Pedometers for Smartphones: Analysis and Comparison of Real-Time Algorithms

# Pedometers for Smartphones: Analysis and Comparison of Real-Time Algorithms

Giacomo Neri[†], Federico Montori[†*], Lorenzo Gigli[†*], Luca Bedogni[‡], Marco Di Felice[†*], Luciano Bononi[†]
[†]*Department of Computer Science and Engineering, University of Bologna, Italy*
[*]*Advanced Research Center in Electronic Systems (ARCES), University of Bologna, Italy*
[‡]*Department of Physics, Informatics and Mathematics, University of Modena and Reggio Emilia, Italy*
Corrsponding Author's Email: federico.montori2@unibo.it

***Abstract*—The recent years have witnessed the rise of an enormous number of software algorithms that implement pedometers (or step counters), which led to the development of several context-aware IoT-based smartphone apps for sports and healthcare, among others. While the number of scientific works in this context is high, there is no comparison study that analyzes the different proposal at implementation level. In this paper we first perform a literature review of software implementations of pedometers for smartphones and then classify them into a taxonomy. With this, we highlight the similarities of their scheme, which is based on a number of defined steps to be applied in a pipeline. We then develop a smartphone application that implements all the configurations of these steps found in literature and evaluates them in various scenarios. Finally, we present comparative results obtained by running extensive and real tests that show the importance of a carefully designed filtering step.**

***Index Terms*—Internet of Things, Pedometers, Step Counter, Activity Recognition**

## I. Introduction

Pedometers, or step counters, are devices or software modules designed to count the steps performed by an individual through motion detection. Historically, pedometers were intended as dedicated physical devices, used to assess a person's physical activity or energy expenditure. More recently, numerous software implementation of pedometers on smartphones and wearables IoT devices have become widespread, as raw data sampled from the embedded sensors of these devices can be processed in order to assess whether steps have been performed [1]. As a matter of fact, studies concerning the implementation of pedometer algorithms for smartphones are now several. By analyzing these studies, it is possible to notice how the majority of them follow a recurrent pattern in terms of implementation choices, often differing only by some defined values or the order of certain algorithmic steps. Furthermore, almost all of the analyzed studies declare a very high precision of the proposed implementation, making it difficult to assess coherently the strength points and the limitations of each of them. Building on these premises, the present study has the goal of performing a detailed analysis of most of these studies, in particular those based on Android devices, by collecting the proposed implementation choices, comparing them with each other and, finally, highlighting their advantages and drawbacks through a concrete and all-encompassing implementation. With this research effort we attempt to give a glance on the factual efficacy of the proposed software pedometers.

The work is divided in four different macro-phases. The first phase consists in the information collection, which we performed through a literature review. In particular, we collected a number of pedometer implementation and described them in Section II, where we analyze the state of the art from which it is already possible to identify the common ground that most of the existing works follow. Next, the second phase consists in the collection of all the common features that characterize the existing works. This led us to frame them within a detailed *taxonomy*. The taxonomy includes all the main algorithmic steps of a software pedometer, by grouping together redundant ones used in different works and providing a clear visualization of the different options that can be chosen for each of them. The taxonomy is outlined throughout Section III. In this work, our taxonomy and subsequent evaluation only considers *real-time* algorithms, which detect whether a step is taken or not right after the relative data point is sampled. Other algorithms, identified as *offline*, count the steps after a certain number of measurements are performed, thus having access to more information for a more accurate processing. These belong to a different category and will be considered for future works. Subsequently, in the third phase we develop an Android mobile application, through which a user can test any of the pedometer implementation proposed in the analyzed studies. The application is described in Section IV and its implementation is guided by the taxonomy proposed in the previous section, thus allowing for the combination of different algorithmic steps used in different works in a unique pipeline. The last phase, described in Section V, includes extensive performance tests of the analyzed algorithms through the developed application. These have been performed by several human volunteers in different conditions and results show how the filtering step has a strong impact on the precision and the effectiveness of such algorithms. Finally, we draw conclusions in Section VI.

