Development and Implementation of a Fall Detection application in Android

Jacopo Carlon*, Nicola Riccardi*
*University of Pisa
j.carlon@studenti.unipi.it, n.riccardi@studenti.unipi.it

***Abstract*—Falls are the leading cause of injury for adults ages 65 years and older [1]. Obviously, a timely and accurate detection of such falls is of paramount importance, since it could lead to prompt intervention, reducing injuries and possibly saving lives. We hereby present "fallenPeopleDetector", an Android application built in Android Studio. The main functionality of this app is to start a service that, once started, will keep running without further user interaction. We will periodically acquire measurements from the mobile device's accelerometer, upon detecting a fall we will generate a loud signal, in hope of alerting nearby people that might help. In our implementation we chose to prioritize the minimization of False Negatives, at the cost of possible false positives, as will be in more detail explained. This paper covers both the decision algorithm and the philosophy behind our personal choices, as well as some implementation choices made during development.**

***Index Terms*—Fall Detection - Accelerometer - Android - Mobile Application Development - Kotlin Java**

I. Introduction

Considering how the life expectancy has increased in the past decades, the increase in elderly population has motivated many healthcare researches and technologies. Considering how very often elderly people live alone, the impact of a fall (which already is the leading cause of injury) can be critical. In order to minimize the after-effects of a fall (and thus possibly saving lives), much scholarly work has been done on fall-detection. Fall detection analysis can generally be of two types: environmental and wearable. Considering how environmental analysis (i.e. detection via cameras) could fail due to existence of blind spots, and how it is quite invasive of the person's privacy, we will only focus on the "wearable" analysis. Many devices could be used in this direction, be it specific shoes, wristbands and similar.

However, considering how commonly used are mobile phones nowadays, and how the coming generations will be most likely familiarized with them, a simple measuring device could be the phone itself, by means of its built-in accelerometer .

In the most intuitive form of accelerometer-fall-analysis, we can assume to have information about 3-axis acceleration of a moving object. In the hypothesis that we are situated about around the surface level of the earth, there will be a constant input due to the gravity. The gravity vector will be projected along the axis depending on the instrument orientation, but considering that any simple rotation of the device would change its projection, and considering that it is ever present, it should be simplified in our analysis.

Other than gravity, if the accelerometer is on a mobile phone (that is the case of study), it can clearly be subject to many spurious events, ranging for example from when you answer to a call and move the phone closer to your ear, when the holder sits, jumps or run, or when the device falls.

In all the instances of "free motion", developing an algorithm able to detect user fall would be extremely hard, due to the overwhelming presence of spurious events (e.g. being in a car that makes a turn could taint the analysis), and many physiological details could come into play (e.g. height, arm/leg length).

If, however, we focus on analyzing the behavior when the device is resting our our pocket, we already remove many "variables". In our implementation, we followed the analysis works by Lee-Tseng [2], and our target use case is that of an android phone which starts the detection Service [3], and then is placed in the owner's pocket.

II. Algorithm Analysis

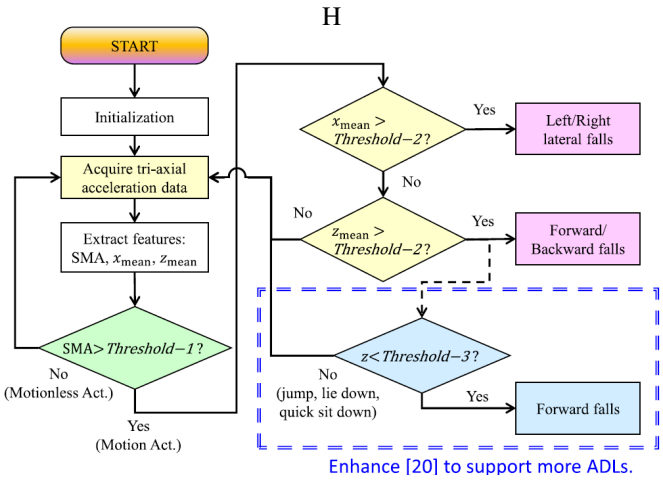

Fig. 1. Fall Based detection approach as described in [2]

A. Target Device

This paper will focus on Android-based mobile-phones. The accelerometer (oriented as in Fig.2 will return values along its

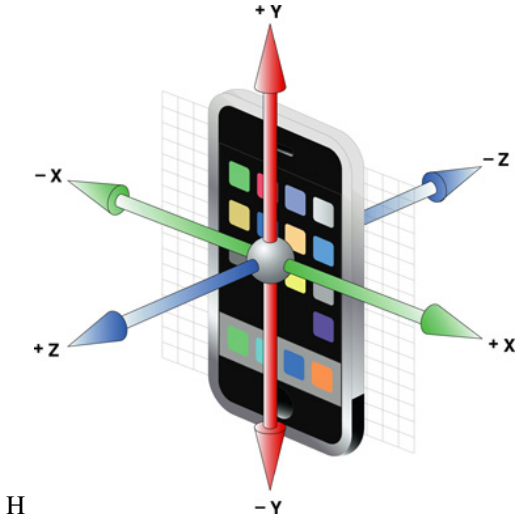

Fig. 2. Three axes of the accelerometer.

