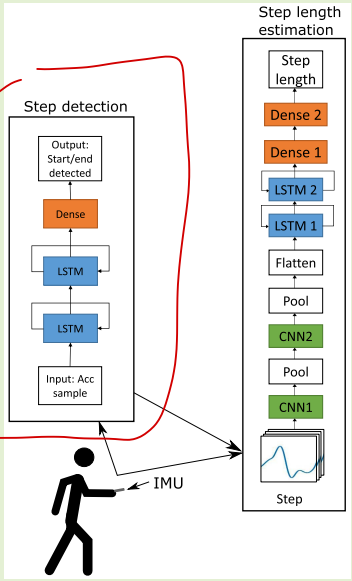
Deep-Learning-Based Step Detection and Step Length Estimation With a Handheld IMU

Stef Vandermeeren and Heidi Steendam, *Senior Member, IEEE*

***Abstract*—A popular approach to track the position of a user is to use an inertial measurement unit (IMU), which allows tracking a user with a pedestrian dead reckoning (PDR) system by estimating the length and heading of each step a user takes. Furthermore, a user's steps together with their length can be used to detect anomalies in the gait due to, e.g., a gait disorder. To determine the length and/or heading of a step, we first need an algorithm that estimates the boundaries of a step, i.e., the start and end, in the IMU signals. In this article, we propose a deep-learning-based step detection algorithm that detects the start and end of a step using only data from a handheld accelerometer. In this algorithm, also, the relationship between the start and end of a step is used to improve performance. To evaluate the performance of the detector, in contrast to most previous works that only look at the total number of detected steps, we determine if the start and end of each step are detected at the correct instant. This resulted in a step detector with an f-score of 99.2% and 99.0% for, respectively, detecting the start and end of a step. After detecting the start and end of the steps, we use a long short-term memory (LSTM)/convolutional neural network (CNN)-based step length estimator, which resulted in a mean absolute error (mae) on the step length of 3.21 cm. Finally, we also determine the combined performance of the step detector and step length estimator, i.e., where the proposed step detector is used to extract the accelerometer fragments of a step instead of assuming that the boundaries are known. This resulted in an mae of 6.56 cm using the deep-learning-based step length estimator.**



***Index Terms*— Deep learning, inertial measurement unit (IMU), pedestrian dead reckoning (PDR), step detection, step length.**

I. Introduction

DURING the last years, a lot of research has been carried out on indoor positioning techniques. These techniques can provide valuable information to a very broad range of applications, e.g., in emergency situations, such as a fire, but also in situations where a user needs to find his/her way inside a large building, such as a shopping center or an airport. Multiple positioning technologies, e.g., ultrawideband (UWB), Wi-Fi, and Bluetooth low energy (BLE), determine the position of a user by estimating the distance to certain reference points spread through the environment. In general, this results in fairly accurate position estimates, but a drawback of these technologies is that they require dedicated devices distributed over the area in which we want to track the user. Hence, the larger this area, the more devices are needed, and the higher the deployment cost. Furthermore, the accuracy drops when the line-of-sight between the user and one or more reference points is obstructed by, e.g., a metal rack. Finally, the positioning update decreases for an increasing number of users that need to be localized.

A promising technique that does not suffer from these drawbacks uses an inertial measurement unit (IMU) in a pedestrian dead reckoning (PDR) application [1], [2], [3], [4], [5], [6]. In a PDR application, a user's position is tracked incrementally by detecting his/her steps and estimating the length and heading of those steps. Unlike other positioning techniques, such as, e.g., Wi-Fi, UWB, BLE, or camera-based positioning [7], [8], a PDR application is self-contained, meaning that it does not rely on additional infrastructure in the environment. A disadvantage of a PDR system, however, is that it only provides relative position updates and that, due to errors in step length and heading, the estimated position starts to drift from the true position. Hence, in practice, PDR

Manuscript received 31 August 2022; revised 18 October 2022; accepted 21 October 2022. Date of publication 9 November 2022; date of current version 14 December 2022. This work was supported in part by the Excellence of Science (EOS) under Grant 30452698 from the Belgian Research Councils, Fonds Wetenschappelijk Onderzoek (FWO) and Fonds de la Recherche Scientifique (FNRS), and in part by the Flemish Government (AI Research Program). The associate editor coordinating the review of this article and approving it for publication was Dr. Yulong Huang. *(Corresponding author: Stef Vandermeeren.)*

The authors are with the Department of Telecommunications and Information Processing, Ghent University, 9000 Ghent, Belgium (e-mail: stef.vandermeeren@ugent.be; heidi.steendam@ugent.be).