## II. State of The Art

Technological advancement has gradually replaced the historical mechanic pedometers, used mainly for military purposes, with dedicated sensors first and with software modules later. With the advent of smartphones and wearable IoT

devices that embed inertial sensors, the deployment of pedometers is nowadays pervasive and potentially accompanying the device owner all day. IoT software pedometers are then relying on two main components: the sensors and the detection algorithm. The smartphone sensors used for the purpose of counting steps are mainly accelerometer, gyroscope and magnetometer – this works similarly for other IoT devices. The accelerometer detects the acceleration of the device along the X, Y and Z axes (including the gravity acceleration to which it is constantly exposed). The gyroscope detects the device rotation around the X (roll), Y (pitch) and Z (yaw) axes. Finally, the magnetometer detects the orientation of the magnetic field to which the device is exposed. In absence of artificial ones, it is in practice able to discern the device orientation. Sensors embedded into different smartphones are quite similar to each other, their only difference is the maximum sampling frequency, ranging from 100 Hz to 500 Hz depending on the price, however this does not affect the capability of detecting steps, which, as we will see, needs much lower sampling frequencies. The main discriminant for different performances is therefore the detection algorithm, which is the main object of the present paper. The section below aims to discuss the main solutions found in literature.

### A. Detection Algorithms

By looking at the body of literature, we found that almost every publication implement the detection algorithm via a number of steps to be applied in a sequence. This pipeline approach is composed by the following steps:

1) Gathering of sensor data.
2) Application of filters to clean out possible noise.
3) Isolation of relevant information that can possibly correspond to steps.
4) Application of a decision algorithm to decide whether the point corresponds to a step.

As a first example, we take into account the work proposed by [2], which uses all three sensors cited above and a low-pass filter to erase all data that presents a magnitude lower than a certain cutoff frequency. This greatly reduces the amount of data to be analyzed. Furthermore, it applies a rotation matrix to the data, by using the orientation given by the magnetometer, in order to convert the accelerometer values into a rotation-independent reference system. Subsequently, the decision process applies the "Peak algorithm" in order to detect whether a sensor observation is a local maximum or a local minimum and takes the final decision upon them. More in detail, this decision is made through a further step that divides the data gathered in segments and executes the detection algorithm a posteriori on each of the segments. This implies that the detection algorithm is *offline*, therefore it tries to achieve a higher accuracy by looking at an extended set of data, assuming however to have access to an entire time series and therefore tolerating a certain delay in yielding the result. This also implies a number of different policies on how to segment the time series (in [2] it is done by looking at the gyroscope values). In certain use cases this delay is not acceptable because the pedometer algorithm is part of a real-time pipeline (as in activity recognition or indoor navigation [3]), thus a *real-time* detection algorithm, able to yield a result for each sensor observation, is needed.

Much like in [2], all the studies analyzed in this section use the same pipeline scheme, differing only in how each step is performed. For instance, the works in [4], [5], [6], [7] e [8] all use a low-pass filter, however setting a different cutoff frequency, which varies from 2 Hz to 10 Hz. Among these, only [6] and [7] describe a completely real-time detection algoritm, while all the others use the time series segmentation. In particular, while [5] e [8] use a segmentation process similar to [2], the work in [4] proposes to cut the time series in correspondence of the intersection of the data with the $x$-axis (this is subsequently proposed by other studies). Concerning the sensors, [4] proposes the sole usage of the gyroscope; [7] and [8] only use the accelerometer, while the work in [6] uses all three of them. The latter, in fact, expresses the necessity of a rotation-independent coordinate system. Other papers proposing such a conversion are [9], [10], [11] and [12]. In all these cases all three sensors are used, however only [6], [9] and [10] use a rotation matrix for this purpose. The conversion to a fixed reference system allows the developer to only focus on the sensor measurements on the $z$-axis, the one that is subject the most to sensitive variations, as it is vertical with respect to the user when the device is in neutral position. This can be clearly observed in [2] and [9]. Furthermore, this conversion facilitates the removal of the gravitational acceleration as it will be only on one axis. In alternative, [11] and [12] use the acceleration magnitude, which is again independent from the reference system. Out of these studies, [9] and [11] propose an offline detection process, while [10] e [12] operate at real-time. Other papers, such as [13], [14], [15], [16] and [17] make use of the accelerometer only, without any filtering step, then apply the peaks algorithm directly on raw data. [13] and [15] implement the process at real-time, while [14], [16] e [17] recognize the peaks within certain segments.

