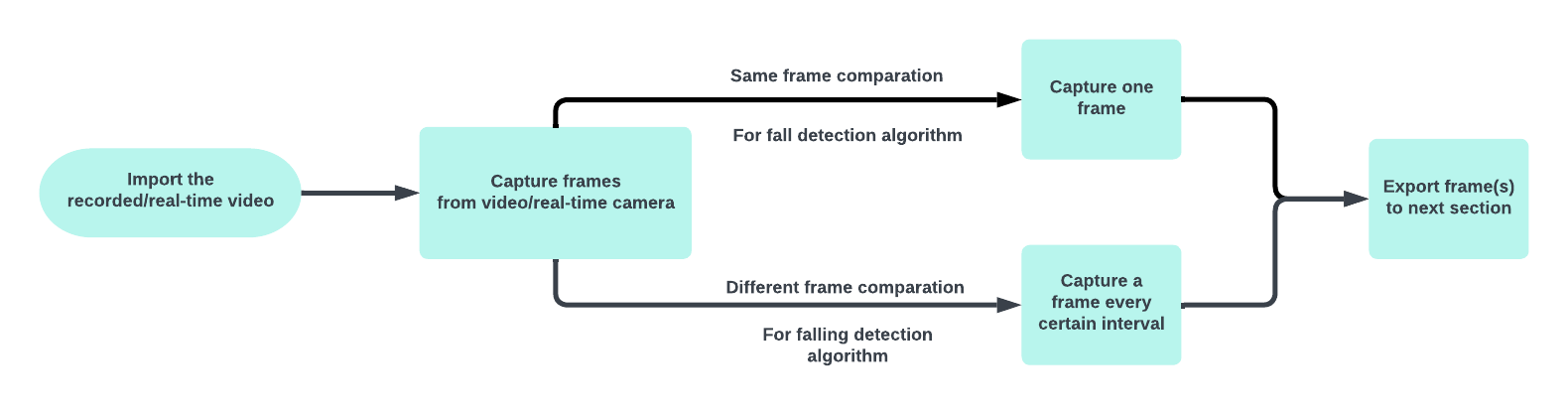


Figure 8: Video Pre-processing flowchart

The terms in the Figure 8 are explained as follows:

* **Import the recorded / real-time video:**

Transferring recorded or live video from cameras or similar capturing devices to a computer or other information processing devices.

* **Capture frames from recorded / real-time video:**

Extract the frames of the video recorded in the previous step or the live video. This step has two branches, the upper part extracts each frame arranged by video time, in preparation for a comparison within the same frame, and is also the basis of the algorithm for fall detection. The lower part extracts the adjacent frames with the same interval, in order to prepare a comparison between the different frames, which is the basis of the algorithm in falling detection.

* **Capture one frame:**

Usually a second of video consists of 24, 30 or 60 pictures. Extracting frames in chronological order based on video timecode involves processing frames in a sequential manner. For instance, in a video with a duration of 3 seconds and 30 frames per second, OpenCV would begin by extracting the first frame of the first second, and upon completing processing of this frame, proceed to iterate through subsequent frames of the first second based on a loop structure. If the frame passes the loop condition, OpenCV would proceed to extract the second frame of the first second, and so on and so forth, until all frames in the video have been processed.

* **Capture a frame every certain interval:**

Extracting frames at regular intervals based on video timecode. For instance, in a video with a duration of 3 seconds and 30 frames per second. The interval is set as 6 frames. OpenCV would begin by extracting the first frame of the first second, and then proceed to extract the seventh frame of the first second, based on the fixed interval of 6 frames. This process would repeat until all frames in the video have been processed. The comparison between different frames would take place between the ith frame and (i+6)th frame.

* **Export frame(s) to next section:**

Transferring the video frames extracted by OpenCV to the next stage.

* + 1. **Video Pre-processing code demonstration and explanations**

This chapter includes a code demonstration and explanation of the functionality implemented in the Video Pre-processing section. The code showcased in this chapter represents critical components of the functionality and provides an understanding of the implementation. The complete code is available in the appendix for reference.

**Code demonstration:**

while cap.isOpened():

        success,image = cap.read()

        if image is None:

            break

        if not success:

            print('Ignoring empty camera frame')

            break

        image.flags.writeable = False

        image = cv2.cvtColor(image,cv2.COLOR\_BGR2RGB)

        results = pose.process(image)

Figure 9: Video Pre-processing code

**Code explanation:**

while cap.isOpened():

Figure 9(a): Video Pre-processing code part 1

This line starts a ‘while’ loop that continues as long as the ‘cap’ object is open. This means that the loop will keep running until the video capture source (camera or video file) is available.

success,image = cap.read()

Figure 9(b): Video Pre-processing code part 2

This line reads the next frame from the video capture object ‘cap’. The ‘read()’ method returns a tuple with a boolean value ‘success’ indicating whether the frame ‘image’ was successfully read.

if image is None:

   break

if not success:

   print('Ignoring empty camera frame')

Figure 9(c): Video Pre-processing code part 3

These code checks if the ‘image’ variable is ‘None’. If it is, it means that there are no more frames to read. The third line checks if the ‘success’ variable is ‘False’. If it is, it means that there was an error reading the frame.

image.flags.writeable = False

Figure 9(d): Video Pre-processing code part 4

This line prevent the image from being modified, as the following processing steps do not require any modifications to original frame.

image = cv2.cvtColor(image,cv2.COLOR\_BGR2RGB)

Figure 9(e): Video Pre-processing code part 5

This line converts the color space of the ‘image’(which means the frame) from BGR format (Blue, Green, Red) to RGB (Red, Green, Blue). OpenCV reads images in BGR format by default, while Mediapipe pose (The next section) requires the input images in RGB format.

results = pose.process(image)

Figure 9(f): Video Pre-processing code part 6

This line processes the input image using the pose estimation model. The variable ‘pose’ is an instance of the ‘mp\_pose.Pose’ class from the MediaPipe library, which is configured with the desired detection and tracking confidence thresholds. The ‘process()’ method takes the input image and returns an object called ‘results’ that contains the detected pose landmarks and their respective coordinates (x, y, and z) for the given image.

* + 1. **Human body landmarks information processing demonstration and explanations**

This chapter explains the body information extraction module in the fall detection system. It covers the process of receiving image information from the previous chapter and extracting the body landmarks information, followed by calculating the parameters required for subsequent chapters. The chapter presents a detailed flowchart and provides a comprehensive explanation of each keyword in the flowchart, illustrating how this module works.

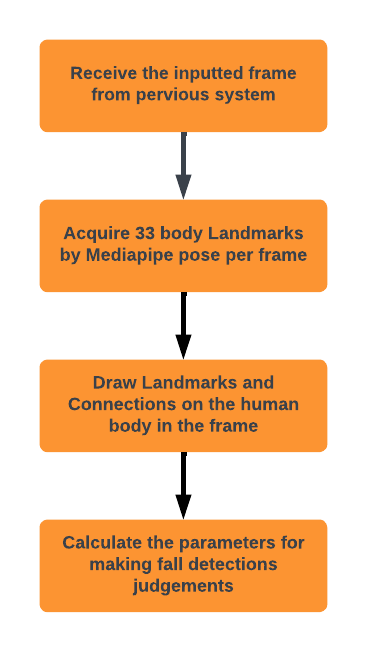


Figure 10: Human body landmarks information extraction flowchart

The terms in Figure 10 are explained as follows:

* **Receive the inputted frame from pervious system:**

Transferring frames extracted from OpenCV in the Video Pre-processing section to MediaPipe and import the video frame into the pipeline. This step involve reading the frame and converting it into a format that can be processed by Mediapipe. It includes resizing the image, applying colour correction and normalizing the pixel values to improve the accuracy of the body pose estimation model. The input images muse have a resolution of 256 × 256 pixels and the format must be in the RGB which means three channels (red, green, and blue) per pixel.

* **Acquire 33 body landmarks by MediaPipe Pose per frame:**

In the Introduction section, a comprehensive exposition of MediaPipe is presented. Once a frame is imported into the MediaPipe framework, a pre-trained model, namely the Full Body Pose Estimation model, is employed to detect and track up to 33 body landmarks per frame, including the nose, shoulders, hip, elbows, and ankles. The model employs advanced computer vision techniques and machine learning algorithms to analyse the input video frame and accurately locate these key points in real-time. Notably, each landmark is represented by four numerical values, namely x, y, z, and visibility. While the first three values indicate the 3D landmark position, the visibility parameter indicates the clarity of the point in the frame.

* **Draw Landmarks and connection line on the human body in the frame:**

The 33 3D body landmarks derived from the MediaPipe pose are received and marked on the body in the frame. A connection line is drawn between the two lines as seen in the Figure 4 in the introduction above.

* **Calculate the parameters for making fall detections judgements**:

In this step we define several parameters, they are s\_h\_high, s\_h\_long, h\_f\_high, h\_f\_long and rate1. Each of them has a specific meaning. s\_h\_high means the height from shoulder to hip; s\_h\_high means the length from shoulder to hip; h\_f\_high means the height from shoulder to hip; h\_f\_long means the length from shoulder to hip and rate1 indicates the direction of the body towards the camera. The difference between the height and length is that height is a vector but length is a scalar. For example, when the person being tested bows, the height difference from his shoulders to his hips shrinks, but the length does not change.