Digital Object Identifier 10.1109/JSEN.2022.3219412

is often combined with another positioning technique, such as UWB, to provide the initial position and the absolute position estimates to correct the drift. The advantage of this combination is that it allows for a position estimate that remains accurate over time and requires a lower number of reference points (lower cost). In addition, the positioning update rate is now determined by the step frequency and, hence, does not decrease for an increasing number of users as for UWB.

To design a PDR system, several subsystems are required. First, we must detect each step a user takes together with the instant of the start and end of that step. Next, we use this start and end to extract the data from the IMU during this step. Finally, we use this data to estimate the length and heading of the step, which then allows for updating the position of the user. In this article, we focus on step detection and step length estimation, as these also can be used in other applications, where heading information is less important, such as activity trackers or detecting gait disorders [9], [10].

A. Step Detection

The first phase of a PDR algorithm is to detect the steps in the signals generated by the IMU. In the literature, several methods can be found to detect/count the steps of a user. A first family of methods employs an ad hoc algorithm based on the periodic pattern in the measured acceleration to detect steps [11], [12], [13], [14], [15], e.g., with peak or zero-crossing detection. A drawback of these algorithms is that they all have some parameters that need to be tuned to the specific user, which limits their usefulness. To avoid parameters that require tuning, a common approach is to use machine learning [16], [17], [18], [19] algorithms. A drawback of machine learning, however, is that it requires the extraction of useful features from the raw accelerometer and/or gyroscope data.

To solve this problem, nowadays, commonly deep learning is applied, which is capable of automatically extracting useful features from the raw data. Edel and Koppe [20] use bidirectional long short-term memory recurrent neural networks (BLSTM-RNNs) to detect steps. As the authors use a bidirectional network, the neural network is able to use information from previous and future time steps of the input time series. This, however, has the downside that the entire time series needs to be known before we can estimate the output so that online processing is impossible. Luu et al. [21] compare a long short-term memory (LSTM) network, a convolutional neural network (CNN), and a WaveNet for step detection. To detect a step, the authors use these networks to predict for a window of acceleration data if the last sample in the window is from a left or right step. By applying a sliding window approach, the steps are then detected when the output changes from a left to a right step or vice versa. In [22], a similar approach is followed, and an LSTM network is trained on accelerometer and gyroscope data to predict for each sample if it is part of a left or right step. An advantage of this approach over the approach of [21] is that, in [22], it is possible to make a new prediction using only one new sample of the accelerometer and gyroscope, while, in [21], an entire window of 2–4 s needs to be processed. A drawback of both works, however, is that, as the networks can only output a left or right step, it is not possible to detect when a user temporarily stands still. In [23], another approach is used, where a CNN estimates the number of steps in a window of acceleration data. Using a sliding window, the total number of steps together with an estimate of their occurrence is then estimated. A problem with this approach, however, is that the accuracy with which the timestamp of a step is detected depends on the step size of the sliding window. Furthermore, it is possible that, although a large part of the acceleration data of a step is part of a window, it does not contribute to the step count of that window as a step is only counted at the end.

Common to most of these works is that the authors only look at the total number of detected steps to determine the performance of the detector and do not verify if each step is detected at the correct time. However, as the step length and heading are extracted from the IMU data during a step, it also is important to verify if the start and end of steps are detected at the correct instant. In addition, the discussed works do not apply postprocessing to the output of the deep learning networks, which could further improve the performance. For example, when the network only shortly (wrongly) predicts a left step during a right step, the step detection network detects this as an extra erroneous step, while, with some postprocessing, this extra step can be filtered out.

B. Step Length

The next phase in a PDR algorithm estimates the length of a step. Several algorithms were already considered for determining the step length. In [24], [25], [26], and [27], the step length is written as a function of variables that are extracted from the accelerometer and, in some papers, also the gyroscope data, and involve user-specific parameters that need to be tuned. A drawback of these approaches is that, while the step length estimator may work well for one user, it might perform badly for another user without retuning some parameters. To avoid user-specific tuning of parameters, a more accurate and currently popular approach is to use deep learning algorithms for step length estimation. Hannink et al. [28] consider a CNN to estimate the step length. The input of this network consists of the measured acceleration and angular rate of two foot-mounted IMUs. This approach, however, requires two IMUs: one for each foot, to estimate the length of a step, and another device, e.g., a smartphone, to collect and process the data of the two sensors. A similar approach is followed in [29], but, instead of a CNN, a bidirectional LSTM network is used. Wang et al. [30] use an LSTM network in combination with denoising autoencoders (DAEs), where the LSTM is used to extract useful features from the input, and the DAE is used to denoise these features. In comparison with the other methods, however, this approach is relatively complex as it uses the accelerometer and gyroscope data in combination with some manually extracted features and requires three separate training phases. Our previous work [31] also considered step length estimation. In that work, we developed a method to systematically build the feature set for multiple machine learning algorithms that were used to estimate the step length using an accelerometer only. A drawback of this work is that the performance of the step length estimator depends on the initial feature set—extracted from the measured acceleration—from which the feature selection algorithm selects the best