### B. Comparative Studies

Pedometer applications have been around for several years, therefore it is expected that a number of survey studies like the one in this paper had been conducted. For instance, the study in [1] evaluates three commercial mobile applications freely available for download in a laboratory environment (i.e. Accupedo, Moves, and Runtastic Pedometer). Differently, in [18] the authors perform an thorough performance evaluation of a commercial pedometer application against a mechanical pedometer in order to investigate its validity under different operation conditions (e.g. speed, location, etc.). Similar studies have been conducted for this purpose, however, to the best of our knowledge, no analysis has yet been performed to compare different algorithms for pedometers under a common ground. Even though the study of this paper is limited to a certain category of algorithms (i.e. non commercial, real-time) it is still a solid basis for encompassing more cases.

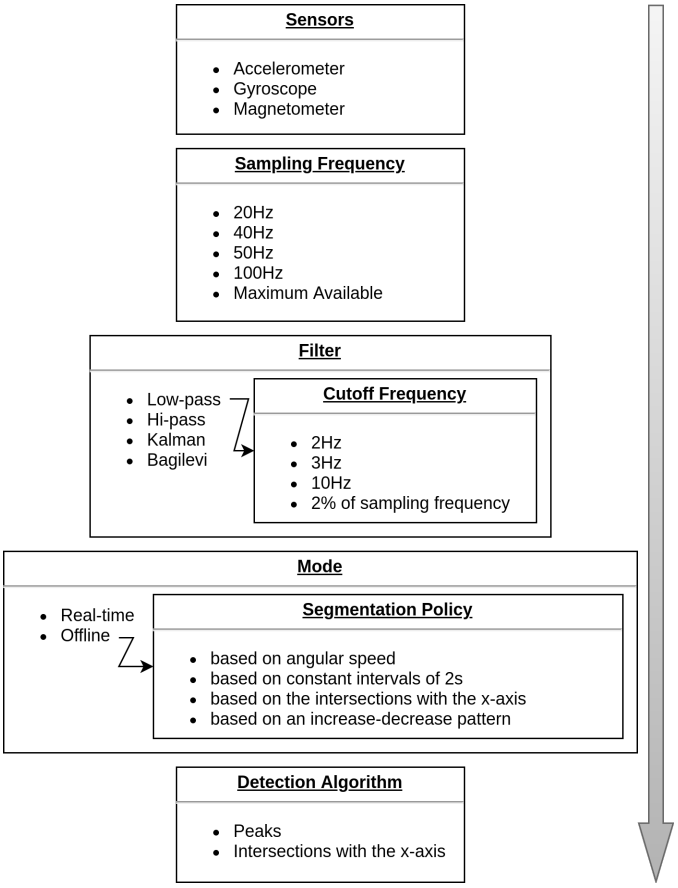



Fig. 1. A simple pictorial representation of the proposed taxonomy, outlining all phases to be executed in sequence.

## III. Taxonomy

Starting from the information collected in the previous section, we can define a complete and exhaustive taxonomy of the steps involved in almost each of the existing implementations. The taxonomy is presented graphically in Figure 1 and, for each step, it explores all the options found in literature. The taxonomy can be used as a guideline for the combination of such options, which could possibly derive new implementations that were not found in literature before. Below we give an overview of each of the steps.

