Human Fall Detection using Visual Tracking with Background Modelling

Sasha Annabel
Department of Computer Science and Electronics
Gadjah Mada University
Yogyakarta, Indonesia
sashaannabel@mail.ugm.ac.id

***Abstract*—This paper talks about proposed methods to detect humans falling using background modelling for visual tracking as it is more time-and-cost-efficient since no data resource is required for model training.**

Keywords—computer vision, visual tracking, background modelling, bounding box

I. Introduction

For the healthcare industry, technological advancement in computer vision has opened new avenues for automatic patient monitoring. Computer vision systems can surveil patients continuously to detect emotions such as signs of distress, to recognize any abnormal movements, or acknowledge any changes in vital signs. With this capability, human resources can be allocated to more pressing matters than surveillance, which is incredibly beneficial in intensive care units (ICUs), hospital dorms, and elderly care facilities, where the number of patients greatly exceed the capacity of working doctors or nurses.

II. Problem Statement

For the elderly or any persons with mobility issues, falling can cause injuries, whether light and treatable or fatal. This accident can be **particularly dangerous if it occurs in private places where no guards or nurses are on standby to watch**. Wearable sensors have been used to alert the related carepersons whether a person has fallen or not, but this approach has been popularly rejected because bracelet sensors are uncomfortable and easily forgotten. Thankfully, computer vision applications has been able to replace these sensors for non-contact observation instead. Most used deep learning models, particularly Convolutional Neural Networks (CNNs), as this type of model gives high accuracy in human posture detection and activity recognition. However, the problems are that these models often require substantial computational resources and large annotated datasets for training, which is not desirable in real-time application. They can also be very costly for some resource-constrained environments to obtain large learning datasets in the first place. As such, there must be another solution where the priority is in the fast-processing time of the human fall detection system, as well as ability to function without any previous dataset for model learning.

III. Proposed Solution

As mentioned in the precious section, this project attempts to address the need for an **efficient**, real-time fall detection system that operates **without the complexity and resource demands of deep learning models**. By employing classical computer vision techniques, specifically background subtraction and aspect ratio analysis of bounding boxes, the proposed system aims to accurately differentiate between a standing and a fallen person in video inputs. Focusing on **detecting movements of one singular person (private monitoring)**, this approach will be able to track the human object by leveraging the geometric properties of the human silhouette and inform the decision whether a person within the video recording has fallen.

IV. Methods Designed

A. Video Acquisition

Since no large datasets will be used, the camera takes only videos for trials and testing. Trial videos are for parameter measurement, whereas the testing videos will show how the program effectively detects a falling person (or *none* detected, if there is no human falling). It is noteworthy that although the goal is to detect elderly persons falling, test subjects will **not** be elderly as it is deemed unsafe, and instead will be young volunteers.

There is an important constraint in which the camera must be in a static, non-moving position during the entire time of the video acquisition, so that only the moving human is detected. The author assumes this to not be a problem as CCTVs and monitoring cameras have been commonly placed in the same constant corners or places (where the camera stays still at all times).

B. Human Object Detection with Background Modelling

Background modeling method is used to distinguish moving object from the static background in a video. Some techniques involved frame differencing, which compares consecutive frames to detect changes; running average, which averages frames over time to maintain a dynamic background model; and Gaussian Mixture Model (GMM), which models each pixel with multiple Gaussian distributions to handle variations in the scene.

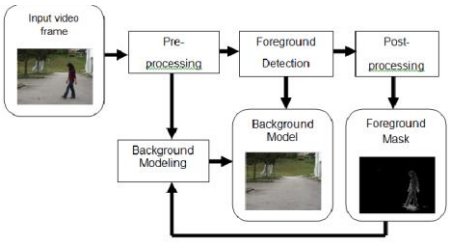

Fig. 1. Diagram of how background subtraction method works

As this project primarily focuses singular monitoring, the Background Subtractor MOG2 from OpenCV is deemed effective as it combines the strengths of GMM for robust background modeling and adapts quickly to changes in the scene. Additionally, MOG2 effectively handles shadow detection, improving the accuracy of distinguishing between foreground and background objects.

C. *Object Size Measurement with Bounding Box*

A bounding box is a rectangular frame used in computer vision to define the position and dimensions of an object within an image. In the program, the bounding box is drawn around the largest detected contour, which is assumed to be a person. To determine if the person is falling or standing, the program calculates the **aspect ratio of the bounding box**; a wider (horizontal) aspect ratio indicates a fall, while a taller (vertical) aspect ratio indicates standing.