* + 1. **Human body landmarks information processing code demonstration and explanation**

This chapter includes a code demonstration and explanation of the functionality implemented in the Human body landmarks information processing section. The code showcased in this chapter represents critical components of the functionality and provides an understanding of the implementation. The complete code is available in the appendix for reference.

**Code Demonstration:**

with mp\_pose.Pose(min\_detection\_confidence=0.6,min\_tracking\_confidence=0.6) as pose:

        p\_lst = results.pose\_landmarks

#check input existence

        if p\_lst is None:

            print('no point is founded')

            continue

        else:

            for i in results.pose\_landmarks.landmark:

                lst.append((i.x, i.y, i.z, i.visibility))

        dir1[str(flag)]=(lst)

        flag+=1

#parameter setting

        shoulder\_wide = abs(lst[11][0] - lst[12][0])

        s\_h\_high = abs((lst[23][1]+lst[24][1]-lst[11][1]-lst[12][1])/2)

        s\_h\_long=np.sqrt(((lst[23][1]+lst[24][1]-lst[11][1]-lst[12][1] )/2)\*\*2+((lst[23][0]+lst[24][0]-lst[11][0]-lst[12][0] )/2)\*\*2)

        h\_f\_high = ((lst[28][1]+lst[27][1]-lst[24][1]-lst[23][1])/2)

        h\_f\_long=np.sqrt(((lst[28][1]+lst[27][1]-lst[24][1]-lst[23][1] )/2)\*\*2+((lst[28][0]+lst[27][0]-lst[24][0]-lst[23][0] )/2)\*\*2)

Figure 11: Human body landmarks information extraction code

**Code Explanation:**

with mp\_pose.Pose(min\_detection\_confidence=0.6,min\_tracking\_confidence=0.6) as pose:

Figure 11(a): Human body landmarks information extraction code part 1

This line creates an instance of the ‘Pose’ class from the ‘mp\_pose’ module. The instance is created with a minimum detection confidence of 0.6 and a minimum tracking confidence of 0.6. The ‘Pose’ object is used as a context manager in a ‘with’ statement.

The minimum detection confidence represents the confidence threshold that must be exceeded for a keypoint to be considered detected in an image and the minimum tracking confidence represents the confidence threshold that must be exceeded for a keypoint to be considered tracked from one image to the next in a video stream. [21]

Setting minimum detection confidence and minimum tracking confidence to 0.6 was determined to result in the most stable performance for keypoint extraction in the system. At thresholds of 0.5 and 0.7, varying degrees of errors were observed in keypoint extraction, leading to unstable performance. Therefore, selecting a threshold of 0.6 strikes an appropriate balance between the accuracy and robustness of pose estimation, thereby improving system performance.

p\_lst = results.pose\_landmarks

Figure 11(b): Human body landmarks information extraction code part 2

This line retrieves the 33 pose landmarks from the ‘results’ object. It is obtained in the pervious section. And the pose landmarks were assigned to the ‘p\_lst’ variable.

if p\_lst is None:

    print('no point is founded')

    continue

Figure 11(c): Human body landmarks information extraction code part 3

This line checks if ‘p\_lst’ is ‘None’, which means that no pose landmarks were detected in the processed frame. When it is, system prints the message ‘no point is founded’ to the console and jump to the next iteration of the loop.

else:

    for i in results.pose\_landmarks.landmark:

        lst.append((i.x, i.y, i.z, i.visibility))

h, w, \_ = image.shape

Figure 11(d): Human body landmarks information extraction code part 4

If the pose landmarks were detected in the processed frame, the ‘for’ loop iterates through all the pose landmarks detected in the processed frame and append a tuple to the ‘lst’ list. The tuple containing the x, y, z and the visibility of the current pose landmark ‘i’.

Normally for a complete appearance of a person in a frame, the system will extract 33 points, i.e. results will be 32. (Taking into account the presence of 0)

dir1[str(flag)]=(lst)

flag+=1

Figure 11(e): Human body landmarks information extraction code part 5

This line appends the list of pose landmarks, denoted as 'lst', to the dictionary 'dir1', utilizing the string representation of the variable 'flag' as the key. With each iteration, the 'flag' variable is incremented by 1, starting from 0. Upon completion, 'dir1' will encompass the x, y, z, and visibility values for all detected pose landmarks in the processed frame, designated as 'str(0)', which corresponds to the first entry in 'dir1'. In the subsequent frame, the system gathers all detected body landmarks from the second frame and consolidates them into 'str(2)' within 'dir1', representing the second element of the dictionary. The process continues in this manner for subsequent frames.

shoulder\_wide = abs(lst[11][0] - lst[12][0])

Figure 11(f): Human body landmarks information extraction code part 6

This line computes the **horizontal distance between the left and right shoulder** which are landmarks 11 and 12 ( lst[11][0] and lst[12][0] ). It assigns the absolute value to the variable ‘shoulder\_wide’. The '[0]' in lst[11][0] refers to the x-value of the 11th coordinate, and likewise, '[1]' represents its y-value, '[2]' represents its z-value, and '[3]' represents its visibility value according to the definition at Figure11(d).

s\_h\_high = abs((lst[23][1]+lst[24][1]-lst[11][1]-lst[12][1])/2)

Figure 11(g): Human body landmarks information extraction code part 7

This line calculates the average vertical distance between the shoulders (landmark 11, lst[11][1] and landmark 12, lst[12][1]) and the hips (landmark 23, lst[23][1] and landmark 24, lst[24][1]) and assign the absolute value to the variable ‘s\_h\_high’.

s\_h\_long=np.sqrt(((lst[23][1]+lst[24][1]-lst[11][1]-lst[12][1] )/2)\*\*2+((lst[23][0]+lst[24][0]-lst[11][0]-lst[12][0] )/2)\*\*2)

Figure 11(h): Human body landmarks information extraction code part 8

This line calculates the average Euclidean distance between the shoulders (landmarks 11 and 12) and the hips (landmarks 23 and 24) and assigns the value to the ‘s\_h\_long’ variable.

The Euclidean distance is calculated based on the Pythagorean theorem, which states that the square of the hypotenuse is equal to the sum of the squares of the other two sides.[35][36] The first part of the code ((lst[23][1]+lst[24][1]-lst[11][1]-lst[12][1] )/2)\*\*2) represents the square of the average height difference between the hips and shoulders, while the second part ((lst[23][0]+lst[24][0]-lst[11][0]-lst[12][0] )/2)\*\*2) represents the square of the average width difference between the hips and shoulders. These two parts represent the lengths of the two legs of a right triangle, and the hypotenuse calculated using the Pythagorean theorem represents the length from the person's shoulder to their hip. The reason for using the Euclidean distance is that this parameter remains constant when the person's posture changes, as it is not affected by changes in the person's movements.

h\_f\_high = ((lst[28][1]+lst[27][1]-lst[24][1]-lst[23][1])/2)

Figure 11(i): Human body landmarks information extraction code part 9

This line calculates the average vertical distance between the hips and the feet and assigns the value to the ‘h\_f\_high’ variable.

h\_f\_long=np.sqrt(((lst[28][1]+lst[27][1]-lst[24][1]-lst[23][1] )/2)\*\*2+((lst[28][0]+lst[27][0]-lst[24][0]-lst[23][0] )/2)\*\*2)

Figure 11(j): Human body landmarks information extraction code part 10

This line calculates the average Euclidean distance between the hips (landmarks 23 and 24) and the feet (landmarks 27 and 18) and assigns the value to the ‘h\_f\_long’ variable.

Explanation of the Euclidean distance is provided in Figure 11(h). The logic of these two lines of code is almost the same. However, this line of code calculates the Euclidean distance from the hips to the feet using the height and width differences between them.

* + 1. **Fall Detection System demonstration and explanations**

This chapter explains the fall detection module in the fall detection system. It presents various conditions for fall detection and explains the principles behind these conditions. The chapter proposes a detailed flowchart and provides a comprehensive explanation of each keyword in the flowchart, illustrating how this module works.

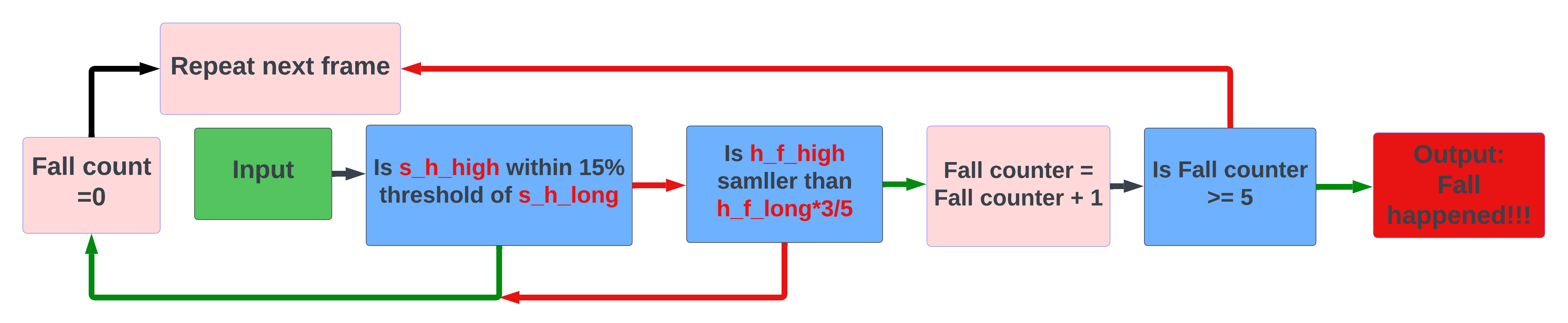


Figure 12: Fall Dectection System Flowchart

The terms in Figure 12 are explained as follows:

* **Input:**

This input includes the x, y, z and visibility values for 33 points on a frame and the five custom parameters calculated.

