Elderly Fall Detection using Wearable Sensors: A Low Cost Highly Accurate Algorithm

Majd SALEH and Régine LE BOUQUIN JEANNÈS

***Abstract*—Every year, more than 37 million falls that require medical attention occur. The elderly suffer the greatest number of fatal falls. Therefore, automatic fall detection for elderly is one of the most important health-care applications as it enables timely medical intervention. The fall detection problem has extensively been studied over the last decade. However, since the hardware resources of wearable devices are limited, designing highly accurate embeddable algorithms with feasible computational cost is still an open research challenge. In this paper, a low-cost highly-accurate machine learning-based fall detection algorithm is proposed. Particularly, a novel online feature extraction method that efficiently employs the time characteristics of falls is proposed. In addition, a novel design of a machine learning-based system is proposed to achieve the best accuracy/numerical complexity trade-off. The low computational cost of the proposed algorithm not only enables to embed it in a wearable sensor but also makes the power requirements quite low and hence enhances the autonomy of the wearable device where the need for battery recharge/replace is minimized. Experimental results on a large open dataset show that the accuracy of the proposed algorithm exceeds 99.9% with a computational cost of less than 500 floating point operations per second.**

***Index Terms*—fall detector, wearable sensor, low power consumption, online feature extraction, machine learning.**

I. Introduction

EVERY year, more than 37 million falls that require medical attention occur [1]. The elderly suffer the greatest number of fatal falls [1]. Therefore, automatic fall detection for elderly is one of the most important health-care applications as it enables timely medical intervention. Fall detectors can be divided into wearable and ambient devices [2], [3]. Wearable fall detectors have the advantage that they accompany the elderly indoors and outdoors contrary to the ambient devices, including computer vision-based ones, which are restricted to the indoor usage.

Wearable fall detectors are responsible for sending an alarm to some authorized team as soon as a fall is detected. The geographical position of the elderly is sent as well. These devices acquire the activity of the elderly using one sensor or a combination of the following sensors: accelerometer, gyroscope, magnetometer and barometer. For instance, the algorithm proposed in [4] estimates the orientation of the sensor frame with respect to the earth frame using Madgwick filter [5] which fuses the measurements of accelerometer, gyroscope and magnetometer. The estimated orientation is used to extract the dynamic vertical acceleration which is then fused with barometer measurements in order to estimate the altitude of the elderly where the latter is thresholded to detect falls. Similarly, the algorithm proposed in [6] estimates the orientation of the body by fusing the gyroscope and accelerometer measurements using extended Kalman filter. The extracted dynamic vertical acceleration is fused with the measurements of barometer in order to evaluate the vertical velocity and the height of the elderly where the last two quantities are used to detect falls in a threshold based method. However, a multitude of researchers focused on the acceleration-based solutions where the interest of using only a 3-axial accelerometer is to decrease the power consumption of the fall detector which results in increasing the autonomy of this wearable device and thus increasing its acceptability by the elderly. In addition, decreasing the power consumption enables to use small batteries and thus to design a small and light-weight devices [7]. It is worth mentioning that the power consumption of a gyroscope is tens of times greater than the consumption of an accelerometer [7]. Therefore, the acceleration-based solutions are more appropriate for the wearable devices than the ones that combine multiple sensors like [4] and [6].

Indeed, solving the activity recognition problem regardless the required computational complexity has almost perfectly been achieved in the literature in the context of fall detection as well as in the relevant context of human activity recognition thanks to elegant deep learning techniques [8]–[10]. However, the problem of designing highly accurate embeddable algorithms with feasible computational cost is still an open research challenge since the hardware resources of wearable devices are limited. In order to reduce the computational load on wearable fall detectors, several solutions were proposed in the literature. These solutions as well as their limitations and drawbacks are discussed hereafter. Note that only acceleration-based fall detectors are considered in this article.

The most widely used solution is to design simple threshold-based fall detection algorithms. These algorithms compare the acceleration signals on x, y and z axes (or some quantity derived from these signals) with predefined threshold(s) [11]–[14]. Some of them also use time thresholds to take advantage of the time characteristics of falls [12], [13]. However, the low computational cost of threshold-based algorithms is at the expense of the accuracy [15]–[18], i.e. a huge number of false alarms are generated by these algorithms if the threshold is too low and, on the other hand, a considerable number of falls will not be detected if the threshold is too high. In order to overcome the accuracy limitations of threshold-based algorithms, a multitude of machine learning-based algorithms

M. SALEH and R. LE BOUQUIN JEANNÈS are with INSERM U1099, Rennes, F-35000, France and with Université de Rennes 1, LTSI, Rennes, F-35000, France. e-mail: (majd.saleh@univ-rennes1.fr, regine.le-bouquin-jeannes@univ-rennes1.fr).

were proposed in the literature. Although these algorithms are more accurate, they are more computationally demanding and implementing them in wearable devices is a challenge. One solution used to tackle this problem is to avoid embedding the machine-learning based algorithm on the wearable device itself but on a base-station instead. Raw data (or preprocessed data) are sent via some wireless link from the wearable device to the base station where these data are processed to detect falls [17], [19]. This solution enables to use powerful and complicated machine learning-based methods such as deep neural networks [8]. However, the latter solution is not appropriate for outdoor environments as the distance between the wearable device and the base-station is limited in the considered technologies e.g. *ZigBee* in [17] and *Bluetooth* in [8], [19]. Therefore, developing embeddable machine learning-based algorithms is mandatory to achieve an accurate wearable fall detector that could work in both indoor and outdoor environments. In order to reduce the computational cost of such algorithms, researchers focused on reducing the dimensionality of the features vector using e.g. Principal Component Analysis (PCA) and/or Independent Component Analysis (ICA) [20], [21]. Given that the computational cost of a trained machine is proportional to the dimensionality of the input features vector, this solution reduces the complexity of the embedded algorithm. However, the computational cost of calculating the features vector could be several times greater than the trained machine one. This claim is proven later in this paper. In addition, the aforementioned solution requires a preprocessing to multiply the features vector by some parameter matrix after each features extraction step. This multiplication itself is computationally demanding.
In the present paper, a low computational-cost highly-accurate machine learning-based fall detection algorithm is proposed where the activity of the elderly is captured using a 3-axial accelerometer. It overcomes the aforementioned drawbacks and limitations thanks to the following three novel contributions: 1) a two-segment feature extraction method that efficiently employs the time characteristics of falls, 2) an online method that calculates the features with low computational cost, and 3) a design of a machine learning-based system that satisfies the best accuracy/complexity trade-off.

This paper is organized as follows: the proposed fall detection algorithm is explained in Section II. The computational complexity of the proposed algorithm is provided in Section III. Section IV shows the experimental results evaluated on a large open dataset. Finally, Section V concludes the paper and discusses some topics to be explored in future work.

II. The Proposed fall detection algorithm

The proposed fall detection algorithm uses acceleration signals acquired by a triaxial accelerometer mounted to the waist of the elderly. It extracts features from a sliding time window and presents them to a trained machine that, in turn, makes a binary decision: 0 for the absence or 1 for the presence of a fall. The proposed features to be extracted from the triaxial acceleration signal as well as the proposed online feature extraction method are explored in the next subsection. Then, the proposed SVM-based fall detection method is explained before encapsulating the full fall detection algorithm and presenting it in a stand alone pseudo-code.