### A. Sensors

The usage of one or more sensors at the same time can yield different results. In this case the choice of the sensors is not mutually exclusive, meaning that an algorithms can use them in any possible combination. This step also involves implicitly any software sensor that combines or alters the data from accelerometers, gyroscopes and/or magnetometers, for instance the compass, the orientation sensor or the linear acceleration sensor.

### B. Sampling Frequency

Sensors can be sampled at a different frequency in an IoT device and this can impact the final results. Typically the sampling frequency spans from 20 Hz to the maximum available, however, Figure 1 only reports the values that have been found in literature. A higher frequency does not necessarily mean better performance, as it is likely to generate more noise, thus needing a more sophisticated filter.

### C. Filter

The filtering process is an important step when dealing with data that can possibly include noise, as they aim to clean up the waveform in a way in which it is easier to process in the subsequent steps. The most used filter is the low-pass filter, which attenuates portion of signals presenting a high frequency, mostly associated with noise. This filter must be tuned by selecting a proper *cutoff frequency*, which indicates the frequency below which the signal is left untouched. Other filters found are the Kalman filter (used with all three sensors) the hi-pass filter (used only with the gyroscope) and the Bagilevi filter (only used with the accelerometer). The latter is not associated to a specific work in literature, as it is only proposed in the form of an implementation freely available on Gihtub[1], however, we included it because it has been used as it is in scientific papers (e.g. [19]).

### D. Mode (or Modality)

With this term we mean whether the algorithm works in real-time or offline. As we anticipated, a real-time algorithm outputs the result of the computation immediately after the data point is sampled, while the offline algorithms need to wait for a whole data segment to be acquired, in order to process it with the whole information content available. Most of the times the acquisition windows do not last for more than few seconds, however the algorithms themselves are more complicated to apply and still cause a delay in returning a feedback. In scenarios where the output of the pedometer must be instantaneous, for instance when it is part of a more complex an time-sensitive pipeline of tools, then offline algorithms are less preferable, even though their output is likely to be more accurate. In this paper we do not deal with offline algorithms, which are envisioned for future works, therefore, we skip the discussion on the data segmentation criteria. Real-time algorithms simply do not apply this step.

### E. Detection Algorithm

This is the final step that determines whether a data point corresponds to a step or not. All the studies considered implement the "Peaks algorithm", which identifies a step if the data point is a local maximum or minimum in the data series. Some of the works apply a rotation matrix to accelerometer and gyroscope data, some other do not, as peaks tend to be significant in any direction. The application of the filter prior to this step is quite important, as the number of local maximum/minimum in a noisy wave is much higher, since they may be caused by small vibrations that are not necessarily steps. Another aspect that is applied in this phase and helps

[1] https://github.com/bagilevi/android-pedometer

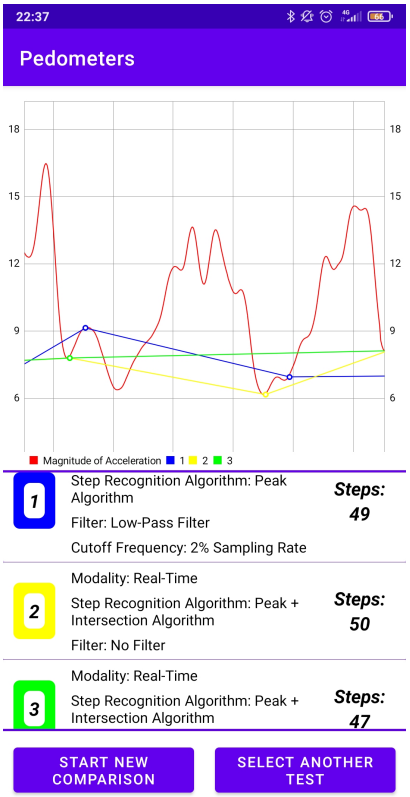



Fig. 2. Screenshot of the mobile application showing the comparison of different configurations and, for each of them, a visual feedback on where the steps are detected in the data trace.

in filtering out false positives is the intersection with the $x$-axis (accelerometer only): an additional check that does not count a step if the wave did not cross the $x$-axis after the last maximum/minimum.