* **Is s\_h\_high within 15% threshold of s\_h\_long:**

This is the first condition of the Fall detection System. This term means whether the height from shoulder to hip in y axis is between 85% (length from shoulder to hip) and 115% (length from shoulder to hip). The purpose of this step is to confirm whether the upper body of the human in the current frame is upright, specifically from the shoulders to the hips. Since a fall involving an upright back is not a common occurrence, even if the back is initially upright during the fall, the subsequent movements will result in a significant bending of the back or lying down. Therefore, we utilize this feature as the first criterion for detecting falls.

The reason for defining 15% as the threshold in the project is based on human body upright posture considerations. Based on aggregated data on human height[34], the average height of an adult is approximately 170cm, with shoulder height from the ground around 130cm, chest height from the ground around 120cm, and hip height from the ground around 80cm. The system does not consider actions where the shoulder height is lower than the chest height when judging that a person is standing upright, as the chest height is approximately 120cm or 0.85 times the height of an adult. This value is derived from the detection of the average human height and can be adjusted based on the physical condition of elderly people to achieve better performance.

* **Is h\_f\_high smaller than h\_f\_long × :**

This is the second criterion for the fall detection system. This term refers to whether the height from the hips to the feet is less than 3/5 of the length from the hips to the feet. The purpose of this step is to confirm whether the lower part of the human body in the current frame is not upright. Since the position of the hips is almost always lowered during a fall, and most hip heights are reduced by at least half or more, this feature serves as the second criterion for detecting falls.

Based on the reference of human body posture[34], in most falling postures of the human body, the position of the hips falling will almost always fall within the height of the knees when standing. However, in special cases, such as falling forward and supporting the ground with both hands, the height of the hips will be at a position approximately at the height of the knees, which is half the length of the leg. In addition, due to subtle differences in the length of the lower and upper legs among different people, the system does not use 1/2 as the threshold for judgment, but instead uses 3/5 to provide a more lenient threshold for fall detection.

* **Fall count = 0:**

This is a parameter that calculate the detected fall frames. This step is for initializing the parameters when there is a non-fall frame happened.

* **Fall counter = Fall counter + 1:**

When the detected person in the current frame is determined to have fallen, the fall counter is automatically incremented by one, indicating that this frame is counted as a fall frame.

* **Is Fall counter >= 5:**

When there are five or more consecutive fall frames, we determine that the observed person is currently in a fall state.

* **Output: Fall happened:**

When the observed person is detected as in a fall state, generate a fall message out and pass to the IoT Alarm.

* **Repeat next frame:**

When a fall message is generated or a non-fallen frame is found in the middle, the system resets the Fall count to 0 and cycles through the next frames.

* + 1. **Fall Detection System Code demonstration and explanations**

This chapter includes a code demonstration and explanation of the functionality implemented in the fall detection algorithm. The code showcased in this chapter represents critical components of the functionality and provides an understanding of the implementation. The complete code is available in the appendix for reference.

**Code Demonstration:**

if s\_h\_high < s\_h\_long\*1.15 and s\_h\_high > s\_h\_long\*0.85:

    print(f'Not Fall')

    fall = 0

elif h\_f\_high < (3/5) \* h\_f\_long:

    fall+=1

else:

    fall=0

    print(f'Bend Over')

if fall>=5:

    print(f'fall')

counter += 1

    fall=0

Figure 13: Fall Dectection System code

**Code Explanation:**

if s\_h\_high < s\_h\_long\*1.15 and s\_h\_high > s\_h\_long\*0.85:

    print(f'Not Fall')

    fall = 0

Figure 13(a): Fall Dectection System Code part 1

This condition is the first step of the fall detection. It checks if the vertical distance between the shoulders and hips (s\_h\_high) is within 15% range of the Euclidean distance between the length of shoulders and hips (s\_h\_long). If the condition is met, the subject is considered to be in a not-fall position. The system will print the message ‘Not Fall’ to the console and reset the fall counter to 0.

elif h\_f\_high < (3/5) \* h\_f\_long:

    fall+=1

Figure 13(b): Fall Dectection System Code part 2

This condition is the second step of the fall detection. It checks if the vertical distance between the hips and feet ( h\_f\_high ) is less than 3/5 of the Euclidean distance between the hips and feet ( h\_f\_long ). If this condition is met, the subject is considered to a fall position and the fall counter is incremented by 1.

else:

    fall=0

    print(f'Bend Over')

Figure 13(c): Fall Dectection System Code part 3

This part of the code handles cases where the subject doesn’t met neither of these conditions. For example, in a posture like bending over, the shoulders are lower than the threshold of the first condition, but the legs are not bent.

if fall>=5:

    print(f'fall')

    fall=0

Figure 13(d): Fall Dectection System Code part 4

This condition checks if the fall counter reached 5, indicating that the subject has fallen for 5 consecutive frames. When it is, system will print a message ‘fall’ to the console and reset the fall counter to 0.

* + 1. **Falling Detection System demonstration and explanations**

This chapter explains the fall detection algorithm in the fall detection system. It proposes conditions for fall detection and explains the principles behind these conditions using a flowchart and comprehensive explanations of the keywords used.

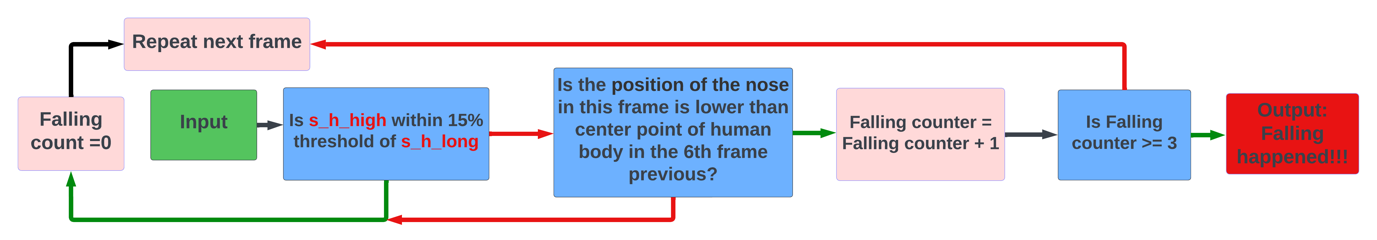


Figure 14: Falling detection System Flowchart

The terms in Figure 14 are explained as follows:

* **Input:**

In contrast to the input of the fall detection algorithm, this input provides information on all points in every frame within one second. Since fall detection is based on the detection of the skeletal points across different frames, the input information also includes the information between different frames.

* **Is the position of the nose in the frame is lower than centre point of human body in the 6th frame pervious:**

By comparing the heights of the nose and the body's centre point in the current frame, it is possible to determine whether a sudden decrease in body height is due to a fall or to actions such as crouching or lying down, by measuring the elapsed frames and the change in height. The nose and body centre are used as measuring points in this algorithm because, unlike other points, the height difference and speed change between these two points during a fall are relatively smooth, without sudden changes in speed due to different falling postures. For instance, the hands and feet are not suitable as speed judgment points for detecting falls since they may move rapidly during a fall, and each type of fall may have many differences that are difficult to discern. This comparison is made between the sixth and first frames, and extensive statistical calculations indicate that the common falling time of the human body is around 0.2 to 0.5 seconds, while the time it takes for the head to fall to the body's centre is about 0.1 to 0.3 seconds.Thus, in a 30 frames per second video, the current time standard for falls is six frames and 0.2 seconds, also it can be optimized for different individuals and video frame rates.

* **Is Falling counter >= 3:**

The algorithm defines that on the occurrence of three consecutive frames defined as falling, the algorithm determines that the observer is falling and outputs a signal that it is falling.

* + 1. **Falling Detection System code demonstration and explanations**

This chapter includes a code demonstration and explanation of the functionality implemented in the falling detection algorithm. The code showcased in this chapter represents critical components of the functionality and provides an understanding of the implementation. The complete code is available in the appendix for reference.