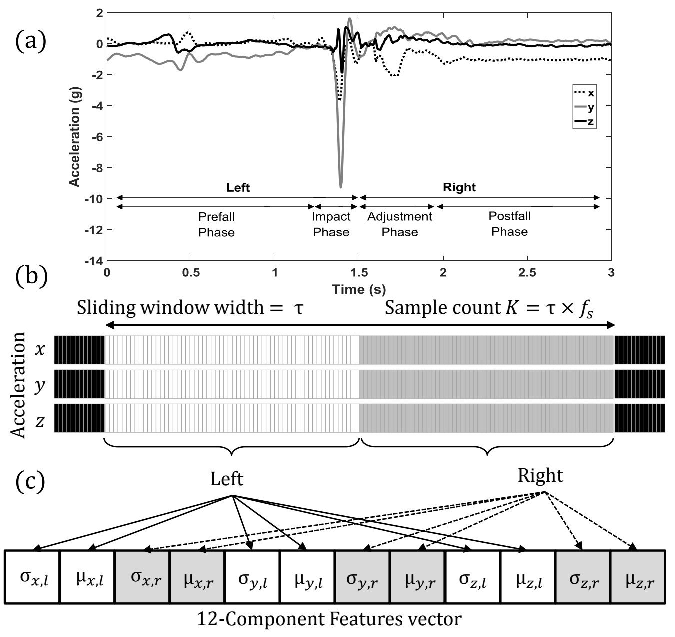


Fig. 1. The proposed features to be extracted from a triaxial acceleration signal. (a) an observed fall signal within a 3-second sliding window, (b) dividing the sliding window into to equal-length parts and (c) building a 12-component features vector.

A. The Proposed Features and Feature Extraction Method

Unlike the conventional machine learning-based fall detection algorithms e.g. [15], [16], [19], [21]–[24], the proposed algorithm divides the sliding window into two equal segments noted left and right, as shown in Figure 1.a and 1.b. Then, the 1st and 2nd order moments of both segments are extracted resulting in a 12-component features vector $\boldsymbol{f} = [\sigma_{x,l}\ \mu_{x,l}\ \sigma_{x,r}\ \mu_{x,r}\ \sigma_{y,l}\ \mu_{y,l}\ \sigma_{y,r}\ \mu_{y,r}\ \sigma_{z,l}\ \mu_{z,l}\ \sigma_{z,r}\ \mu_{z,r}]$ as illustrated in Figure 1.c. The importance of such two-segment features comes from the time characteristics of falls. A fall could be characterized by 4 phases that are prefall, impact, body adjustment and postfall phases. These phases are illustrated in Figure 1.a.
Indeed, when statistical features are extracted from the whole sliding window, some ADL and fall types could show similar statistics. Figure 2 shows an example, where a fall (backward fall while sitting, caused by fainting) is illustrated in Figure 2.a and an ADL (gently jumping trying to reach a high object) is illustrated in Figure 2.b. The standard deviations (STDs) of x, y and z accelerations of the illustrated fall and ADL are $[0.137,\ 0.666,\ 0.278]$ and $[0.131,\ 0.669,\ 0.271]$, respectively as shown in Figure 2.c. They are very similar which means that this widely used feature, i.e. the standard deviation, is not able to discriminate between some falls and ADLs when it is extracted as a global feature from a sliding window. The aforementioned ambiguity between ADL and fall features is expected to decrease considerably using the proposed two-segment feature extraction method as illustrated

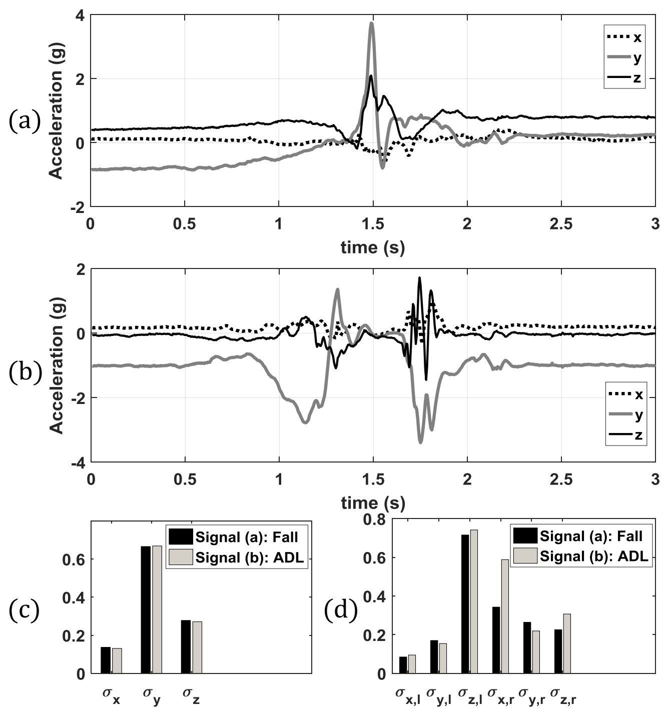


Fig. 2. Triaxial acceleration signals of a Fall (a) and ADL (b) during a 3-second sliding window. The considered feature, i.e. the STD, is illustrated in (c) where it is globally extracted from the whole sliding window, and in (d) where it is extracted from the left and right segments of the sliding window.

in the aforementioned example in Figure 2.d. Indeed, as the STD is proportional to the power of the signal, it is expected to have higher STD in the prefall and impact phases (left segment) than in the adjustment and postfall phases (right segment) on one acceleration axis at least. This is because the subject moves slightly in the adjustment phase and he/she is almost stationary in the postfall phase, while the prefall phase normally contains some activity and the impact phase contains a considerable variation in acceleration. Following the aforementioned logic of segmentation, one could be motivated to segment the sliding window into four segments corresponding to the four phases of a fall. However, as the lengths of the impact and the body-adjustment phases depend on the type of fall, the proposed method is safer as it guarantees the presence of the first two phases in the left segment and the last two phases in the right one when using an adequate window length. One key factor to make the fall detection algorithm embeddable is to calculate features with low computational cost. To this end, a novel online feature extraction method is proposed. This method is explained hereafter.

Let $T_e = \kappa \times T_s$ be the feature extraction period where $\kappa \in \{1, 2, 3, \ldots, \kappa_{max}\}$ and T_s denotes the accelerometer sampling period. Features are extracted from a sliding window of width $\tau = \lambda \times T_e$ where $\lambda \in \{2, 4, 6, \ldots, \lambda_{max}\}$. The number of samples in the sliding window (per one acceleration axis) is $K = \tau/T_s$. $f_s = 1/T_s$ and $f_e = 1/T_e$ denote the sampling and feature extraction frequencies, respectively. Note that even if the logic of the proposed algorithm is to use a sliding time window of width τ, the full history of the acceleration signal over τ seconds will not be stored but just the following buffers are needed instead: $\{\boldsymbol{l}^{(\rho)}, \boldsymbol{s}^{(\rho)}, \boldsymbol{\sigma}^{(\rho)}, \boldsymbol{\mu}^{(\rho)}, r_1^{(\rho)}, r_2^{(\rho)}\} \in \mathbb{R}^\delta \times \mathbb{R}^\delta \times \mathbb{R}^\delta \times \mathbb{R}^\delta \times \mathbb{R} \times \mathbb{R}, \forall \rho \in \{x, y, z\}$, where $\delta = \lambda/2 + 1$. The aforementioned buffers are first initialized by zeros. Buffers $\boldsymbol{l}^{(\rho)}$ and $\boldsymbol{s}^{(\rho)}$ are updated as soon as a new acceleration sample a_ρ is acquired. Particularly, their last components are updated using the following equations, respectively:

$$l_\delta^{(\rho)} \longleftarrow l_\delta^{(\rho)} + a_\rho \tag{1}$$

$$s_\delta^{(\rho)} \longleftarrow s_\delta^{(\rho)} + a_\rho^2 \tag{2}$$

After κ updates of the aforementioned buffers, the features vector is updated according to the following steps. $r_1^{(\rho)}$ and $r_2^{(\rho)}$ are updated using the first and last components of $\boldsymbol{l}^{(\rho)}$ and $\boldsymbol{s}^{(\rho)}$, respectively, using the following equations:

$$r_1^{(\rho)} \longleftarrow r_1^{(\rho)} + l_\delta^{(\rho)} - l_1^{(\rho)} \tag{3}$$

$$r_2^{(\rho)} \longleftarrow r_2^{(\rho)} + s_\delta^{(\rho)} - s_1^{(\rho)} \tag{4}$$