## IV. Data Collection through a Mobile Application

The similarity and repetitiveness of the steps outlined in the previous section suggests that most of the algorithms in literature – as well as all the other configurations generated by the combination of the different options in each step – can be compared within a single environment. For this reason we developed a mobile application for Android smartphones, using Java as a programming language, in which a user can compose his or her own pedometer by choosing, for each step, which option to use. The application is open source and available on Github[2]. The application implements all the options for the steps shown in Figure 1, with the exception of the offline aspects. Users can combine the options into a pipeline by choosing one of them for each step, thus creating a **pedometer configuration**, which can be saved and reused later. At this point, users can use the application in two main ways: *Live testing* and *Offline testing*. In the first case, once a specific configuration is selected, the application starts sampling sensor data and shows in a separate screen the accelerometer data series updated in real time as well as a counter for the number of steps detected by then, also updated in real time. In the second case, users can test multiple configurations on the same data set in order to compare their performance. First of all, users can "record a trace", meaning that, with a specific function of the application, they can collect sensor data for a certain period of time and export it for later in a special trace file within the application folder. Users can also import traces from other files using the File Manager of the smartphone. Finally, the offline test feature allows users to select a number of pre-saved configurations and test them all together over a single trace. Figure 2 shows a screenshot of the application performing an offline test over a single trace using three different configurations. The application also shows the exact points in the associated wave where the different configurations detected a step. By knowing the ground truth, it is possible to evaluate the performance of different configurations over a single trace.

[2]https://github.com/GiacomoNeriUnibo/Pedometers

TABLE I
List of Tested Configurations

| Test | Detection Algorithm | Filter | Cutoff freq. |
|---|---|---|---|
| I | Peaks | No filter | — |
| II | Peaks + x-axis intersect. | No filter | — |
| III | Peaks | Rotation m. | — |
| IV | Peaks + x-axis intersect. | Rotation m. | — |
| V | Peaks | Bagilevi | — |
| VI | Peaks + x-axis intersect. | Bagilevi | — |
| VII | Peaks | low-pass | 2Hz |
| VIII | Peaks + x-axis intersect. | low-pass | 2Hz |
| IX | Peaks | low-pass | 3Hz |
| X | Peaks + x-axis intersect. | low-pass | 3Hz |
| XI | Peaks | low-pass | 10Hz |
| XII | Peaks + x-axis intersect. | low-pass | 10Hz |
| XIII | Peaks | low-pass | 2% samp. freq. |
| XIV | Peaks + x-axis intersect. | low-pass | 2% samp. freq. |

## V. Testing and Results

Using the application described in section IV, we compared the accuracy of different pedometer configurations on the same data sample. Specifically, 11 different human testers had to perform 6 different walks, of 50 steps each, recording a different test individually. In addition, we instructed each tester to hold the device in *texting* mode and to turn off device notifications to avoid vibrations that could compromise the data recorded by the sensors. The traces recorded were then analyzed using all the 14 different pedometer configurations obtainable by combining all implemented options (details of every configuration are shown in Table I). This section discusses the result of the different combinations between algorithms and filters for all walking styles. A sample of 11 different people, with an age ranging from 22 to 26, averaging at 24, was used to record the different tests. Detailed information about the different testers is summarized in Table

TABLE II
LIST OF THE TESTERS

| Tester | Device | Age |
|---|---|---|
| I | Xiaomi Redmi Note 5 | 25 |
| II | Xiaomi Redmi Note 5 | 24 |
| III | Xiaomi Redmi S2 | 23 |
| IV | Samsung Galaxy S8 | 25 |
| V | Samsung Galaxy A40 | 24 |
| VI | Samsung Galaxy A40 | 24 |
| VII | Honor View 20 | 26 |
| VIII | Honor 10 | 22 |
| IX | Huawei P20 Lite | 23 |
| X | Google Pixel 4a | 23 |
| XI | Asus Zenfone 5Z | 25 |