**Code Demonstration:**

1. if len(dir1) % 30 == 0:
2. interval = 1
3. time\_num = len(dir1)//interval
4. start\_num = len(dir1) - 30
5. falling1 = 0
6. for i in range(start\_num+6 ,time\_num):
7. now\_lst = dir1[str(i\*interval)]
8. pre\_lst = dir1[str((i-6)\*interval)]
9. s\_h\_high = (pre\_lst[23][1] - pre\_lst[11][1] + pre\_lst[24][1] - pre\_lst[12][1])/2
10. s\_h\_long=np.sqrt(((pre\_lst[23][1]+pre\_lst[24][1]-pre\_lst[11][1]-lst[12][1] )/2)\*\*2+((pre\_lst[23][0]+pre\_lst[24][0]-pre\_lst[11][0]-pre\_lst[12][0] )/2)\*\*2)
11. if s\_h\_high < s\_h\_long\*1.15 and s\_h\_high > s\_h\_long\*0.85:
12. print(f'not falling')
13. elif now\_lst[0][1] < 0.5\*((pre\_lst[11][1] + pre\_lst[12][1])/2):
14. print(f'falling step 1')
15. falling1 +=1
16. if falling1 >= 3:
17. print(f'falling situation now')
18. falling1= 0
19. else:
20. print(f'not falling neither')
21. falling1=0

Figure 15: Falling detection System Code

**Code Explanation:**

if len(dir1) % 30 == 0:

Figure 15(a): Falling detection System Code part 1

This condition checks if the length of the dictionary ‘dir1’ (which contains the pose landmarks for all the processed frames) is divisible by 30. The parameter is set to 30 because the video input to the system is 30 frames per second. This is an indication that the system processes every 30frames which is also one second.

interval = 1

time\_num = len(dir1)//interval

start\_num = len(dir1) - 30

falling1 = 0

Figure 15(b): Falling detection System Code part 2

The first line sets the interval between frames to be analysed to 1 and the second line calculates the total number of frames in 1 second to be analysed in falling detection system. For example, the video is 30 frames per second and the parameter ‘interval’ is 1. It means that the number of frames need to be calculated in this second is 30. If ‘interval’ is 2, it means that the number of frames need to be calculated in this second is 15. ‘start\_num’ means the starting frame number for the analysis, which is 30 frames before the current frame. ‘falling1’ initializes the falling counter to 0.

for i in range(start\_num+6 ,time\_num):

now\_lst = dir1[str(i\*interval)]

pre\_lst = dir1[str((i-6)\*interval)]

Figure 15(c): Falling detection System Code part 3

This for loop iterates through the frames from ‘start\_num+6’ to ‘time\_num’. The frame between these two parameter is 30 which indicates 30 frames. ‘now\_lst’ is assigned to the landmark data for the current frame. ‘pre\_lst’ is assigned to the landmarks data for the 6 frames prior to the current frame. 6 at here is a parameter that can be adjust by different frames rate of the video and other factors depending on the falling speed of individuals.

s\_h\_high = (pre\_lst[23][1] - pre\_lst[11][1] + pre\_lst[24][1] - pre\_lst[12][1])/2

s\_h\_long=np.sqrt(((pre\_lst[23][1]+pre\_lst[24][1]-pre\_lst[11][1]-lst[12][1] )/2)\*\*2+((pre\_lst[23][0]+pre\_lst[24][0]-pre\_lst[11][0]-pre\_lst[12][0] )/2)\*\*2)

if s\_h\_high < s\_h\_long\*1.15 and s\_h\_high > s\_h\_long\*0.85:

    print(f'not falling')

Figure 15(d): Falling detection System Code part 4

These line calculate the vertical and Euclidean distance between the shoulders and hips, which is similar to the parameter calculate in the previous section but the calculation is about the pervious frame. The following condition is same as the fall detection step 1 condition refer to figure 12(a).

elif now\_lst[0][1] < 0.5\*((pre\_lst[11][1] + pre\_lst[12][1])/2):

    print(f'falling step 1')

    falling1 +=1

    if falling1 >= 3:

        print(f'falling situation now')

        falling1= 0

else:

    print(f'not falling neither')

    falling1=0

Figure 15(e): Falling detection System Code part 5

This part of the code checks if the subject is falling or not.

0.5\*((pre\_lst[11][1] + pre\_lst[12][1])/2) is the centre of human body. This code indicates half of the average shoulder height which is also the height of body centre point. The ‘now\_lst[0][1]’ denotes the meaning of the height of nose. If the subject's nose in the current frame is lower than the body centre point in the previous 6th frame, the code prints "falling step 1", increments the falling counter ‘falling1’, and checks if it has reached 3. If so, the code prints "falling situation now" and resets the falling counter to 0. If the subject is neither standing nor falling, the code prints "not falling neither" and resets falling1 to 0.

* + 1. **Whole system with repeat system demonstration and explanations**

This section contains a detailed flowchart of the entire system and the various conditions that trigger a loop.

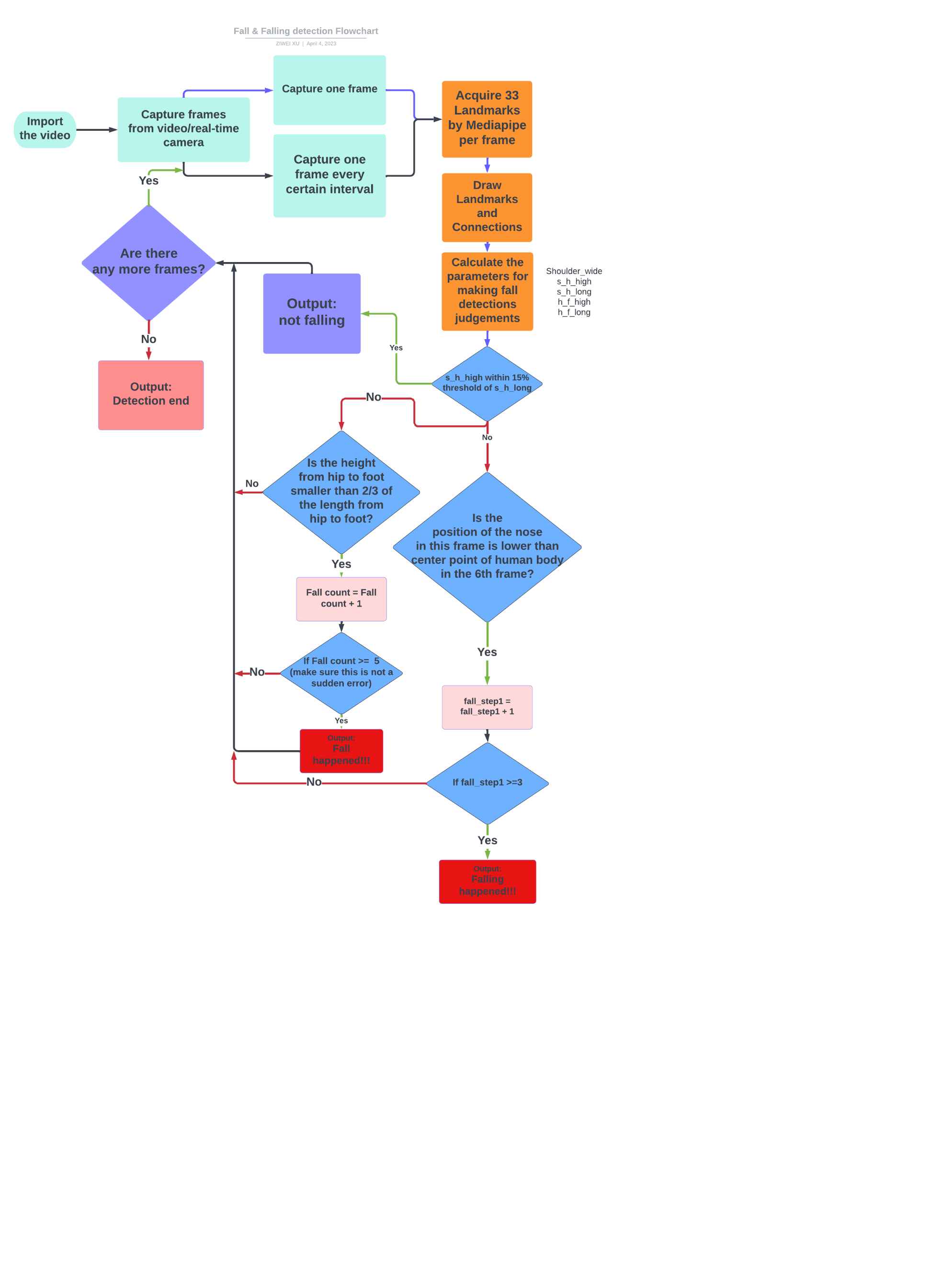


Figure 16: Whole system Flowchart

As there are many possibilities for triggering a loop system, a detailed explanation of the flow diagram of the whole system will facilitate an understanding of the way in which cycling occurs in this system.

Looping means reloading one or more frames and then making a judgement about the pose of the body on that frame. The system will continue to loop until all frames in the loaded video have been processed or the camera is turned off. However, there will still be cases where the loop system will be triggered early to start looping the next frame.

* **0.85\*s\_h\_long <= s\_h\_high <= 1.15 \* s\_h\_long**

**(Corresponding to s\_h\_high within 15% threshold of s\_h\_long)**

In the current frame, the detected individual has passed the upright evaluation. This suggests that the system has determined that the person is in an upright position within this particular frame. Considering that it is highly unlikely for a fall to occur while an individual is in an upright stance, the system concludes that no fall has taken place in this frame. Consequently, it proceeds to load and evaluate the subsequent frame.

**Fall detection:**

* **h\_f\_high > 3/5 \* h\_f\_long**

**(Corresponding to Is the height from hip to foot smaller than 2/3 of the length from hip to foot?)**

In this frame, the detected individual does not pass the second step of the fall detection, where the height from the hip to the foot is less than two-thirds of their total length. Consequently, the system determines that the observed behaviour in the frame is not a fall but rather a potentially confusable action, such as bending over or bowing. As a result, the evaluation is concluded, the subsequent frame is loaded, and the next iteration commences.

* **Fall count < 5**

**(Corresponding to If Fall count >= 5)**

The behaviour of the detected individual does not pass the third step of the fall detection process, in which the system has not yet generated five consecutive frames indicative of a fall. Prior to the occurrence of these frames, the system proceeds to the next frame for evaluation, starting from the frame extraction stage. Upon detecting five consecutive fall-indicative frames, a fall warning message is generated, and the loop system resumes after the message is issued. If a non-fall frame is detected within the sequence of five consecutive fall frames, the fall frame count is reset, a new frame is loaded, and the loop system continues.