Using $r_1^{(\rho)}$ and $r_2^{(\rho)}$, the means and standard deviations of the right part of the sliding window are calculated and stored in buffers $\boldsymbol{\mu}^{(\rho)}$ and $\boldsymbol{\sigma}^{(\rho)}$ using the following equations:

$$\mu_\delta^{(\rho)} = 2r_1^{(\rho)}/K \tag{5}$$

$$\sigma_\delta^{(\rho)} = \sqrt{(r_2^{(\rho)} - 2\frac{(r_1^{(\rho)})^2}{K})/(\frac{K}{2} - 1)} \tag{6}$$

The same features for the left part of the sliding window are just extracted from the first components of buffers $\boldsymbol{\mu}^{(\rho)}$ and $\boldsymbol{\sigma}^{(\rho)}$ as these features have already been calculated before $(\delta - 1) \times T_e$ seconds. As soon as the features vector is built, buffers $\boldsymbol{l}^{(\rho)}, \boldsymbol{s}^{\rho}, \boldsymbol{\mu}^{(\rho)}$ and $\boldsymbol{\sigma}^{(\rho)}$ are shifted left with zero insertion at the last component e.g. $l_i^{(\rho)} \leftarrow l_{i+1}^{(\rho)}\ \forall i \in \{1, \delta - 1\}$ and $l_\delta^{(\rho)} \leftarrow 0$. Figure 3 illustrates the proposed feature extraction method for $\kappa = 12, \lambda = 10, f_s = 40$ Hz and $\rho = x$. Another online way to calculate mean and STD features would be based on Welford algorithm [25] with some modification to adapt it for dealing with sliding windows. However, it is more demanding regarding computational complexity and storage space than the method already proposed in this section. Appendix A shows the online update rules for calculating mean and STD features using Welford algorithm.

B. The Proposed Machine Learning-Based System for detecting falls

In this section, a brief background on SVM is introduced in the context of fall detection. Then, the proposed SVM-based fall detection algorithm is explored.

1) Background on support vector machine: Let $\boldsymbol{f} = [f_1, f_2, \ldots, f_N]^\intercal$ be an N dimensional vector representing the extracted features. For instance, in the current work $\boldsymbol{f}$ represents the 12-component features vector introduced in the previous subsection where the features are statistical quantities extracted from the 3-axis acceleration signals. The features vector $\boldsymbol{f}$ represents the input of the SVM while the output is a binary value (e.g. 1 for fall and 0 for ADL) which represents the class of $\boldsymbol{f}$ and consequently the class of the activity from which $\boldsymbol{f}$ has been extracted.

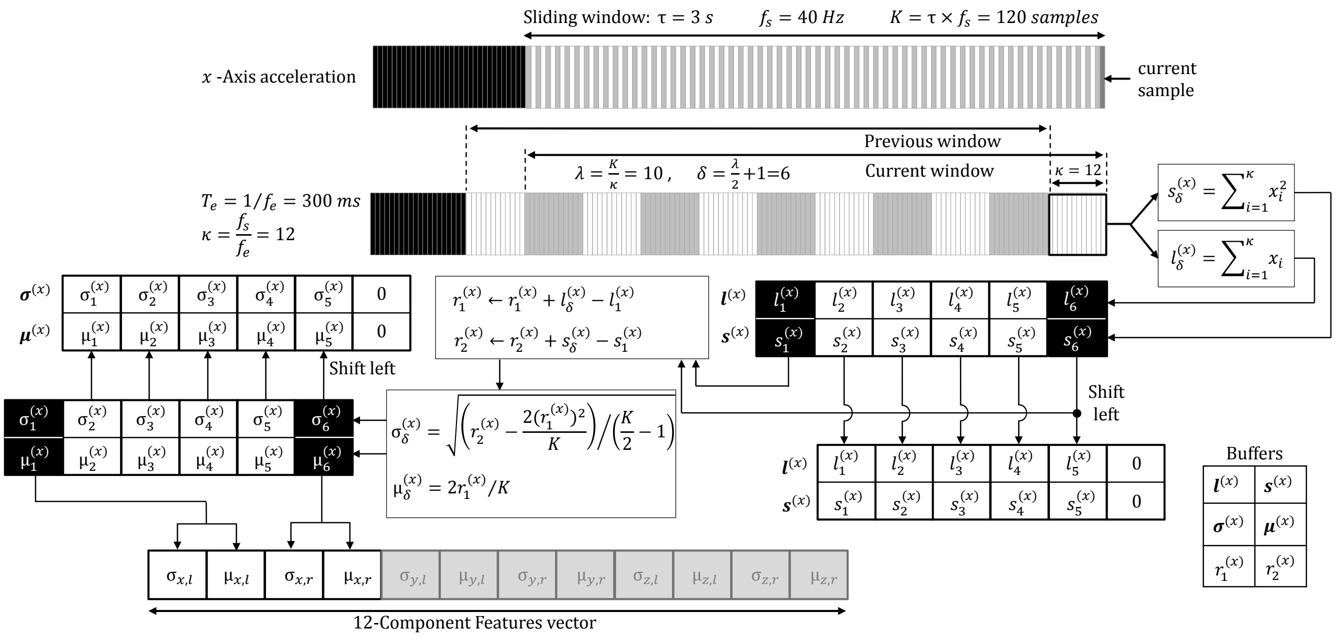


Fig. 3. The proposed online feature extraction method

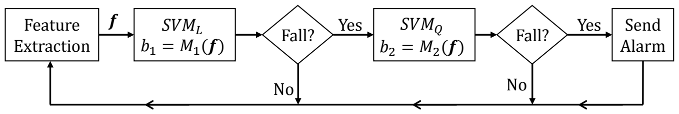


Fig. 4. The proposed machine learning-based system for detecting falls

Algorithm 1

```
Input: $\kappa \in \{1, 2, 3, \ldots, \kappa_{max}\}$ and $\lambda \in \{2, 4, 6, \ldots, \lambda_{max}\}$;
Calculate: $\delta \leftarrow \frac{\lambda}{2} + 1$; do the following steps $\forall \rho \in \{x, y, z\}$:
Initialize: $\boldsymbol{l}^{(\rho)} \leftarrow \boldsymbol{s}^{(\rho)} \leftarrow \boldsymbol{\mu}^{(\rho)} \leftarrow \boldsymbol{\sigma}^{(\rho)} \leftarrow \mathbf{0}_{1\times\delta}$,
$r_1^\rho \leftarrow r_2^\rho \leftarrow 0$ and $i \leftarrow 0$;
While true
    • Acquire a new sample $a_\rho$ from the 3-axial accelerometer.
    • Update $l_\delta^{(\rho)}$ and $s_\delta^{(\rho)}$ using equations (1) and (2), respectively.
      $i \longleftarrow i + 1$;
    • If $i = \kappa$
        – Update $r_1^{(\rho)}$ and $r_2^{(\rho)}$ using (3) and (4), respectively.
        – Calculate $\mu_\delta^{(\rho)}$ and $\sigma_\delta^{(\rho)}$ using (5) and (6), respectively.
        – Build features vector:
          $\boldsymbol{f} \leftarrow [\sigma_1^x\ \mu_1^x\ \sigma_\delta^x\ \mu_\delta^x\ \sigma_1^y\ \mu_1^y\ \sigma_\delta^y\ \mu_\delta^y\ \sigma_1^z\ \mu_1^z\ \sigma_\delta^z\ \mu_\delta^z]$;
        – $b_1 \leftarrow M_1(\boldsymbol{f})$ where $M_1$ is given by equation (12).
        – If $b_1 = 1$ //Fall has been detected
            * $b_2 \leftarrow M_2(\boldsymbol{f})$ where $M_2$ is given by equation (15).
            * If $b_2 = 1$ //Fall has been confirmed
                · Send an alarm to some authorized team.
            * End
          End
        – Shift left $\boldsymbol{l}^{(\rho)}, \boldsymbol{s}^{(\rho)}, \boldsymbol{\mu}^{(\rho)}$ and $\boldsymbol{\sigma}^{(\rho)}$.
          $i \leftarrow 0$;
      End
End
```