3 axis, with a set sampling rate of 50 Hz (i.e. one acquisition every 20ms). It should be noted that the sampling rate is not always respected by the actual device, but that does not present a problem, as will be explained later.
In Android, instead of taking the raw values of the accelerometer, we used the sensor "TYPE_LINEAR_ACCELERATOR" [4], which removes the influence of gravity from the values.

B. Motion Activity : SMA vs Threshold_1

With a simple ideal analysis, during a free fall, the device center of mass would accelerate following the gravity, and therefore the apparent acceleration perceived by the accelerator would be null (except for rotation effects). However, if we consider the linear_accelerator, which is at default 0 if the phone is on a table, during a free fall it would register g (projected along the different axis).
In our analysis, we will use the average of the acceleration intensity value SMA, (with a target windows of 5 samples) as a way to register a movement of the device.

$$SMA[n] = \frac{1}{N}\Big(\sum_{i=n-N+1}^{n} |x[i]| + \sum_{i=n-N+1}^{n} |y[i]| + \sum_{i=n-N+1}^{n} |z[i]|\Big) \quad (1)$$

This SMA value will be tested against a "Threshold_1" of $27\,\mathrm{ms}^{-2}$, following experimental observations described in [2]. If SMA is smaller than Threshold_1, than we will exclude the possibility of a fall, otherwise we will continue with the analysis.

C. Lateral Fall Detection : x_mean vs Threshold_2

The paper [2] describes how the mean absolute value of acceleration along x-axis can be related to a lateral fall, since they work under the hypothesis that the smartphone be placed in the front pocket. Such mean of acceleration is extracted as follows:

$$\mathrm{x}_{mean}[n] = \frac{1}{N}\Big(\sum_{i=n-N+1}^{n} |x[i]|\Big) \quad (2)$$

x_mean is then tested against a "Threshold_2" of $10.05\,\mathrm{ms}^{-2}$ (again, a number extracted from experimental observations). If the mean of absolute accelerations over x is greater than Threshold_2, then a Lateral-Fall is detected.

D. Forwards-Backwards Fall Detection : z_mean vs Threshold_2

The same mean can be extracted from the values of the z-axis, and then compared against Threshold_2.
While both a Forwards and a Backwards fall would cause this check to be triggered, the paper observes how other events along the z-axis could also cause spikes in z acceleration, for example jumping or sitting down quickly.
For this reason, the paper proposes a new analysis of the z-axis acceleration, that follows.

E. Forwards-Backwards Fall Detection : z_mean vs Threshold_2

If the phone is on the front pocket of the user, then the instantaneous signed acceleration values on the z-axis have a negative spike on a forward fall, and not during a jump or a quick sit, while the mean of absolute z-axis value is above Threshold_2 for all 3 cases.
This observation led the authors of [2] to suggest the use of a new "Threshold_3" of $-10\,\mathrm{ms}^{-2}$.
While the other thresholds are used with means of sums of absolute values, and are thus positive, this new threshold is negative since it is used against the signed value register on the z-axis. If the value on z is less than Threshold_3, than a Forwards fall is detected.
Sadly, this analysis cannot be extended to separate a backwards fall from jumps and the other spurious activities.

III. Design Choices

When choosing how to implement the application, we decided that minimizing False Negatives (i.e. instances of actual falls that we might no detect) was of paramount importance.

A. z-axis decisions

While correctly detecting events of Lateral and Forwards Fall, the algorithm was not able to distinguish between Backwards Fall and some other "innocuous" activities like jumping or a quick sit-down (although these are not exactly common amongst elderly people).

In order to not miss any fall, we have decided to consider as backwards fall also these activities, meaning that any event that causes the mean absolute average of z-axis to be bigger than Threshold_2 will in fact be detected as a fall.