**Falling detection:**

* **Falling count < 3**

**(Corresponding to If Falling count > 3)**

The behaviour of the detected individual does not pass the third step of the falling detection process. Similar to the Fall count in the pervious subsection, the system necessitates the identification of three consecutive frames that indicate a falling in order to categorize the detected individual as undergoing a falling process. Prior to the manifestation of these frames, the system iteratively extracts and evaluates subsequent frames. Upon recognizing three successive falling-indicating frames, a falling warning message is generated, and the iterative procedure resumes following the issuance of the message. In the event that a non-falling frame is detected within the series of three consecutive falling-indicating frames, the falling count is reset to zero, and the iterative process recommences.

**End of the Repeat system:**

Owing to the fact that the loop's condition necessitates iterating through all frames, the system will cease to receive additional frames upon the termination, pause, or closure of the video or camera feed. Consequently, the loop will come to a halt under such circumstances.

* 1. **The IoT Alarm**

In this section, this IoT Alarm requires a notification push software. It is a software that allows applications and websites to send notifications directly to a user’s device, even when the user is not actively using the application or website. These notifications can appear as a pop-up or banner on the user’s device, or as a badge on the app icon.

In this IoT alarm, PushDeer was used to send fall and falling alarming message to the family members’ phone. It was integrated into the whole system by using API provided by the platform. Also the open source and simplicity of this software was one of the key reasons why we adopted it.

**IoT Alarm Flowchart:**

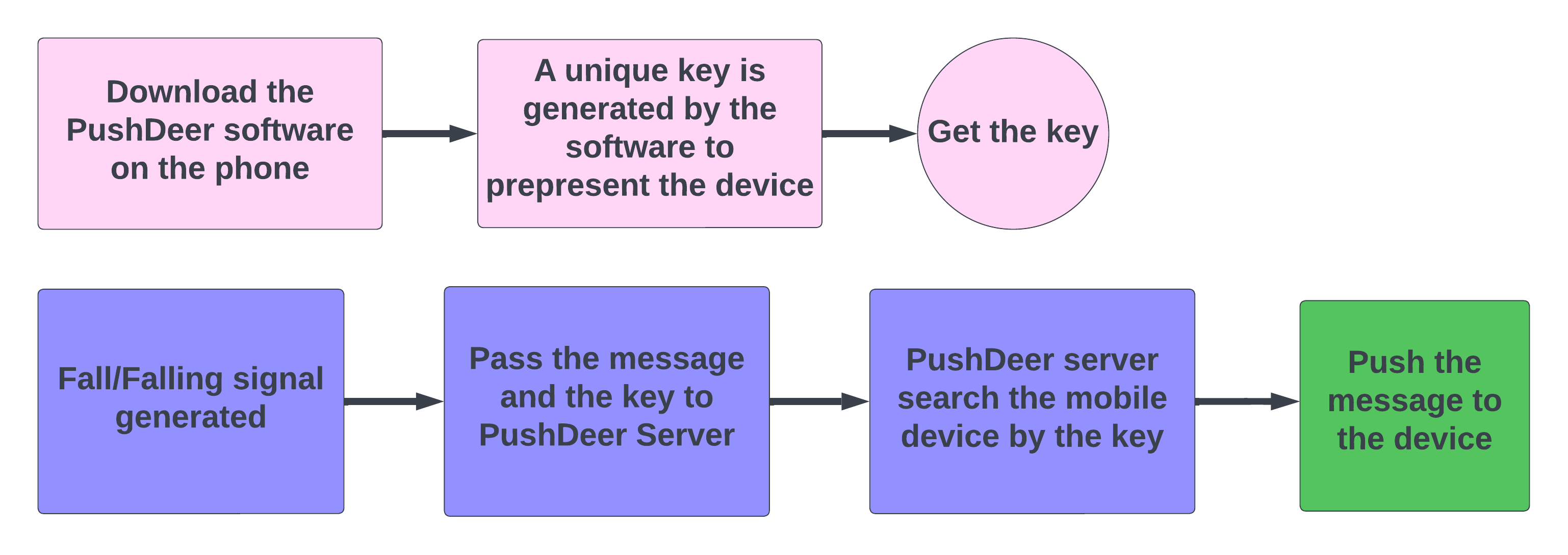


Figure 17: IoT Alarm Flowchart

The terms in Figure 17 are explained as follows:

* **IoT Alarm (Device terminal):**

To enable the PushDeer server to locate and communicate with the mobile device, it is necessary to download the PushDeer mobile application from a reputable application store or an authorized website. Once the mobile application is installed, it will generate a unique key for connection between the phone and PushDeer server. It is unique to each application or website that uses the service. It can be used to authenticate the sender’s request to send a push notification.

* **IoT Alarm (Network terminal):**

The fall and falling detection system determines the fall or falling state of the subject, and generates corresponding alarming message. This message, along with an encryption key, is transmitted to the PushDeer server over the network. Subsequently, the PushDeer server uses the received key to locate the matching device. Once a matching device is found, the received fall/falling information is transmitted to the corresponding device.

* + 1. **IoT Alarm Code demonstration and explanations**

This section explains the implementation of the IoT alarm's functionality through a code demonstration (Figure 18). It also provides a step-by-step explanation (Figure18(a) to (d)) of the principles behind the system's operation.

**Code demonstration:**

!pip install pypushdeer

from pypushdeer import PushDeer

pushdeer = PushDeer(pushkey="your push key")

pushdeer.send\_text("Fall", desp="optional description")

#pushdeer.send\_text(("Falling", desp="optional description")

Figure 18: IoT Alarm Code

**Code explantation:**

!pip install pypushdeer

Figure 18(a): IoT Alarm Code part 1

This system requires the pypushdeer package to be installed in order to use the PushDeer notification push service. This line installs the ‘pypushdeer’ package using Python package manager, ‘pip’ and the ‘!’ symbol at the beginning is used in the IDE called Jupyter Notebook environment to run shell commands.

from pypushdeer import PushDeer

Figure 18(b): IoT Alarm Code part 2

This line imports the ‘PushDeer’ class from the ‘pypushdeer’ package. The ‘PushDeer’ class provides methods for sending push notification using the PushDeer service.

pushdeer = PushDeer(pushkey="your push key")

Figure 18(c): IoT Alarm Code part 3

This line creates an instance of the ‘PushDeer’ class, passing the push key as an argument. The push key is a unique identifier associated with the device.[24], which is required to send push notifications.

pushdeer.send\_text("Fall", desp="optional description")

#pushdeer.send\_text(("Falling", desp="optional description")

Figure 18(d): IoT Alarm Code part 4

The ‘send\_text’ method takes two arguments: the title of the push notification (‘Fall’ in this case) and an optional description(‘optional description’ in this case). The method sends a push notification with the given title and description to the devices associated with the PushDeer software on that particular device.

1. **Datasets & Results**

In this chapter, the characteristics of the datasets required for the project, the three adopted datasets, and their respective results and analyses will be discussed.

* 1. **Ideal dataset criteria**

In order to comprehensively evaluate the accuracy and stability of a fall and falling detection algorithm, it is crucial to utilize multiple datasets during the testing process. An idealised dataset would need to have the following characteristics or functions：

**Variability in environment characteristics:**

Different datasets may contain variations in various factors such as lighting conditions, camera angles, subject demographics, and environmental settings. By using three datasets, the robustness and adaptability of the algorithm can be evaluated, ensuring that the fall detection system can operate effectively under a range of real-world conditions. Additionally, multiple datasets can effectively measure the algorithm's generalization ability. Generalization is a term used to demonstrate the extent to which a well-trained model can classify or predict data that it has not seen before [29]. In the context of fall detection algorithms, generalization is crucial because the algorithm must be able to accurately detect fall events in a wide range of scenarios, including different environments, subject demographics, and camera angles. A model that performs well on training data but struggles with new, unseen data has poor generalization and is of limited use in real-world applications.

**Variability of falls characteristics:**

Datasets should include a variety of falls, such as forward falls, backward falls, and lateral falls, among others. Including a variety of falls in a fall detection dataset would ensure that the fall detection system is capable of detecting falls from different angles and directions. This variety would help make the system more robust and reliable in real-world scenarios, where falls can occur in a variety of ways.

**Balancing false positives and false negatives:**

Fall detection systems must strike a delicate balance between detecting true fall events (minimizing false negatives) and avoiding false alarms (minimizing false positives). Using three datasets enables the assessment of the algorithm's performance in terms of sensitivity, specificity, and overall accuracy. This approach ensures that the system is capable of detecting fall events with high precision while simultaneously minimizing the occurrence of false alarms, which are particularly detrimental in real-world applications.

* 1. **Own dataset**

In the majority of fall detection video training datasets available online, it is challenging to find datasets containing multiple camera angles. Most fall videos feature a single or, at most, two camera perspectives, whereas measuring the model's accuracy necessitates considering factors from various angles. To address this issue, this dataset with four camera perspectives were created. The own datasets include three types of postures: sitting down, sitting down brutally, and falling down. The camera perspectives encompass facing the camera, having the back towards the camera, and facing the camera from both the left and right sides.