Algorithm 2

```
Input: $\kappa \in \{1, 2, 3, \ldots, \kappa_{max}\}$ and $\lambda \in \{2, 4, 6, \ldots, \lambda_{max}\}$;
Calculate: $\delta \leftarrow \frac{\lambda}{2} + 1$; do the following steps $\forall \rho \in \{x, y, z\}$:
Initialize: $\boldsymbol{l}^{(\rho)} \leftarrow \boldsymbol{\mu}^{(\rho)} \leftarrow \boldsymbol{g}^{(\rho)} \leftarrow \mathbf{0}_{1\times\delta}$
$U^{(\rho)} \leftarrow L^{(\rho)} \leftarrow \tilde{U}^{(\rho)} \leftarrow \tilde{L}^{(\rho)} \leftarrow$ empty queue.
$r_1^\rho \leftarrow r_2^\rho \leftarrow 0$ and $i \leftarrow 0$;
While true
    • Acquire a new sample $a_\rho$ from the 3-axial accelerometer.
    • Update $l_\delta^{(\rho)}$ using (1).
      Update $U^{(\rho)}$ and $L^{(\rho)}$ as in [26] using a window width of $w = \lambda \times \kappa/2$ samples.
      Assign acceleration values corresponding to new items in $U^{(\rho)}$ and $L^{(\rho)}$ to $\tilde{U}^{(\rho)}$ and $\tilde{L}^{(\rho)}$ respectively.
      $i \longleftarrow i + 1$;
    • If $i = \kappa$
        – Update $r_1^{(\rho)}$ using (3).
          Update $r_2^{(\rho)} \leftarrow \tilde{U}^{(\rho)}(1) - \tilde{L}^{(\rho)}(1)$
        – Calculate $\mu_\delta^{(\rho)}$ using (5).
          $g_\delta^{(\rho)} \leftarrow r_2^{(\rho)}$;
        – Build features vector:
          $\boldsymbol{f} \leftarrow [g_1^x\ \mu_1^x\ g_\delta^x\ \mu_\delta^x\ g_1^y\ \mu_1^y\ g_\delta^y\ \mu_\delta^y\ g_1^z\ \mu_1^z\ g_\delta^z\ \mu_\delta^z]$;
        – $b_1 \leftarrow M_1(\boldsymbol{f})$ where $M_1$ is given by equation (12).
        – If $b_1 = 1$ //Fall has been detected
            * $b_2 \leftarrow M_2(\boldsymbol{f})$ where $M_2$ is given by equation (15).
            * If $b_2 = 1$ //Fall has been confirmed
                · Send an alarm to some authorized team.
            * End
          End
        – Shift left $\boldsymbol{l}^{(\rho)}, \boldsymbol{\mu}^{(\rho)}$ and $\boldsymbol{g}^{(\rho)}$.
          $i \leftarrow 0$;
      End
End
```

SVM calculates the class of $\boldsymbol{f}$ as follows [27], [28]:

$$\text{class}(\boldsymbol{f}) = \text{sign}(\hat{\mathfrak{F}}(\boldsymbol{f})) \tag{7}$$

where $\hat{\mathfrak{F}}(\boldsymbol{f})$ is given by:

$$\hat{\mathfrak{F}}(\boldsymbol{f}) = \sum_{i=1}^{L} \alpha_i \mathfrak{K}(\boldsymbol{f}, \boldsymbol{s}_i) + \hat{\beta}_0 \tag{8}$$

$\boldsymbol{s}_i = [s_{i_1}, s_{i_2}, \ldots, s_{i_N}]^\intercal, \forall i \in \{1, \ldots, L\}$, are the support vectors, L is the number of these support vectors, α_i are weights, $\mathfrak{K}(.,.)$ denotes the kernel function and $\hat{\beta}_0$ denotes the bias. The support vectors, the weights as well as the bias are set by the training process. The most widely used kernels, namely the linear, polynomial and radial-basis function (RBF) kernels are given, respectively, by:

$$\mathfrak{K}_1(\boldsymbol{f}, \boldsymbol{s}_i) = \boldsymbol{f}^\intercal \boldsymbol{s}_i \tag{9}$$

$$\mathfrak{K}_2(\boldsymbol{f}, \boldsymbol{s}_i) = (c + \boldsymbol{f}^\intercal \boldsymbol{s}_i)^d \tag{10}$$

$$\mathfrak{K}_3(\boldsymbol{f}, \boldsymbol{s}_i) = \exp \frac{-\text{dist}(\boldsymbol{f}, \boldsymbol{s}_i)}{2\sigma^2} \tag{11}$$

where $^\intercal$ denotes the transpose operator, d is the polynomial degree, c is a constant used for controlling the relative weightings between the features of degrees $(0, 1, \ldots, d)$ and dist is a function used for measuring the similarity between $\boldsymbol{f}$ and $\boldsymbol{s}_i$ (e.g. one widely used function in practice is the Euclidean distance $\text{dist}(\boldsymbol{f}, \boldsymbol{s}_i) = \|\boldsymbol{f} - \boldsymbol{s}_i\|^2$).

The aforementioned background on SVM is sufficient to follow up the proposed algorithms. However, for more details about the SVM theory, the reader is referred to chapter 6 in [27] and to chapter 12 in [28].

2) The proposed SVM-based fall detection algorithm: In the following, the proposed SVM-based fall detection algorithm is explored where the objective is to design an embeddable solution that satisfies the best complexity/accuracy trade-off. The proposed design is illustrated in Figure 4 where two SVMs are used: the first one, namely SVM_L, is of low computational cost and high sensitivity, while the second one, namely SVM_Q, is more accurate but more computationally demanding. SVM_Q is called only when SVM_L claims the presence of a fall event. The aforementioned SVMs are described hereafter.

SVM_L uses a linear kernel function given by equation (9). Therefore, the equation of SVM_L is given by:

$$b_1 = M_1(\boldsymbol{f}) = \text{sign}(\boldsymbol{f}^\intercal \hat{\boldsymbol{\beta}}_1 + \hat{\beta}_0) \tag{12}$$

where $\hat{\boldsymbol{\beta}}_1 = \sum_{i=1}^{L} \alpha_i \boldsymbol{s}_i$.

SVM_Q uses the following proposed quadratic kernel:

$$\mathfrak{K}_Q(\boldsymbol{f}, \boldsymbol{s}_i) = \gamma_2(\boldsymbol{f}^\intercal \boldsymbol{s}_i)^2 + \gamma_1(\boldsymbol{f}^\intercal \boldsymbol{s}_i) + \gamma_0, \; \{\gamma_0, \gamma_1, \gamma_2\} \in \mathbb{R}^{+3} \tag{13}$$

Note that the latter kernel (Appendix B shows a proof that $\mathfrak{K}_Q$ is a kernel) allows to freely adjust the relative weightings between the features of degrees $(0, 1, 2)$ that is not the case of the polynomial kernel given by equation (10) as the relative weightings in (10) are governed by the following rule:

$$\gamma_i = \frac{d!}{i! \times (d-i)!} c^{d-i} \; \forall i \in \{0, 1, \ldots, d\} \tag{14}$$

The equation of SVM_Q is given by (see Appendix C for proof):

$$b_2 = M_2(\boldsymbol{f}) = \text{sign}(\boldsymbol{f}^\intercal \hat{\boldsymbol{B}} \boldsymbol{f} + \boldsymbol{f}^\intercal \hat{\boldsymbol{\beta}}_1 + \hat{\hat{\beta}}_0) \tag{15}$$

where $\hat{\hat{\beta}}_0 = \hat{\beta}_0 + \gamma_0 \sum_{i=1}^{L} \alpha_i$, $\hat{\boldsymbol{\beta}}_1 = \gamma_1 \sum_{i=1}^{L} \alpha_i \boldsymbol{s}_i$ and $\hat{\boldsymbol{B}} = \gamma_2 \sum_{i=1}^{L} \alpha_i \boldsymbol{s}_i \boldsymbol{s}_i^\intercal$. The pseudo-code of the proposed fall detection algorithm is shown below (Algorithm 1). One interesting feature that could be used instead of STD in the aforementioned algorithm is the range. This feature could also be calculated efficiently online using for example the algorithm proposed by Lemire [26]. The full pseudo-code of the proposed algorithm when using mean and range features is shown in Algorithm 2. Both aforementioned algorithms have been implemented and compared subsequently.