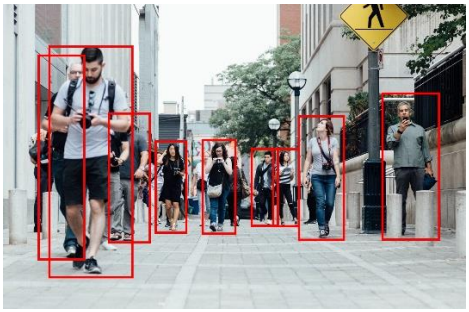

Fig. 2. Example of how bounding boxes detect moving objects

D. *Code Implementation*

The *detect_fall()* function handles the segmentation process of getting the moving human object (largest contour found in the frame), then puts the bounding box and decides whether the aspect ratio of the bounding box can determine a falling/standing person.

```
pplies background substraction to get the human object and decides if it is 'falling' or not
def detect_fall(frame, bg_subtractor, min_width=30, min_height=30):
    #gets the threshold for the mask
    fg_mask = bg_subtractor.apply(frame)
    _, fg_mask = cv2.threshold(fg_mask, 200, 255, cv2.THRESH_BINARY)

    #gets the biggest contour, assuming it is human
    #there can be a case where no moving object is detected --> program just continues analyzing next frame
    contours, _ = cv2.findContours(fg_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    #at first, no fall is detected
    fall_detected = False

    if contours:
        #finds the largest contour (assuming it is the person) --> get bounding box of it
        largest_contour = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(largest_contour)

        #double-checking : if the bounding box is large enough to be considered a human (if not, go next frame)
        if w >= min_width and h >= min_height: #min_width and min_height can be altered according to camera positioning?

            #gets the bounding box
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

            #calculates ratio of width and height, then use the ratio to determine 'falling' or 'standing'
            aspect_ratio = w / float(h)
            if aspect_ratio >= 0.5:
                status = "Falling"
                fall_detected = True
            else:
                status = "Standing"

            #display the status on top of the bounding box
            cv2.putText(frame, status, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (36, 255, 12), 2)
    return frame, fall_detected
```

Fig. 3. Code snippet of *detect_fall()* function

The *main()* function receives the input video and calls *detect_fall()* on each of the video frames. Through trials, it is discovered that there can be cases where up to three frames are detected falsely, e.g. a standing human object detected as 'falling' or vice versa. To mediate this problem, it is decided to use a threshold, in which an **alert** of 'human falling' is **created** only **if the human object is detected as 'falling' for as many as the threshold amount of consecutive frames**. For this program specifically, it is found that 15 is quite a decent number for the threshold.

```
def main(video):
    #gets the video input
    cap = cv2.VideoCapture(video)
    #initializes the bg substractor
    bg_subtractor = cv2.createBackgroundSubtractorMOG2(history=500, varThreshold=50, detectShadows=True)

    #if the detected human object has status of 'falling' for at least 15 consecutive frames, then it gives an alert
    #15 as threshold because there can be 'misdiagnosis' where a standing person is detected as falling in 1 or 2 frames, etc
    fall_counter = 0
    fall_threshold = 15
    fall_alerted = False

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        #detecting if the human object is falling or standing
        frame, fall_detected = detect_fall(frame, bg_subtractor)

        #updates fall counter based on detection
        if fall_detected:
            fall_counter += 1
        else:
            fall_counter = 0

        #if fall counter exceeds the threshold, set the alert
        if fall_counter >= fall_threshold and not fall_alerted:
            print("Alert: A person has fallen!")
            fall_alerted = True

        #displays the frames
        cv2.imshow(frame)
        if cv2.waitKey(30) & 0xFF == ord('q'):
            break

    #if there is no fall detected at all (for research purposes, in reality no alert should be given)
    if fall_counter < fall_threshold:
        print("No fall detected.")
    cap.release()
    cv2.destroyAllWindows()
```

Fig. 4. Code snippet of *main()* function

The full code can be analyzed more closely in this link : https://colab.research.google.com/drive/1v7BmPFZJDUJsXlNOqyGt_VZHvuXuVMxT?usp=sharing.

E. *Project Assumptions/Limitations*

- The program is unfortunately only able to carefully detect **one** object that has the biggest contour within each frame. As such, the visual tracking works as desired if only a single moving human object is within the entire video.
- As of the currently, other objects (if smaller than humans) will not be detected.
- Background modelling will best work if the background is quite serene and static, as opposed to having lots of moving things as well.