**Results demonstration:**

Table :

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Fall | | | | Sit | | | | Sit brutely | | | |
| Actual | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Predicted | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Results | TP | TP | TP | FN | TN | TN | TN | TN | TN | TN | TN | TN |

Table : Confusion matrix of Own Dataset

|  |  |  |
| --- | --- | --- |
| Total poses  P+N = 12 | Actual Value | |
| Predicted Value | TP = 3 | FP = 1 |
| FN = 0 | TN = 8 |

Table : Performance evaluation of Own Dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fall Precision | Non-Fall Precision | Accuracy | Sensitivity | F1 Score |
|  |  | 91.67% |  | 85.71% |

* 1. **50 ways to fall**

This fall dataset, released by Kevin Parry, showcases 50 distinct ways an individual can fall. [32] This video dataset was chosen to evaluate the system's accuracy due to its minimal background clutter, high contrast between the subject and the environment, overall clarity, and representation of a diverse range of falling postures. In addition to the 50 falling postures, the dataset also includes pre-fall positions, amounting to approximately 82 distinct postures in total. The dataset's exceptional diversity in terms of falling posture types and data features, as well as the transitions between pre-fall and fall positions, effectively contributes to adjusting the algorithm's sensitivity.

**Results Demonstration:**

Table : Confusion matrix of 50 ways to fall

|  |  |  |
| --- | --- | --- |
| Total poses  P+N = 82 | Actual Value | |
| Predicted Value | TP = 49 | FP = 1 |
| FN = 1 | TN = 31 |

Table : Performance Evaluation of 50 ways to fall

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fall Precision | Non-Fall Precision | Accuracy | Sensitivity | F1 Score |
| 98.00% | 96.87% | 97.56% | 98.00% | 97.79% |

* 1. **UR Fall detection dataset**

The UR Fall Dataset, released by the University of Rzeszow, comprises 30 falling and 40 daily life activity sequences.[33] These sequences include both depth and RGB image videos, totalling approximately 110 distinct postures after examination. The primary reason for adopting this dataset is its diversity. The dataset includes various lighting conditions, such as bright and dim scenes, which can be used to evaluate the algorithm's generalization capabilities. Additionally, it includes numerous simulated falling postures, which aid in balancing false positives and false negatives.

**Results Demonstration:**

Table : Confusion matrix of UR Fall Detection dataset

|  |  |  |
| --- | --- | --- |
| Total poses  P + N = 30 + 80 = 110 | Actual Value | |
| Predicted Value | TP = 26 | FP = 4 |
| FN = 9 | TN = 71 |

Table : Performance evaluation of UR Fall Detection dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fall Precision | Non-Fall Precision | Accuracy | Sensitivity | F1 Score |
| 86.67% | 88.75% | 88.18% | 74.29% | 80.64% |

Due to the fact that this algorithm does not specifically address the recognition of postures such as lying down and crouching, which are prone to confusion with falling, the data for these two postures in the UR fall detection dataset are automatically classified as falls. As a result, after excluding the five test cases involving lying down and crouching, the system's accuracy has been re-assessed as follow.

Table 8: Confusion matrix of UR Fall Detection dataset (without lying down and crouching)

|  |  |  |
| --- | --- | --- |
| Total poses  P + N = 30 + 75 = 105 | Actual Value | |
| Predicted Value | TP = 26 | FP = 4 |
| FN = 4 | TN = 71 |

Table : Performance evaluation of UR Fall Detection dataset (without lying down and crouching)

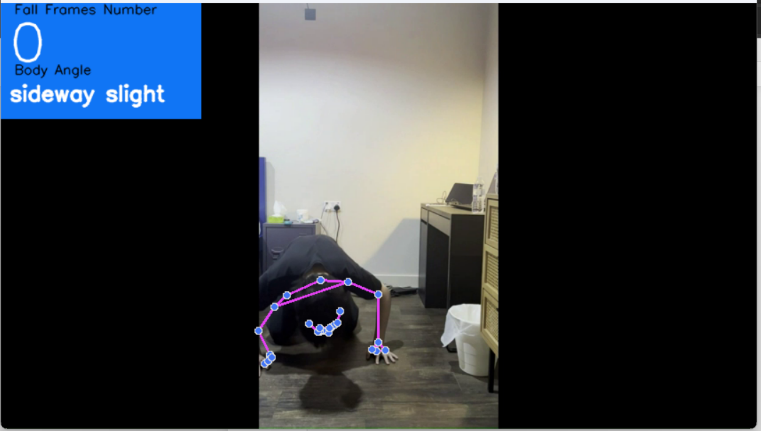
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fall Precision | Non-Fall Precision | Accuracy | Sensitivity | F1 Score |
| 86.67% | 94.67% | 92.38% | 86.67% | 86.67% |

* 1. **Results Evaluation**

Table : Performance evaluation of 3 datasets

|  |  |  |  |
| --- | --- | --- | --- |
|  | Own Dataset | 50 ways to Fall Dataset | UR Fall Detection Dataset |
| Fall Precision |  | 98.00% | 86.67% |
| Non-Fall Precision |  | 96.87% | 94.67% |
| Accuracy | 91.67% | 97.67% | 92.38% |
| Sensitivity |  | 98.00% | 86.67% |
| F1 Score | 85.71% | 97.79% | 86.67% |

According to the table, the system's performance varies greatly across different datasets. The first dataset, or the "own dataset," demonstrates that the algorithm is not particularly challenged by different viewing angles. However, the system still struggles with extracting the coordinates of the lower body key points when the upper body obstructs them during a forward fall (Figure 19 (left)). This issue was addressed in the second dataset, "50 ways to fall," where the clear background and high contrast between the person and the environment greatly improved key point extraction accuracy, leading to fewer false positives and a satisfactory performance. Nonetheless, the same problem reappeared in the UR Fall detection dataset due to low light conditions and cluttered surroundings, resulting in multiple false positives such as mistaking a chair for a person (Figure 19 (Middle)) or failing to accurately identify a person in dim light (Figure 19(Right)). These errors led to a system accuracy of around 90%, with an F1 score of 86.67%.

人站在房间里

中度可信度描述已自动生成黑暗的房间里玩体感游戏

低可信度描述已自动生成

Figure 19: Fall Forward (Left);Detection Failure of Chair (Middle);Detection failure of dim environment.(Right)

From the above situations, it can be seen that the stability of this technology is affected by factors such as the detection environment and whether the monitored subject is obstructed. In environments with cluttered surroundings, the accuracy rate is maintained at approximately 91% to 93%. If the environment and the subject are both clear, the accuracy rate can reach 97.67%, resulting in better system performance.

1. **Conclusion and Future work**

This section contains three subsections. The first subsection provides a general overview of the entire project. The second subsection focuses on the conclusions drawn from the data obtained, as well as the remaining questions that the project has raised. The third subsection discusses potential modifications to the project, its practical applications, and possible enhancements to its functionality.

* 1. **Project Summary**

The purpose of this project is to provide non-invasive fall detection through visual algorithm, which can instantly notify the elderly of their fall status to avoid delayed notification that could result in fatalities. This project utilizes Mediapipe pose technology to extract human keypoint coordinates, as well as OpenCV to extract video frames from RGB cameras and analyse them frame by frame. Based on threshold algorithm, the fall detection system exhibits excellent performance in environments with strong contrast between the subject and the surrounding environment, clear images, and few obstacles. However, in environments with many foreign objects and unclear overall human figures, the system does not give a satisfactory performance. It is therefore evident that the surrounding environment has a significant impact on the algorithm's performance. Once the system generates a fall signal, the IoT alarm will use the pre-set key to locate the device and transmit the fall information to the designated device, usually a family member, to notify them in real-time that the elderly has fallen.

* 1. **Project Conclusion**

The performance of the system varies in different environments, as concluded from a total of 199 detections across three datasets. In the Own dataset, the system achieved an accuracy of 91.67% and an F1 score of 85.71%. However, the F1 score may not reflect the performance of the dataset as it only tested 12 postures. This dataset mainly demonstrated the system's high tolerance towards different angles of the human body facing the camera, which does not cause confusion in the detection due to angle reasons. However, the system may fail to extract important human body coordinates, such as Figure 19 (left), if there is occlusion due to angle. In the 50ways to fall dataset, due to the strong contrast between the white background and the person and no other objects in the monitoring environment, the system achieved an accuracy of 97.67% and an F1 score of 97.79%. This dataset mainly proved that the algorithm can achieve satisfactory performance in testing human subjects and clear monitoring environments. However, in the UR Fall detection dataset, due to the complexity and diversity of the monitoring background, the accuracy demonstrated by the dataset is only 92.38% and the F1 score is 86.67%. This dataset proved that in dark environments or with too many surrounding objects, the system may fail to accurately extract human body key coordinates, such as Figure 19 (middle) and (right).

* 1. **Project Future Work**

The project has wide applicability, particularly in indoor monitoring of elderly people to prevent delayed emergency response due to delayed notification in the event of a fall. Potential application scenarios include homes, hospitals, nursing homes, and other monitoring environments. If the accuracy is further improved, it can be applied to outdoor monitoring cameras to evaluate pedestrians who fall in real-time and provide a strong foundation for smart city projects.