III. Computational Complexity

In this section, the computational complexity of the proposed fall detection algorithms is explored. Moreover, the complexity of several standard machine learning methods applied to the proposed features are shown. Particularly, the following methods have been tested: 1) SVM with linear, polynomial for $d \in \{2, 3, 4\}$ and RBF kernels, 2) artificial neural networks (ANN) and 3) k-nearest neighbors (KNN) classifier. Appendices D and E describe, respectively, the ANN and the KNN classifiers designed for performance evaluation. The complexity is calculated in terms of the number of floating point operations (flops) needed for making one decision (fall/no fall).

The complexity of the proposed feature extraction methods is as follows:

$$\mathfrak{C}_{f_1}(\kappa) = 3 \times (2\kappa + 15) \tag{16}$$

$$\mathfrak{C}_{f_2}(\kappa) = 12 \times (\kappa + 1) \tag{17}$$

where $\mathfrak{C}_{f_1}$ and $\mathfrak{C}_{f_2}$ represent the complexity of calculating {Mean, STD} and {Mean, Range} features, respectively. The complexity of polynomial kernel-based SVM classifier is:

$$\mathfrak{C}_{SVM_P}(N, L, d) = \min(N + 2\sum_{i=0}^{d-2} \frac{(N+d-i-1)!}{(N-1)!\,(d-i)!}, L(N+d+1)) \tag{18}$$

This formula could be used to calculate the complexity of equations (12) and (15) replacing d by 1 and 2, respectively. It also works for any polynomial kernel. The complexity of RBF-based SVM is:

$$\mathfrak{C}_{SVM_R}(N, L) = 2L(N+3) \tag{19}$$

The complexity of the proposed artificial neural network is:

$$\mathfrak{C}_{ANN}(N, n_1, n_2) = n_1(N + n_2 + 9) + n_2 \tag{20}$$

where n_1 and n_2 are the number of neurons in the hidden and output layers, respectively. The complexity of the KNN classifier is:

$$\mathfrak{C}_{KNN}(M, N, k) = M(2N + k) \tag{21}$$

where k is the number of nearest neighbors and M is the number of training vectors.
Using equations (16), (17) and (18), the complexities of Algorithms 1 and 2 are given, respectively, by:

$$\mathfrak{C}_1(\kappa, N) = 3 \times (2\kappa + 15) + N + \omega_q(N(N+2)) \quad (22)$$

$$\mathfrak{C}_2(\kappa, N) = 12 \times (\kappa + 1) + N + \omega_q(N(N+2)) \quad (23)$$

where $0 \leq \omega_q \leq 1$ is an experimental value that represents the percentage of calling SVM_Q. All the complexity expressions above could be multiplied by $\frac{\lambda}{\tau}$ to convert to flop-per-second.

IV. Experimental Results

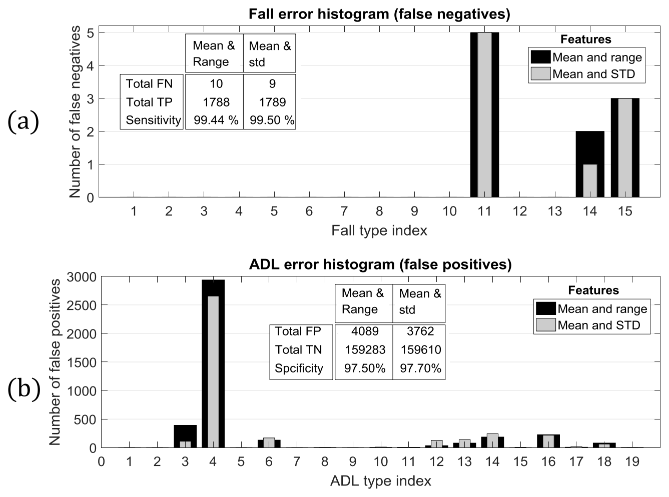
Performance criteria: the performance of the proposed fall detection algorithms is evaluated in terms of four performance criteria, namely computational complexity, accuracy, sensitivity and specificity. The last three quantities are computed as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (24)$$

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (25)$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \quad (26)$$

where $\text{TP}, \text{TN}, \text{FP}$ and FN denote true positives, true negatives, false positives (i.e. false alarms) and false negatives, respectively. FP is used as an explicit performance criterion as reducing the number of the false alarms is one of the main objectives of the proposed algorithms.
Dataset: a large open dataset named *Sisfall* [29] is used to evaluate the performance of the proposed algorithms. It consists of more than 4500 activity files representing 19 ADL and 15 fall types. These activities (see Tables 1 and 2 in [29]) were simulated by 23 young adults $\{A_1, \dots, A_{23}\}$ and 15 elderly ones $\{E_1, \dots, E_{15}\}$. Motion data were acquired using a wearable device mounted to the waist of the subject. This device contained two different accelerometers and one gyroscope. Sampling frequency of these sensors was set to 200 Hz.
In our experimental analysis, only acceleration data acquired using the energy efficient 3-axial accelerometer *ADXL345* are used as in [29]. In addition, the original acceleration data are down sampled at 40 Hz in our experiments for two reasons: 1) the original sampling frequency (200 Hz) is too high for fall detection as the maximum frequency of human body movement is 20 Hz [30] and a frequency of 40 Hz is sufficient according to Nyquist-Shannon sampling theorem, and 2) using a low sampling frequency decreases the power consumption since it decreases the power needed for acquisition and also decreases the complexity of feature extraction (see equations (16) and (17) and recall that $\kappa = f_s/f_e$). In order to evaluate the performance of the proposed algorithms, the aforementioned dataset is divided into two parts: the first one contains the activities performed by young adults $\{A_1, \dots, A_{12}\}$ and elderly $\{E_1, \dots, E_8\}$, while the second part contains activities performed by the remaining young adults $\{A_{13}, \dots, A_{23}\}$ and elderly $\{E_9, \dots, E_{15}\}$. Two experiments were executed: in the first one, the first part of the dataset was used for training and the second one for testing, while, in the second experiment, the second part of the dataset was used for training and the first part for testing. The total numbers of $\text{TP}, \text{TN}, \text{FP}$ and FN are counted from both experiments, then these quantities are used to assess the performance. In addition, the frequency of calling SVM_Q is also calculated from both experiments. Note that activities performed by somebody are always tested using machines trained by activities of different persons, leading to realistic evaluation.



	Mean & Range	Mean & std
Total FN	10	9
Total TP	1788	1789
Sensitivity	99.44 %	99.50 %

	Mean & Range	Mean & std
Total FP	4089	3762
Total TN	159283	159610
Spcificity	97.50%	97.70%

Fig. 5. Performance of SVM_L applied to the proposed features: error histograms over all fall and ADL types in *Sisfall* dataset [29], (a) histogram of false negatives, (b) histogram of false positives.