B. Linearization

Despite having set the accelerometer at 50Hz sampling, sometimes some inputs could be missed. In order to properly evaluate the averages described at 1 and II-C, we "fill those gaps" by populating the local ArrayDeques corresponding to each acceleration with values that are result of linear interpolation between the last value received and the new value (weighted over the respective times).
The "targetValue" (value at a target time) is obtained as follows (value V, time T):

$$trgV = oldV + (newV - oldV) * \frac{trgT - OldT}{NewT - OldT} \quad (3)$$

which is simply obtained from the *parametric form of a Straight Line* :

$$\mathrm{y} = m * x + q \quad (4)$$

and of its *slope* m :

$$\mathrm{m} = \frac{y\_end - y\_start}{x\_end - x\_start} \quad (5)$$

IV. Architecture

The application consists of three main components:

- **MainActivity**: gives an UI to start the fall detection service.
- **AccelerometerService**: listens to the accelerometer and fires up a notification in case of detected fall
- **AlarmActivity**: manages the alarm after a fall is detected

A. Starting the Service

When the user starts the app, **MainActivity** shows a brief description of the service provided and a *switch button* to activate it.

If the user checks the switch, **MainActivity** will first determine if the app has the necessary permissions:

- `POST_NOTIFICATIONS` (only for API level higher or equal to 33), needed to send any kind of notification
- `USE_FULL_SCREEN_INTENT` (only for API level higher or equal to 29), needed to associate an urgent notification with a *FullScreenIntent*, i.e. an Activity that wakes up the phone and occupies all the screen.

Once all permissions are granted to the application, checking the *switch button* will start **AccelerometerService**, that will continue working in the background until the user either un-checks the *switch* or closes the app completely.

B. Fall Detection

When it's started, **AccelerometerService** registers an handler thread as listener to `LINEAR_ACCELERATION` sensor, to process the data in the background, and creates a `NotificationChannel` for sending the notification once a fall is detected. When receiving an update of sensor status, it is processed as described in previous sections, using a support class called **StatusSetector**.

If the **StatusDetector** registers a fall event, **AccelerometerService** builds an high-priority notification associated with a pending `FullScreenIntent` for **AlarmActivity**. If the necessary permissions were not granted, the notification cannot be sent and an error will be logged.

C. Alarm

Once the notification is sent, **AlarmActivity** is created and starts a `MediaPlayer` to play an alarm. To ensure that the alarm can be heard, the volume is forced to its maximum value by using the interface provided by `AudioManager` and the sound track is set to loop. **AlarmActivity** also displays a simple UI to stop the alarm.

It has to be noted that when the user is actively using the phone, Android can decide to not show a full-screen intent associated to an urgent notification, preventing therefore the alarm from playing. This occurrence, however, was not considered for this simple implementation, since the user is assumed to have the phone in his pocket while making use of the provided service.

V. TESTING

Testing the application properly would prove to be difficult, since it would require a real fall. However, since the detection algorithm is very simple, we found some specific movements that can trick it into detecting a fall without a real one happening.

These movements aim to generate a strong deceleration, similar to an impact on the ground. An example is straightening the arm abruptly while holding the phone in hand.

While this kind of movement is not totally unnatural, it is very infrequent and is outside of the assumption of the phone being in a pocket, so the occurrences of false positives caused by that are negligible.

The application was tested on the following devices:

- Xiaomi Redmi 11S (Android 13)
- Huawei EML-L09 (Emui 12.0.0 - Android 10)

VI. Conclusion

The application developed, despite being somewhat situational, is definitely a move in the right direction towards a more rapid response to possibly life-threatening events.

A fitting feature would be an automatic call to emergency service, or an otherwise set emergency number. Considering our policy of no False Negatives, this feature would inevitably risk many false alarms, and such choice should be carefully weighted.

Another thing of note is that, although the accelerometer uses 10 times less power than any other sensor [5], we are still running it in background all the time the service is active. When implementing this application, we evaluated this energy cost to be an acceptable trade-off, compared to the possible reduction in response time to a real life fall event, possibly saving lives.

VII. Additional Notes

All the code is available on GitHub at : https://github.com/nickrick3/FallenPersonDetection [6], alongside the main reference paper [2] for this work, and this paper itself.

References

1 ”older adults fall data”. [Online]. Available: https://www.cdc.gov/falls/data-research/index.html

2 Lee, J.-S. and Tseng, H.-H., “Development of an enhanced threshold-based fall detection system using smartphones with built-in accelerometers,” *IEEE Sensors Journal*, vol. 19, no. 18, pp. 8293–8302, 2019.

3 Service — android developers. [Online]. Available: https://developer.android.com/reference/android/app/Service

4 Linearacceleration — android developers. [Online]. Available: https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion#sensors-motion-linear

5 Motion sensors — sensors and location — android developers. [Online]. Available: https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion

6 Nicola Riccardi, Jacopo Carlon. ”Fallen Person Detection — Mobile and Social Sensing Systems Project”. [Online]. Available: https://github.com/nickrick3/FallenPersonDetection