However, there are several issues to consider when applying this project. Firstly, the system cannot differentiate between postures that are similar to falls, such as lying down and prone postures. Therefore, it is necessary to further optimize and use fall detection algorithms, compare different frames, and determine whether the fall speed is controlled by the human body or caused by accidental falls. Specific measures can optimize the conditions of the fall detection code, such as using multiple conditions for the falling judgment, such as from head to shoulder, from shoulder to the centre of the human body. By using multiple-step judgments and parameter adjustments to make this algorithm more accurate. The verification method can be used to test the falling dataset to determine whether the algorithm is improved.

Secondly, the detection accuracy of the system is sensitive to environmental conditions. When the environment is too dark, cluttered, or crowded, it cannot accurately detect human actions. This limitation is caused by the inherent characteristics of the Mediapipe posture and can be mitigated by using other keypoint extraction frameworks (such as Yolov5) or ensuring clean and suitable environments in practical applications. After replacing the keypoint extraction framework, test the accuracy of the dataset again and observe whether there is an improvement. Then test different scenarios and environments with various datasets to identify specific factors that affect the extraction of human key coordinates.

Thirdly, the current PushDeer notification push software does not provide a method for transmitting images. If it is possible to capture and transmit images of falls to a designated device, it can effectively analyse the severity of falls among elderly people and determine whether they can be handled alone or need assistance. This can be achieved by using other open-source message push software that can push images or by creating one's own. The transmission speed and quality of photos can be used to determine whether this information push software is suitable.

Finally, the camera used in this project is currently connected to a computer for information processing, but in the future, a microcontroller can be used to connect to monitoring cameras and transmit real-time video information to servers for processing, reducing usage costs and expanding the scope of applications.

1. **Reference**

[1] World Health Organization, "World Report on Ageing and Health," Geneva, Switzerland: World Health Organization, 2015.

[2] C. Iftimoaei, "Demographic Aging in France and Romania. A Comparative Statistical Analysis," in 2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Pitesti, Romania, 2019, pp. 1-4.

[3] World Health Organization, "WHO Global report on falls prevention in older age," Geneva, Switzerland: World Health Organization, 2007.

[4] National Council on Ageing, "Falls among older adults: risk factors and prevention strategies," in Fall Free: Promoting a National Falls Prevention Action Plan, J.A. Stevens, Ed. 2005.

[5] Centers for Disease Control and Prevention. (2016). Web-based Injury Statistics Query and Reporting System (WISQARS). Retrieved from <https://www.cdc.gov/injury/wisqars/index.html>

[6] "Why is a quick response so important when the elderly fall?" Helpline, 2021. [Online]. Available: <https://helpline.co.uk/blogs/articles/why-is-a-quick-response-so-important-when-the-elderly-fall>. [Accessed: April 8, 2023].

[7] P. T. Lewis, "The Next Decade in Information Technology," Proceedings of the Joint Computer Conference, San Francisco, CA, USA, 1985, pp. 537-549.

[8] IBM, "What is the Internet of Things (IoT)?," [Online]. Available: <https://www.ibm.com/cloud/learn/internet-of-things>. [Accessed: April 08, 2023].

[9] Lord, C. J., and Colvin, D. P. (1991). “Falls in the elderly: detection and assessment,” in Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society (Orlando, FL: IEEE), 1938–1939.

[10] Bourke, A., O’brien, J., and Lyons, G. (2007). Evaluation of a thresholdbased tri-axial accelerometer fall detection algorithm. Gait Post.26,194-199. doi:10.1016/j.gaitpost.2006.09.012

[11] W. Xueyi, J. Ellul, and G. Azzopardi, "Elderly Fall Detection Systems: A Literature Survey," Frontiers in Robotics and AI, vol. 7, pp. 71, 2020. doi: 10.3389/frobt.2020.00071.

[12] Rougier, C., Meunier, J., St-Arnaud, A., and Rousseau, J. (2011b). Robust video surveillance for fall detection based on human shape deformation. IEEE Trans. Circ. Syst. Video Technol. 21, 611–622. doi: 10.1109/TCSVT.2011.2129370

[13] Charfi, I., Miteran, J., Dubois, J., Atri, M., and Tourki, R. (2012). Definition and performance evaluation of a robust SVM based fall detection solution. SITIS 12, 218–224. doi: 10.1109/SITIS.2012.155

[14] Rougier, C., Auvinet, E., Rousseau, J., Mignotte, M., and Meunier, J. (2011a). “Fall detection from depth map video sequences,” in International Conference on Smart Homes and Health Telematics (Montreal: Springer), 121–128. doi: 10.1007/978-3-642-21535-3\_16

[15] Z. Huang, Y. Liu, Y, Fang, and B.K.P.Horn, “Video-based Fall Detection for Seniors with Human Pose Estimation,” in 2018 4th International Conference on Universal Village(UV),2018, pp.1-4,doi:10.1109/UV.2018.8642130

[16] C.-B. Lin, Z. Dong, W.-K.Kuan, and Y.-F. Huang, “A Framework for Fall Detection Based on OpenPose Skeleton and LSTM/GRU Models,” Applied Sciences, vol.11,no.1.2021,doi:10.3390/app11010329.

[17] G.Sun and Z. Wang, “Fall detection algorithm for the elderly based on human pose estimation,” in 2020 Asia-Pacific Conference on Image Processing. Electronics and Computers (IPEC), 2020, pp.172-176,doi:10.1109/IPEC49694.2020.9114962.

[18] M.D. Solbach and J.K.Tsotsos,”Vision-Based Fallen Person Detection For the Elderly,” in 2017 IEEE International Conference on Computer Vision Workshops (ICCVW),2017,pp.1433-1442,doi:10.1109/ICCVW.2017.170

[19] W. Lina and J.K. Tsotsos, “Vision-Based Fallen Person Detection for the Elderly,” in 2017 IEEE International Conference on Communications, Information System and Computer Engineering (CISCE), 2020, pp.326-330,doi:10.1109/CISCE50729.2020.00072

[20] J. Brieva, H. Ponce, E. Moya-Albor, and L. Martinez-Villasenor, “An intelligent Human Fall Detection System Using a Vision-Based strategy,” in 2019 IEEE 14th International Symposium on Autonomous Decentralized System (ISADS), 2019, pp.1-5,doi:10.1109/ISADS45777.2019.915567

[21] Google. (2021, November 19). Google Mediapipe. [Online]. Available: <https://github.com/google/mediapipe>. [Accessed: Apr. 09, 2023].

[22] Google. (2021, March 16). MediaPipe Pose Landmark Model. [Online]. Available: <https://developers.google.com/mediapipe/solutions/vision/pose_landmarker/>. [Accessed: Apr. 09, 2023].

[23] "OpenCV," Wikipedia, the free encyclopedia. [Online]. Available: <https://en.wikipedia.org/wiki/OpenCV>. [Accessed: Apr. 09, 2023].

[24] Easy Chen, "PushDeer," GitHub. [Online]. Available: <https://github.com/easychen/pushdeer>. [Accessed: Apr. 09, 2023].

[25] Sunwish, “PushDeer schematic”, PushDeer, July. 24, 2022.[Online]. Available: https://zhuanlan.zhihu.com/p/545641082. [Accessed: April,09,2023].

[26] Google, "MediaPipe," GitHub. [Online]. Available: <https://github.com/google/mediapipe>. [Accessed: Apr. 09, 2023].

[27] C. A. Q. Bugarin, J. M. M. Lopez, S. G. M. Pineda, M. F. C. Sambrano and P. J. M. Loresco, "Machine Vision-Based Fall Detection System using MediaPipe Pose with IoT Monitoring and Alarm," 2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC), Hyderabad, India, 2022, pp. 269-274, doi: 10.1109/R10-HTC54060.2022.9929527.

[28] Adrian Rosebrock, "Convolutional Neural Networks (CNNs) and Layer Types," PyImageSearch, May 14, 2021. [Online]. Available: <https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/>. [Accessed: Apr. 09, 2023].

[29] DeepAI. (n.d.). What is Generalization in Machine Learning? [Online]. Available: <https://deepai.space/what-is-generalization-in-machine-learning/#:~:text=Definition%20of%20generalization%3F%20In%20machine%20learning%2C%20generalization%20is,it%20works%20for%20all%20subset%20of%20unseen%20data>.. [Accessed: Apr. 09, 2023].

[30] Powers, David M. W. (2011). "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation". *Journal of Machine Learning* Technologies. 2 (1): 37–63. S2CID 55767944.

[31] Zaki, Zunash & Shah, Muhammad Arif & Wakil, Karzan & Sher, Falak. (2020). LOGISTIC REGRESSION BASED HUMAN ACTIVITIES RECOGNITION. JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES. 15. 228-246. 10.26782/jmcms.2020.04.00018.

[32] "50 ways to fall," YouTube video, duration 3:52, posted by Kevin Parry, Oct. 18, 2018. [Online]. Available: <https://www.youtube.com/watch?v=8Rhimam6FgQ>. [Accessed: Apr. 10, 2023].

[33] B. Kwolek and M. Kepski, "Human fall detection on embedded platform using depth maps and wireless accelerometer," Computer Methods and Programs in Biomedicine, vol. 117, no. 3, pp. 489-501, Dec. 2014, doi: 10.1016/j.cmpb.2014.08.003.

[34] 00, " Human standing and sitting dimensions," Zhihu, 2020. [Online]. Available: <https://zhuanlan.zhihu.com/p/102724605>. [Accessed: Apr. 12, 2023].