Configurations: For both algorithms 1 and 2, configurations $\kappa = 12$, $\lambda = 10$, $\tau = 3$ seconds and $f_s = 40$ Hz are used. SVM_Q is designed to achieve high accuracy. Particularly, $\{\gamma_0, \gamma_1, \gamma_2\} = \{1, \frac{1}{(2.2)^2}, \frac{1}{2(2.2)^4}\}$. These values are motivated from Taylor expansion of the Gaussian function with a standard deviation of $\sigma = 2.2$. The latter value showed the best performance with the Gaussian kernel-based SVM.

Experimental results: Given the aforementioned configurations, the total storage space needed for the 18 buffers $(\boldsymbol{l}^{(\rho)}, \boldsymbol{s}^{(\rho)}, \boldsymbol{\sigma}^{(\rho)}, \boldsymbol{\mu}^{(\rho)}, r_1^{(\rho)}, r_2^{(\rho)}) \forall \rho \in \{x, y, z\}$ is $(4 \times \delta + 2) \times 3 \times 4 = 312$ Bytes (using 4-Byte floating point numbers). This storage requirement is about 43% of the space needed to store the whole samples within the 3-second sliding window which is $f_s \times \tau \times 3 \times 2 = 720$ Bytes where the raw samples are 2-Byte integers.
False negatives and false positives histograms of SVM_L given by equation (12) are illustrated in Figure 5.a and 5.b, respectively. The aforementioned linear SVM satisfies the design objectives of achieving high sensitivity with very low complexity. Numerically, it achieves a sensitivity of 99.50% and 99.44% (see Tables I and II) when using {Mean, STD} and {Mean, Range} features, respectively, and a complexity of $N = 12$ $flops$ only. Figure 5.a shows that slow falls are the only reason of false negatives. Particularly, fall type 11 represents falling backward when trying to sit down, while

TABLE I
PERFORMANCE OF SEVERAL MACHINE LEARNING-BASED METHODS USING {MEAN, STD} FEATURES IN TWO SCENARIOS: 1) THE FEATURES ARE EXTRACTED GLOBALLY FROM THE WHOLE SLIDING WINDOW AND 2) THE FEATURES ARE EXTRACTED USING THE PROPOSED METHOD.

Feature extraction	Conventional					Two-segment (proposed)				
Criteria → Method ↓	Acc. (%)	Sen. (%)	Spe. (%)	FP	Com. (flops)	Acc. (%)	Sen. (%)	Spe. (%)	FP	Com. (flops)
KNN	98.92	98.83	98.92	1759	**33233**	**99.37**	**99.78**	**99.37**	**1029**	56609
ANN	95.42	99.67	95.37	7561	**278**	**98.70**	**99.83**	**98.69**	**2140**	338
SVM, equation (9)	92.72	98.89	92.64	12020	**123**	**97.72**	**99.50**	**97.70**	**3758**	129
SVM, equation (10), $d=2$	98.46	98.22	98.46	1858	**165**	**99.59**	**99.78**	**99.59**	**670**	183
SVM, equation (10), $d=3$	96.58	98.61	96.56	5624	**277**	**98.94**	**99.78**	**98.93**	**1748**	1013
SVM, equation (10), $d=4$	96.32	98.11	96.30	5275	**529**	**99.28**	**99.89**	**99.28**	**1176**	2225
SVM, equation (11)	98.76	97.39	98.78	1995	**3231**	**99.85**	**99.72**	**99.85**	**245**	5577
SVM_Q, equation (15)	98.62	97.72	98.63	2242	**165**	**99.82**	**99.78**	**99.82**	**294**	183
Algorithm 1 (proposed)	-	-	-	-	-	99.92	99.17	99.93	**114**	131

TABLE II
PERFORMANCE OF SEVERAL MACHINE LEARNING-BASED METHODS USING {MEAN, RANGE} FEATURES IN TWO SCENARIOS: 1) THE FEATURES ARE EXTRACTED GLOBALLY FROM THE WHOLE SLIDING WINDOW AND 2) THE FEATURES ARE EXTRACTED USING THE PROPOSED METHOD.

Feature extraction	Conventional					Two-segment (proposed)				
Criteria → Method ↓	Acc. (%)	Sen. (%)	Spe. (%)	FP	Com. (flops)	Acc. (%)	Sen. (%)	Spe. (%)	FP	Com. (flops)
KNN	95.51	**98.94**	95.47	7406	**33272**	**98.28**	97.50	**98.29**	**2794**	56648
ANN	95.23	99.83	95.18	7881	**317**	**98.29**	**99.83**	**98.28**	**2810**	377
SVM, equation (9)	93.20	98.67	93.13	11221	**162**	**97.52**	**99.44**	**97.50**	**4084**	168
SVM, equation (10), $d=2$	99.41	93.05	99.48	842	**204**	**99.74**	**99.05**	**99.75**	**408**	222
SVM, equation (10), $d=3$	98.56	92.21	98.63	2234	**316**	**99.03**	**99.39**	**99.02**	**1601**	1052
SVM, equation (10), $d=4$	98.90	93.72	98.97	1689	**568**	**99.36**	**99.39**	**99.36**	**1046**	1992
SVM, equation (11)	99.65	92.38	**99.74**	**423**	**2982**	**99.70**	**99,56**	99.70	490	5616
SVM_Q, equation (15)	99.65	93.05	99.73	434	**204**	**99.89**	**99.61**	**99.89**	**180**	222
Algorithm 2 (proposed)	-	-	-	-	-	99.94	99.05	99.95	**82**	170

fall types 14 and 15 represent backward and lateral falling while sitting, caused by fainting or falling asleep, respectively. Figure 5.b shows that jogging quickly (ADL type 4) is the main reason of false positives. To summarize, false negatives are caused by slow falls while false positives are caused by fast ADLs. This means that the specificity in real world application is expected to be greater than the one evaluated in this experimental analysis as most of the activities of the elderly are slow contrary to the dataset at hand where, for instance, the number of simulated walking activities is equal to the number of quickly jogging ones.

Positive decisions $(\mathrm{TP}+\mathrm{FP})$ made by SVM_L require calling SVM_Q. Consequently, $\omega_q = (\mathrm{TP}+\mathrm{FP})/(\mathrm{TP}+\mathrm{FP}+\mathrm{TN}+\mathrm{FN})$ where $\mathrm{TP}, \mathrm{TN}, \mathrm{FP}$ and FN are calculated for SVM_L. Numerically, $\omega_q = 3.36\%$ when using {Mean, STD} features and $\omega_q = 3.56\%$ when using {Mean, Range} features.

Tables I and II show the performance of the two proposed algorithms as well as all the machine learning-based methods considered in this paper using {Mean, STD} and {Mean, Range} features, respectively. The performance is evaluated in two different scenarios: 1) the features are extracted globally from the whole sliding window (conventional method) and 2) the features are extracted using the proposed two-segment feature extraction method. The considered performance criteria are accuracy, sensitivity, specificity, number of false alarms and computational complexity abbreviated in Tables I and II as Acc, Sen, Spe, FP and Com, respectively. In order to simplify the comparison between the results based on the conventional method vs. the proposed feature extraction method, the best performance values are shown in bold-face in Tables I and II.

Table I shows that the two-segment feature extraction improves the performance of all the considered classifiers in terms of accuracy, sensitivity, specificity and the number of false alarms FP. The latter criterion shows the significance of the specificity improvement. Let us consider, as an example, the best specificity shown using the conventional feature extraction with {Mean, STD} features. KNN shows the best specificity 98.92%. Now, comparing this result with the specificity of Algorithm 1 i.e. 99.93%, the latter improved the specificity only by 1.01%. However, this difference is significant as it means reducing the number of false alarms from 1759 to 114 only.