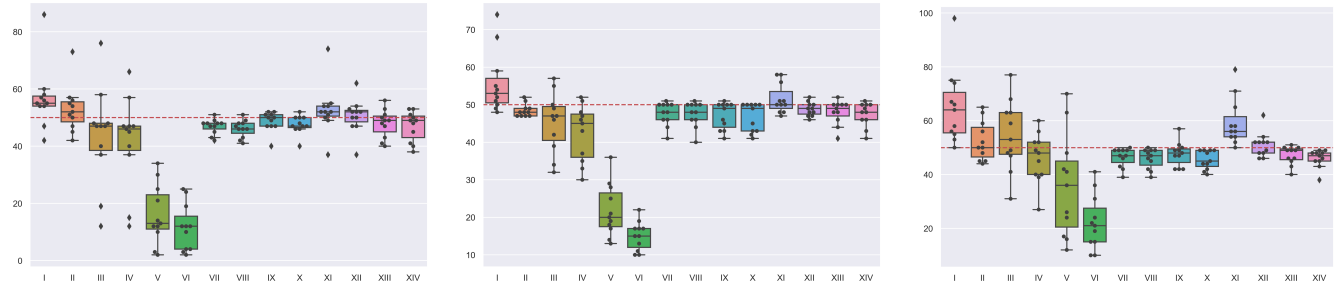
II. A total of 9 Android devices, belonging to 6 smartphone brands, were used to conduct the tests. The use of several distinct devices made it possible to obtain even more reliable indications of the effectiveness of the various algorithms since they are equipped with different physical sensors. Figure 3(a) shows the data collected during the tests performed by a plain walk. We can see that Bagilevi's algorithm ultimately gives disappointing results, while the No Filter and Rotation Matrix are much more accurate, as their average approaches the value of 50. However, their amplitude denotes inconsistent results, especially in the case of the Rotation Matrix, in which outliers are frequently recorded. The low-pass filter provides optimal results in each case, mainly when used with a low cutoff frequency. In all cases, using the $x$-axis Intersection algorithm did not lead to any particular improvement, except when using No Filter. In Figure 3(b), uphill walk, the data collected are very similar to those in Figure 3(a) but without outliers. The Rotation Matrix provides slightly worse results than the configurations without a filter, which perform a little better, especially in combination with the $x$-axis Intersection algorithm. Moving to Figure 3(c), downhill walk, the observations regarding Bagilevi's algorithm are still in line with the previous analysis, while the use of No Filter and the Rotation Matrix provide much less consistent results in this case, as their representation is much broader. A low-pass filter is the most accurate method, except when used with a cutoff frequency of 10Hz, as in this case, it provides slightly overestimated results. Figure 3(d), on the other hand, represents the data collected during the tests by performing a walk with irregular steps; as seen from the image, this walking style challenged most algorithms, highlighting their limitations. Using the Rotation Matrix and no filter generated very uneven results, slightly corrected by using the $x$-axis Intersection algorithm. Again, using the low-pass filter seems to have generated the most accurate results, if often slightly underestimated. Figure 3(e), concerning the data collected during the baby steps walk, also challenged most algorithms. Since the results are similar to those shown in Figure 3(d), we refer the reader to the data analysis was done for that plot. In the last plot, Figure 3(f), representing the run, we can see how the algorithms calculated inaccurate results, especially when using No Filter and the Rotation Matrix. In both cases, however, the $x$-axis Intersection algorithm provided a particularly accurate correction, giving much more precise and compact results. Bagilevi's algorithm is the least accurate, while using the low-pass filter presents the best results, especially with 2% of the sampling rate cutoff frequencies. In general we can observe how, in all cases, avoiding to use a filter leads to a number of false positives due to the noise that has not been attenuated. In graph 4, we have an overview of the overall accuracy of all 14 configurations. The diagram shows the average error that corresponds to the average of the distances between the recorded data and the ground truth 50, calculated in absolute value. The closer the value is to zero, the more accurate results the corresponding configuration provides. We can observe that Bagilevi's algorithm is, in every situation, the worst, as it gives highly inconsistent results that are almost always utterly distant from the correct value and is subject to an exceptionally high average error. On the other hand, it is evident that the low-pass filter is the best in almost all cases, regardless of the chosen cutoff frequency. Using the $x$-axis Intersection algorithm as a corrective generates relevant improvements when used in configurations related to No Filter and the Rotation Matrix. At the same time, it is not particularly relevant when used in configurations related to the low-pass filter. We can see that irregular and baby steps walks caused the most significant difficulties; unfortunately, in these two cases, the data produced by the sensors are highly peculiar, and processing them becomes a particularly complex task. Again, the most reliable configurations were those exploiting the low-pass filter. Ultimately, the best configuration is undoubtedly the one that uses the low-pass filter with a cutoff frequency equal to 10Hz, calculated with the Peak algorithm and corrected with the Intersection algorithm. The results are consistent with each other and almost always close to the correct value, generating almost no outliers and the absolute lowest average error value.