[35] "Pythagorean Theorem," Wikipedia, The Free Encyclopedia, 10 Mar. 2022. [Online]. Available: <https://en.wikipedia.org/wiki/Pythagorean_theorem>. [Accessed: 14 Apr. 2023]

[36] "Euclidean distance," Wikipedia, The Free Encyclopedia, 9 Mar. 2022. [Online]. Available: <https://en.wikipedia.org/wiki/Euclidean_distance>. [Accessed: 14 Apr. 2023].

1. **Appendix**

!pip install mediapipe opencv-python

import cv2

import mediapipe as mp

import numpy as np

import requests

# declear and import data

mp\_drawing =mp.solutions.drawing\_utils

mp\_drawing\_styles =mp.solutions.drawing\_styles

mp\_pose = mp.solutions.pose

#record all the 33 different key landmarks in each frame of the video

dir1={}

#Key for the video parameter

flag=0

#Count how many fall frame happened

counter = 0

#Initialize the body angle to the camera

body\_angle='front'

#body position parameter

sideway\_slight=0

sideway\_whole=0

front = 0

#Fall parameters

fall=0

#the path of video to import

#video\_path = r'C:\Users\22095\Desktop\Third year project\testing videos\fall\_front2.mp4'

video\_path = r'C:\Users\22095\Desktop\Third year project\testing videos\Fall\_test.mp4'

cap = cv2.VideoCapture(video\_path)

cap = cv2.VideoCapture(1)

with mp\_pose.Pose(min\_detection\_confidence=0.6,min\_tracking\_confidence=0.6) as pose:

    while cap.isOpened():

    #while True:

        success,image = cap.read()

        if image is None:

            break

        if not success:

            print('Ignoring empty camera frame')

            #loading a video, use break

            #real-time, use continue

            break

        image.flags.writeable = False

        image = cv2.cvtColor(image,cv2.COLOR\_BGR2RGB)

        results = pose.process(image)

        lst=[]

        p\_lst = results.pose\_landmarks

        #check whether there is an input

        if p\_lst is None:

            print('no point is founded')

            continue

        else:

            for i in results.pose\_landmarks.landmark:

                lst.append((i.x, i.y, i.z, i.visibility))

        h, w, \_ = image.shape

        print('test')

 #Parameter settings

        ##  设肩宽为38 上半身60 躯干的比例为0.6

        ##  因为有可能弯腰，肩胯长不该是肩与胯的高度之差，但由于无法精确地确定身体的角度，无法准确地求出上半身的长度

        shoulder\_wide = abs(lst[11][0] - lst[12][0])

        print(lst[23][1],lst[24][1],'\t',lst[11][1],lst[12][1])

        #the height of the shoulder to hip

        s\_h\_high = abs((lst[23][1]+lst[24][1]-lst[11][1]-lst[12][1])/2)

        #the lenght between the shoudler and the hip

        s\_h\_long=np.sqrt(((lst[23][1]+lst[24][1]-lst[11][1]-lst[12][1] )/2)\*\*2+((lst[23][0]+lst[24][0]-lst[11][0]-lst[12][0] )/2)\*\*2)

        #the height of the hip to feet

        h\_f\_high = ((lst[28][1]+lst[27][1]-lst[24][1]-lst[23][1])/2)

        #the length between the hip and the feet

        #这是在两只脚与肩膀同宽的情况下，如果可以需要把这个设为定值

        h\_f\_long=np.sqrt(((lst[28][1]+lst[27][1]-lst[24][1]-lst[23][1] )/2)\*\*2+((lst[28][0]+lst[27][0]-lst[24][0]-lst[23][0] )/2)\*\*2)

    #test code for avoiding bow detect as fall

    #=========================================================================

        #the length between hip and the palm which is the ground

        h\_g\_high = abs((lst[23][1] + lst[24][1] - lst[29][1] - lst[30][1])/2)

        h\_g\_long=np.sqrt(((lst[32][1]+lst[31][1]-lst[24][1]-lst[23][1] )/2)\*\*2+((lst[32][0]+lst[27][0]-lst[31][0]-lst[23][0] )/2)\*\*2)

        #the length between shoulder and the ground

        s\_g\_high = abs((lst[11][1] + lst[12][1] - lst[29][1] - lst[30][1])/2)

    #=========================================================================

        rate1=shoulder\_wide/s\_h\_high

# determine the position

        if 0.2< rate1 < 0.4:

            sideway\_slight +=1

            sideway\_whole = 0

            front = 0

        elif rate1< 0.2 :

            sideway\_whole+=1

            sideway\_slight=0

            front = 0

        else:

            sideway\_whole  = 0

            sideway\_slight = 0

            front = 0

        if sideway\_slight >= 3:

            print(f'sideway slight')

            sideway\_slight = 0

            body\_angle = 'sideway slight'

        elif sideway\_whole >= 3:

            print(f'sideway whole')

            sideway\_whole = 0

            body\_angle = 'sideway whole'

        else:

            front += 1

        if front >= 3:

            body\_angle = 'front'

            front = 0

            print('front')

        print('s\_h\_high: ',s\_h\_high, 's\_h\_long: ',s\_h\_long,'h\_f\_high: ', h\_f\_high,'h\_f\_long: ')

 #Fall detection Step 1

#first Part test code for detect Not Fall

#==================================================================================

        if s\_h\_high < s\_h\_long\*1.15 and s\_h\_high > s\_h\_long\*0.85:

            #在这个区间意味着人上半身是直立的，而跌倒不可能为直立，所以不是

            print(f'Not Fall')

            fall = 0

#==================================================================================

 #Fall detection Step 2

        elif h\_f\_high < (3/5) \* h\_f\_long:

            #跌倒的时候臀部的高度一定会低于2/3的腿长，

            #参考四种跌倒方式。所以在上面排除了上半生直立的可能性的时候，臀部低于2/3的腿长时可以认定为摔倒

            fall+=1

        else:

            fall=0

            print(f'Bend Over')

        if fall>=5:

            print(f'fall')

            counter += 1

#Notification code

            api="https://api2.pushdeer.com/message/push?pushkey=PDU20611T3PFk9T08TwCmUuMcALjVdsJjOjgOAIRR & text= Fall  "

            req = requests.post(api)

            fall=0

            print(lst[0][1],'\t',lst[11][1],'\t',lst[23][1])

        print('============================================================================================================')

        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)

                                 )

        #dri1 is a dictionary, it contain severial frames with 33 different keypoints information

        #flag is the time parameter

        #for different frame comparation

        dir1[str(flag)]=(lst)

        flag+=1

        #5 is interval

        #because it is not quite easy to make a comparation between 2 frames

        if len(dir1) % 30 == 0:

            interval = 1

            time\_num = len(dir1)//interval

            start\_num = len(dir1) - 30

            falling1 = 0

            print(time\_num)

            for i in range(start\_num+6 ,time\_num):

                now\_lst = dir1[str(i\*interval)]

                pre\_lst = dir1[str((i-6)\*interval)]

                s\_h\_high = (pre\_lst[23][1] - pre\_lst[11][1] + pre\_lst[24][1] - pre\_lst[12][1])/2

                s\_h\_long=np.sqrt(((pre\_lst[23][1]+pre\_lst[24][1]-pre\_lst[11][1]-lst[12][1] )/2)\*\*2+((pre\_lst[23][0]+pre\_lst[24][0]-pre\_lst[11][0]-pre\_lst[12][0] )/2)\*\*2)

                if s\_h\_high < s\_h\_long\*1.15 and s\_h\_high > s\_h\_long\*0.85:

                    print(f'not falling')

                elif now\_lst[0][1] < 0.5\*((pre\_lst[11][1] + pre\_lst[12][1])/2):

                    print(f'falling step 1')

                    falling1 +=1

            #         if now\_lst[0][1] < 0.3\*((pre\_lst2[11][1] + pre\_lst2[12][1])/2):

            #             print(f'falling step 2')

            #             falling2 +=1

                    if falling1 >= 3:

                        print(f'falling situation now')

                        falling1= 0

# Notification Code

#                         api="https://api2.pushdeer.com/message/push?pushkey=PDU20611T3PFk9T08TwCmUuMcALjVdsJjOjgOAIRR & text= Falling"

#                         req = requests.post(api)

                else:

                    print(f'not falling neither')

                    falling1=0

#       print(dir1)

        #setup status box

        cv2.rectangle(image,(0,0),(225,130),(245,117,16),-1)

        # rep data

        cv2.putText(image,'Fall Frames Number',(15,12),

                    cv2.FONT\_HERSHEY\_SIMPLEX,0.5,(0,0,0),1,cv2.LINE\_AA)

        cv2.putText(image,'Body Angle',(15,80),

                    cv2.FONT\_HERSHEY\_SIMPLEX,0.5,(0,0,0),1,cv2.LINE\_AA)

        cv2.putText(image,str(counter),(10,65),

                     cv2.FONT\_HERSHEY\_SIMPLEX,2,(255,255,255),2,cv2.LINE\_AA)

        cv2.putText(image,str(body\_angle),(10,110),

                     cv2.FONT\_HERSHEY\_SIMPLEX,0.8,(255,255,255),2,cv2.LINE\_AA)

#if need a selfie-view display , change image to cv2.flip(image,1)

        cv2.imshow('Mediapipe Feed', image)

        #5 is interval

        #because it is not quite easy to make a comparation between 2 frames

        interval = 1

        time\_num = len(dir1)//interval

        falling1 = 0

        falling2 = 0

        falling = 0

        print(time\_num)

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

            break

    cap.release()

    cv2.destroyAllWindows()