As the computational complexity is proportional to the number of extracted features N, using the conventional feature extraction results in lower complexity assuming that the conventional feature extraction is also implemented on-line as in the proposed method. However, the results in Table I clearly show that Algorithm 1 satisfies the best accuracy/complexity tradeoff. Very similar conclusions could be derived from Table II. The only difference in Table II is that the nonlinear versions of the SVM show high specificity using the conventional features. However, as shown in this table, the competitive specificity is at the expense of the sensitivity. Generally, for all the considered classifiers and features, the classification accuracy has been improved using the two-segment feature extraction. Algorithms 1 and 2 satisfy the best accuracy/complexity tradeoff and considerably reduce the number of false alarms thanks to 1) the highly discriminative two-segment feature extraction method, 2) the efficient online implementation of feature extraction and 3) the proposed machine learning-based system. It is worth mentioning that the high accuracy achieved by all the considered machine learning-based methods when applied to the two-segment-based features reflects the strength of these features in discriminating between falls and ADLs.

V. Conclusion and Future Work

A challenging problem of designing an accurate low-power consumption wearable fall detector for indoor and outdoor environments was tackled in this paper. Thanks to the proposed two-segment feature extraction method, the proposed algorithms showed a high discriminative power in classifying falls and ADLs. Numerically, the accuracy of these algorithms exceeded 99.9%. Moreover, these algorithms showed a low computational cost thanks to the online method for feature extraction and to the proposed design of the machine learning-based system that satisfies the best accuracy/complexity trade-off. One appealing feature of the proposed algorithms, that will be investigated in a future work, is their ability to work with multiple sampling frequencies. This feature allows for more power saving by switching to low sampling frequencies when the estimated activity of the user is low.

Appendix A
Calculating mean and STD online using Welford algorithm

Another way to calculate mean and STD features online is to use Welford algorithm [25]. Although this algorithm doesn't calculate these quantities for a sliding window, it could be modified and applied to sliding windows as follows:

$$\begin{aligned}\mu_+^{(\rho)} &= \mu^{(\rho)} + \frac{\rho_{in} - \mu^{(\rho)}}{K+1}\\ Z_+^{(\rho)} &= Z^{(\rho)} + (\rho_{in} - \mu^{(\rho)})(\rho_{in} - \mu_+^{(\rho)})\\ \mu^{(\rho)} &= \mu_+^{(\rho)} + \frac{\mu_+^{(\rho)} - \rho_{out}}{K}\\ Z^{(\rho)} &= Z_+^{(\rho)} - (\rho_{out} - \mu_+^{(\rho)})(\rho_{out} - \mu^{(\rho)})\\ \sigma^{(\rho)} &= \sqrt{\frac{Z^{(\rho)}}{K-1}}\end{aligned} \tag{27}$$

where $\mu^{(\rho)}$ and $\sigma^{(\rho)}$ are the mean and STD, respectively $\forall \rho \in \{x, y, z\}$, ρ_{in} and ρ_{out} represent the samples that enter and leave the window, respectively. Contrary to our proposed algorithm, the online update rules (27) require the whole window to be stored (because ρ_{out} is needed). In addition, they require a complexity of $\mathfrak{C}_{\tilde{f}_1} = 6 \times (7\kappa + 3)$ which is 4 times greater than $\mathfrak{C}_{f_1}$ for the considered configurations ($\kappa = 12$).

Appendix B
Proof that $\mathfrak{K}_Q$ given by equation (13) is a kernel

Given that $\mathfrak{K}(\boldsymbol{f}, \boldsymbol{s}_i) = (\boldsymbol{f}^\intercal \boldsymbol{s}_i)^d$ is a kernel $\forall d \in \mathbb{Z}^+ - \{0\}$ [27] and using the following kernel properties [27]:

- The sum of two kernels is a kernel.
- If $\mathfrak{K}(\boldsymbol{f}, \boldsymbol{s}_i)$ is a kernel, then $a\mathfrak{K}(\boldsymbol{f}, \boldsymbol{s}_i)$ is a kernel $\forall a \in \mathbb{R}^+$.

the function $\mathfrak{K}'(\boldsymbol{f}, \boldsymbol{s}_i) = \gamma_2(\boldsymbol{f}^\intercal \boldsymbol{s}_i)^2 + \gamma_1(\boldsymbol{f}^\intercal \boldsymbol{s}_i)$ is clearly a kernel $\forall\{\gamma_1, \gamma_2\} \in {\mathbb{R}^+}^2$. Equation (13) could be written as $\mathfrak{K}_Q(\boldsymbol{f}, \boldsymbol{s}_i) = \mathfrak{K}'(\boldsymbol{f}, \boldsymbol{s}_i) + \gamma_0$ with $\gamma_0 \in \mathbb{R}^+$. Let's define a map $\phi : \boldsymbol{x} \rightarrow [\phi'(\boldsymbol{x}), \sqrt{\gamma_0}]^\intercal$ where ϕ' is the feature map of $\mathfrak{K}'$. Using ϕ, $\mathfrak{K}_Q$ could be written as a dot product $\mathfrak{K}_Q(\boldsymbol{f}, \boldsymbol{s}_i) = \phi^\intercal(\boldsymbol{f})\phi(\boldsymbol{s}_i) = \phi'^\intercal(\boldsymbol{f})\phi'(\boldsymbol{s}_i) + \gamma_0$ and hence it is a kernel by definition.

Appendix C
Proof of equation (15)

Replacing (13) in (8) yields:

$$\begin{aligned}\hat{\mathfrak{F}}(\boldsymbol{f}) &= \hat{\beta}_0 + \sum_{i=1}^{L} \alpha_i \mathfrak{K}_Q(\boldsymbol{f}, \boldsymbol{s}_i)\\ &= \hat{\beta}_0 + \sum_{i=1}^{L} \alpha_i\gamma_0 + \sum_{i=1}^{L} \alpha_i\gamma_1(\boldsymbol{f}^\intercal \boldsymbol{s}_i)\\ &\quad + \sum_{i=1}^{L} \alpha_i\gamma_2(\boldsymbol{f}^\intercal \boldsymbol{s}_i)^2\\ &= (\hat{\beta}_0 + \sum_{i=1}^{L} \alpha_i\gamma_0) + \boldsymbol{f}^\intercal(\gamma_1 \sum_{i=1}^{L} \alpha_i \boldsymbol{s}_i)\\ &\quad + \boldsymbol{f}^\intercal(\gamma_2 \sum_{i=1}^{L} \alpha_i(\boldsymbol{s}_i \boldsymbol{s}_i^\intercal))\boldsymbol{f}\\ &= \hat{\hat{\beta}}_0 + \boldsymbol{f}^\intercal \hat{\boldsymbol{\beta}}_1 + \boldsymbol{f}^\intercal \hat{\boldsymbol{B}} \boldsymbol{f}\end{aligned}$$

Replacing the last expression in (7) yields (15).

Appendix D
Description of the artificial neural network

The neural network used in Section IV is a feed-forward one. It consists of one hidden layer of n_1 neurons and an output layer of n_2 neurons. The input of this network is $\boldsymbol{f}$, an N-dimensional features vector. The class of $\boldsymbol{f}$ is determined using the following equations:

$$\begin{aligned}\boldsymbol{r}_1 &= \text{tansig}(\boldsymbol{W}^{(1)}\boldsymbol{f} + \boldsymbol{b}^{(1)})\\ \boldsymbol{r}_2 &= \boldsymbol{W}^{(2)}\boldsymbol{r}_1 + \boldsymbol{b}^{(2)}\\ \text{class}(\boldsymbol{f}) &= \text{sign}(\boldsymbol{r}_2 - 0.5)\end{aligned}$$

where $\boldsymbol{W}^{(1)}$ is an $n_1 \times N$ matrix representing the weights of the links between the network input and the neurons of the hidden layer, $\boldsymbol{b}^{(1)}$ an n_1-dimensional vector represents the bias of the hidden layer neurons, $\boldsymbol{W}^{(2)}$ an $n_2 \times n_1$ matrix represents the weights of the links between the hidden and the output layers, $\boldsymbol{b}^{(2)}$ an n_2-dimensional vector represents the bias of the output layer neurons and $\text{tansig}(x) = 2/(1+\exp(-2x))-1$ is the transfer function of the hidden layer. Numerically, $n_1 = 10$ and $n_2 = 1$ in the proposed design.