## VI. Conclusion

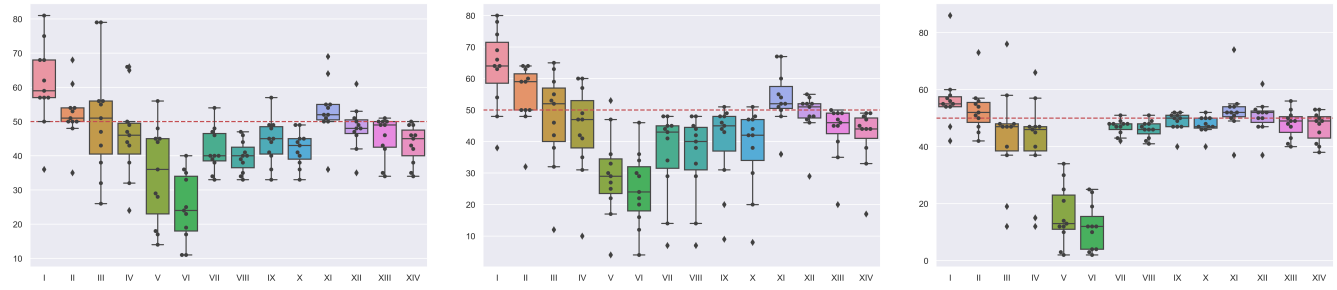
In this work we provided a basis for a thorough comparison analysis on software pedometers for IoT devices. We first analyzed the current literature and highlighted common aspects of existing implementations by identifying a common structure and proposing a taxonomy that builds on such a structure. We then used the taxonomy to develop a mobile application in order to evaluate all the existing implementation within a common environment, highlighting the utility of certain features over others depending on the activity of the user, in particular the importance of applying a coherent filter to attenuate the data noise. This work opens up many possibilities for future works, such as including offline algorithms as well as including existing applications.

## References

[1] K. Orr, H. S. Howe, J. Omran, K. A. Smith, T. M. Palmateer, A. E. Ma, and G. Faulkner, "Validity of smartphone pedometer applications," *BMC research notes*, vol. 8, no. 1, pp. 1–9, 2015.



(a) Plain walking (b) Uphill walking (c) Downhill walking



(d) Irregular steps (e) Baby steps (f) Running

Fig. 3. Boxplots showing the performance of the fourteen tested algorithms over six different walking paces tested by eleven different individuals each owning a different smartphone. The ground truth is set to 50.

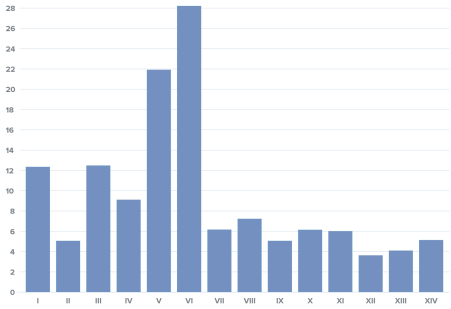


Fig. 4. Bar chart showing the average error of the fourteen tested configurations.