V. Results and Discussions

In this paper, trial videos are not discussed. For the three main testing videos, two shows test subjects that fall and one video where there is no human that falls.

A. *Video Input 1 : Student Irfan Falling*

The first video is of a student tester named Irfan who falls after walking. In fig. 5, the program correctly detects that Irfan is still standing (while walking) even though slightly crouched, and then detects the actual 'fall' later in the video. For this video input, there are some mislabeling where Irfan is considered as 'falling' even though he is standing. However, that error is irrelevant due to the threshold that is previously set; the object must be detected as 'falling' for 15 consecutive frames to have the 'alert' appearing.

Fig. 5. Example frame where the human object is detected as 'STANDING'

Fig. 6 and 7 shows, in order, the first and last frames out of the 15 consecutive frame sequence where Irfan is labeled as 'falling'. As the threshold of 15 is reached, the alert then comes up, as seen in Fig. 8.

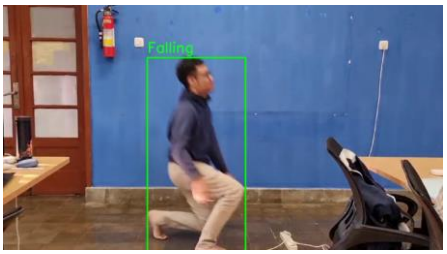

Fig. 6. The first frame out of the 15 consecutive frames where the human is detected as ‘FALLING’.

Fig. 7. The last frame out of 15 consecutive frames where the human is detected as ‘FALLING’

```
Alert: A person has fallen!
```

Fig. 8. The alert that shows up after the frame in Fig. 7.

B. *Video Input 2 : Student Hapiz Falling*

Similar to video input 1, video input 2 shows a different student tester named Hapiz who falls after walking. Fig. 9 displays the program correctly detecting that Hapiz is still standing upright and then finally detecting the actual ‘fall’ after meeting the threshold requirement.

Fig. 9. Example frame where the human is detected as ‘STANDING’

Fig. 10. The first frame out of the 15 consecutive frames where the human is detected as ‘FALLING’.

Fig. 11. The last frame out of 15 consecutive frames where the human is detected as ‘FALLING’

```
Alert: A person has fallen!
```

Fig. 12. The alert that shows up after the frame in Fig. 11.

C. *Video Input 3 : Student Hapiz Walking*

Unlike previous input videos,video 3 records student tester Hapiz walking and turning back but never falling even once. The program successfully detects that Hapiz is ‘standing’ for all frames, giving the result “No fall detected”.

Fig. 13. Example frame where the human object is detected as ‘STANDING’

Fig. 14. Example frame after the detected human object shifts their body

```
No fall detected.
```

Fig. 15. The ‘informative alert’ that shows up if there are no more than 15 consecutive frames where object is labeled ‘FALLING’ in the entire video

D. *Improvements*

As discussed before on project limitations, the proposed method is not inherently capable of detecting multiple human objects simultaneously. Nevertheless, with more research time and additional processing, it is possible to extend these methods to detect multiple objects. The methods may involve techniques such as foreground segmentation to separate and identify multiple foreground objects, object tracking to track multiple objects over time, and multi-model backgrounds handle dynamic scenes with multiple stationary components. Deep learning, such as the use of YOLOv8 or other CNN methods may also handle this special matter in much better way.

ACKNOWLEDGMENT

The author sincerely thanks her lecturer, Mr. Wahyono, for his passionate teachings and guidance in topics of computer vision and image processing, which made this project possible.

REFERENCES

[1] AyaData. “Bounding Boxes in Computer Vision: Uses, Best Practices for Labeling, and More.” *Aya Data*, 21 Mar. 2024, www.ayadata.ai/blog-posts/bounding-boxes-in-computer-vision-uses-best-practices-for-labeling-and-more/.

[2] "Fall Detection and Fall Prevention." *Minimize the Number of Falls*, www.sensio.io/services/fall-alarm#:~:text=Why%20is%20it%20important%20to,in%20the%20worst%20case%2C%20death. Accessed 19 May 2024.

[3] "Goals." *OpenCV*, docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html. Accessed 19 May 2024.

[4] Sabih, Muhammad. "Background Subtraction in Computer Vision." *Medium*, Medium, 11 June 2022, medium.com/@muhammadsabih56/background-subtraction-in-computer-vision-402ddc79cb1b.