Appendix E
Description of the KNN classifier

In KNN-based classification, training is nothing but storing the whole training samples (an $M \times N$ matrix, say $\boldsymbol{Y}$) with their corresponding ground truth. The KNN classifier evaluated in Section IV uses the *exhaustive search* method, i.e. the distances between the subject features vector and all the training vectors are measured. The resultant distances are ranked together with the corresponding ground truth in ascending order and the decision depends on the first k results. The aforementioned classifier uses the standard squared euclidean distance $\boldsymbol{D} = (\boldsymbol{x} - \boldsymbol{Y}_m)(\boldsymbol{x} - \boldsymbol{Y}_m)^\intercal \, \forall m \in \{1, \ldots, M\}$. Numerically, k is set to 5 in the proposed design.

Acknowledgment

This publication is supported by the European Union through the European Regional Development Fund (ERDF), the Ministry of Higher Education and Research, the French Region of Brittany and Rennes Métropole.

References

[1] WHO Media centre. Falls, fact sheet (reviewed january 2018) http://www.who.int/mediacentre/factsheets/fs344/en/, 2018.

[2] K. Chaccour, R. Darazi, A. Hajjam El Hassani, and E. Andrès. From fall detection to fall prevention: A generic classification of fall-related systems. *IEEE Sensors Journal*, 17(3):812–822, 2017.

[3] X. Yu. Approaches and principles of fall detection for elderly and patient. In *HealthCom*, Singapore, Singapore, Jul. 7-9 2008.

[4] P. Pierleoni, A. Belli, L. Maurizi, L. Palma, L. Pernini, M. Paniccia, and S. Valenti. A wearable fall detector for elderly people based on ahrs and barometric sensor. *IEEE Sensors Journal*, 16(17):6733 – 6744, 2016.

[5] S.O.H. Madgwick, A.J.L. Harrison, and R. Vaidyanathan. Estimation of imu and marg orientation using a gradient descent algorithm. In *IEEE International Conference on Rehabilitation Robotics (ICORR)*, Zurich, Switzerland, 29 June-1 July 2011.

[6] A.M. Sabatini, G. Ligorio, A. Mannini, V. Genovese, and L. Pinna. Prior-to- and post-impact fall detection using inertial and barometric altimeter measurements. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 24(7):774 – 783, 2016.

[7] Q. Liu, J. Williamson, K. Li, W. Mohrman, Q. Lv, R.P. Dick, and L. Shang. Gazelle: Energy-efficient wearable analysis for running. *IEEE Transactions on Mobile Computing*, 16(9):2531 – 2544, 2017.

[8] S. Yoo and D. Oh. An artificial neural network-based fall detection. *International Journal of Engineering Business Management*, 10:1 – 7, 2018.

[9] F. J. Ordonez and D. Roggen. Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115, 2016.

[10] C. A. Ronao and S. Cho. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert Systems with Applications*, 59:235–244, 2016.

[11] A. Kurniawan, A.R. Hermawan, and I.K.E. Purnama. A wearable device for fall detection elderly people using tri dimensional accelerometer. In *IEEE, International Seminar on Intelligent Technology and Its Applications (ISITIA)*, Lombok, Indonesia, Jul. 28 - 30 2016.

[12] F. Wu, H. Zhao, Y. Zhao, and H. Zhong. Development of a wearable-sensor-based fall detection system. *International Journal of Telemedicine and Applications*, vol. 2015, Article ID 576364, 2015.

[13] J. Chen, K. Kwong, D. Chang, J. Luk, and R. Bajcsy. Wearable sensors for reliable fall detection. In *IEEE, Engineering in Medicine and Biology*, Shanghai, China, Sep. 1 - 4 2005.

[14] S. Abdelhedi, R. Bourguiba, J. Mouine, and M. Baklouti. Development of a two-threshold-based fall detection algorithm for elderly health monitoring. In *IEEE, Research Challenges in Information Science (RCIS)*, Grenoble, France, Jun. 1 - 3 2016.

[15] L.P. Nguyen, M. Saleh, and R. Le Bouquin Jeannès. An efficient design of a machine learning-based elderly fall detector. In *HealthyIoT, the 4th EAI International Conference on IoT Technologies for HealthCare*, Angers, France, Oct. 24-25 2017.

[16] O. Aziz, M. Musngi, E.J. Park, G. Mori, and S.N. Robinovitch. A comparison of accuracy of fall detection algorithms (threshold-based vs. machine learning) using waist-mounted tri-axial accelerometer signals from a comprehensive set of falls and non-fall trials. *Medical and Biological Engineering and Computing*, 55(1):45–55, 2017.

[17] A. Leone, G. Rescio, and P. Siciliano. Supervised machine learning scheme for tri-axial accelerometer-based fall detector. In *IEEE Sensors*, Baltimore, MD, USA, Nov. 3-6 2013.

[18] N. Noury, A. Fleury, P. Rumeau, A.K. Bourke, G.Ó. Laighin, V. Rialle, and J.E. Lundy. Fall detection - principles and methods. In *IEEE, Engineering in Medicine and Biology Society (EMBS)*, Lyon, France, Aug. 22 - 26 2007.

[19] K. H. Chen, J. J. Yang, and F. S. Jaw. Accelerometer-based fall detection using feature extraction and support vector machine algorithms. *Instrumentation Science and Technology*, 44(4):333–342, 2016.

[20] G. Shi, C. S. Chan, W. J. Li, K. S. Leung, Y. Zou, and Y. Jin. Mobile human airbag system for fall protection using mems sensors and embedded svm classifier. *IEEE Sensors Journal*, 9(5):495–503, 2009.

[21] A. Jahanjoo, M. N. Tahan, and M. J. Rashti. Accurate fall detection using 3-axis accelerometer sensor and mlf algorithm. In *International Conference on Pattern Recognition and Image Analysis (IPRIA)*, Shahrekord, Iran, Apr. 19 - 20 2017.

[22] W. C. Cheng and D. M. Jhan. Triaxial accelerometer-based fall detection method using a self-constructing cascade-adaboost-svm classifier. *IEEE Journal of Biomedical and Health Informatics*, 17(2):411–419, 2013.

[23] N. Mezghani, Y. Ouakrim, M. R. Islam, R. Yared, and B. Abdulrazak. Context aware adaptable approach for fall detection bases on smart textile. In *IEEE EMBS International Conference on Biomedical and Health Informatics (BHI)*, Orlando, FL, USA, Feb. 16-19 2017.

[24] P. Pierleoni, L. Pernini, A. Belli, L. Palma, S. Valenti, and M. Paniccia. Svm-based fall detection method for elderly people using android low-cost smartphones. In *IEEE, Sensors Applications Symposium (SAS)*, Zadar, Croatia, Apr. 13 - 15 2015.

[25] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3), 1962.

[26] D. Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *Nordic Journal of Computing*, 13(4):328–339, 2006.

[27] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press New York, 1 edition, 2000.

[28] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag New York, 2 edition, 2009.

[29] A. Sucerquia, J.D. Lòpez, and J.F. Vargas-Bonilla. Sisfall: A fall and movement dataset. *Sensors Journal*, 17(1), 2017.

[30] C.V.C. Bouten, K.T.M. Koekkoek, M. Verduin, R. Kodde, and J.D. Janssen. A triaxial accelerometer and portable data processing unit for the assessment of daily physical activity. *IEEE Transactions on Biomedical Engineering*, 44(3):136 – 147, 1997.