[2] P. S. Navarro Morales, "Smart movement detection for android phones," 2016.

[3] M. Khedr and N. El-Sheimy, "A smartphone step counter using imu and magnetometer for navigation and health monitoring applications," *Sensors*, vol. 17, no. 11, p. 2573, 2017.

[4] S. Jayalath and N. Abhayasinghe, "A gyroscopic data based pedometer algorithm," in *2013 8th International Conference on Computer Science Education*, 2013, pp. 551–555.

[5] A. K. Siddanahalli Ninge Gowda, S. R. Babu, and D. C. Sekaran, "Umoisp: Usage mode and orientation invariant smartphone pedometer," *IEEE Sensors Journal*, vol. 17, no. 3, pp. 869–881, 2017.

[6] A. Suprem, V. Deep, and T. Elarabi, "Orientation and displacement detection for smartphone device based imus," *IEEE Access*, vol. 5, pp. 987–997, 2017.

[7] D. Salvi, C. Velardo, J. Brynes, and L. Tarassenko, "An optimised algorithm for accurate steps counting from smart-phone accelerometry," in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2018, pp. 4423–4427.

[8] B. Wang, X. Liu, B. Yu, and R. Jia, "Posture recognition and adaptive step detection based on hand-held terminal," in *2018 Ubiquitous Positioning, Indoor Navigation and Location-Based Services (UPINLBS)*, 2018, pp. 1–5.

[9] N. Strozzi, F. Parisi, and G. Ferrari, "A novel step detection and step length estimation algorithm for hand-held smartphones," in *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2018, pp. 1–7.

[10] X. Yang and B. Huang, "An accurate step detection algorithm using unconstrained smartphones," in *The 27th Chinese Control and Decision Conference (2015 CCDC)*, 2015, pp. 5682–5687.

[11] W. Hongman, Z. Xiaocheng, and C. Jiangbo, "Acceleration and orientation multisensor pedometer application design and implementation on the android platform," in *2011 First International Conference on Instrumentation, Measurement, Computer, Communication and Control*, 2011, pp. 249–253.

[12] M. Pan and H. Lin, "A step counting algorithm for smartphone users: Design and implementation," *IEEE Sensors Journal*, vol. 15, no. 4, pp. 2296–2305, 2015.

[13] G. Liu, "Design and implementation of pedometer based on g-sensor," in *2018 3rd International Conference on Smart City and Systems Engineering (ICSCSE)*, 2018, pp. 436–438.

[14] M. Oner, J. A. Pulcifer-Stump, P. Seeling, and T. Kaya, "Towards the run and walk activity classification through step detection - an android application," in *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2012, pp. 1980–1983.

[15] W. Kang, S. Nam, Y. Han, and S. Lee, "Improved heading estimation for smartphone-based indoor positioning systems," in *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, 2012, pp. 2449–2453.

[16] T. O. Oshin and S. Poslad, "Ersp: An energy-efficient real-time smartphone pedometer," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, 2013, pp. 2067–2072.

[17] F. Gu, K. Khoshelham, J. Shang, F. Yu, and Z. Wei, "Robust and accurate smartphone-based step counting for indoor localization," *IEEE Sensors Journal*, vol. 17, no. 11, pp. 3453–3460, 2017.

[18] B. Presset, B. Laurenczy, D. Malatesta, and J. Barral, "Accuracy of a smartphone pedometer application according to different speeds and mobile phone locations in a laboratory context," *Journal of Exercise Science & Fitness*, vol. 16, no. 2, pp. 43–48, 2018.

[19] J. Seo and T. H. Laine, "Accurate position and orientation independent step counting algorithm for smartphones," *Journal of Ambient Intelligence and Smart Environments*, vol. 10, no. 6, pp. 481–495, 2018.