features. If the initial set contains many features that are useless to estimate the step length, the performance of the step length estimator can be bad. Common to all these works is that the authors assume that the boundaries of a step are known, while this information is not a priori available in practical PDR systems.

In recent works [32], [33], [34], [35], an increasingly popular approach for a PDR system is to divide the accelerometer and gyroscope data into fragments of a fixed length. Using these fragments, deep learning is then trained to directly output the displacement and heading (change) during this fragment. Hence, the need for a step detector is eliminated. A drawback, however, is that this approach eliminates the possibility to analyze the gait of a user or count the total number of steps a user has made. Therefore, we do not use this approach so that the step detector and step length estimator can be used for different applications. In addition, using fragments of fixed length has a potential drift problem when the user is standing still as, in this case, nonzero predictions for the step length and heading change will lead to a growing error in the position estimate.

In this work, we present two contributions, i.e., a deep-learning-based step detection and step length estimation algorithm using a single handheld IMU. As we do not consider heading estimation, we only provide the accelerometer data to the deep learning networks and not the gyroscope data as gyroscopes generally consume much more power than an accelerometer. An advantage of this choice is that the presented algorithms are better suited when low power consumption is paramount and no heading information is required. The gyroscope data can, however, be easily included by including this data in the input to the neural networks.

First, we develop a long-short-term-memory-based step detection algorithm that determines the start and end of each step. As opposed to [17], due to the use of an LSTM network, the step detector can readily handle steps that do not have the same duration. Other advantages of our approach are that the algorithm does not depend on user-dependent parameters as in [11], [12], [13], [14], [15], and [16], and we determine the start and end of each step separately. The latter makes it possible to use the relation between the start/end of a step to detect missing or erroneous steps, extract the acceleration fragments for a certain step, and also detect pauses between steps. This is not the case if we only detect when a step is completed, i.e., the end of a step, which is done in most other machine-learning-based approaches. In those approaches, it is silently assumed that the end of the previous step is also the start of a new step, and hence, the pause is included in the accelerometer samples of the detected step. Therefore, in this work, we also propose an algorithm that postprocesses the output of the LSTM and leverages the relation between a start and end of a step so that the step detection performance is further improved. In contrast with other works that determine the performance of a step detector by only looking at the number of detected steps, in this work, we also evaluate if the steps are detected at the right instant.

Second, we present a deep learning network for step length estimation that combines the LSTM and CNN architectures so that the network is capable of automatically extracting useful features from the raw acceleration data using the CNN while also being able to leverage temporal patterns with the LSTM network. To train this network, we assume that the start and end of a step are known, and compare its performance with the step length estimator of our previous work and a state-of-the-art approach [28]. Finally, in contrast with most other works that only consider the true steps to evaluate the step length estimate, we also combine the step detection and step length estimation algorithms and evaluate the performance degradation of the step length estimator due to the estimated start and end of a step.

The rest of this article is organized as follows. In Section II, we discuss the step boundary detection algorithm. First, we introduce the topology of the deep learning network that estimates for each sample of the accelerometer how likely a step started or ended, and we elaborate on the training of the network. Next, we use the output of this network to extract the time instants at which a start or end of a step was detected and present how the relationship between the start and end of a step can be used to improve the performance of the step detector. In Section III, we describe the deep learning network that is used to estimate the length of a step. Then, the results are discussed in Section IV, where we evaluate the performance of the step boundary detector and step length estimator separately, as well as their combined performance. In addition, we also compare the proposed algorithms to other state-of-the-art algorithms. Finally, conclusions will be given in Section V.

II. Step Boundary Detection

In this section, we introduce the deep learning algorithm to detect the boundaries of a step, i.e., the start and end instant, from the measured acceleration of a handheld phone in a texting position. In Fig. 1, we show an example of the magnitude of the measured raw acceleration and the filtered acceleration, low-pass filtered with a third-order Butterworth filter with a cutoff frequency of 3 Hz, which correspond to four steps together with the boundaries of these steps. This cutoff frequency was chosen so that we remove as much as possible of the higher frequency components in the acceleration that do not correspond to taking a step and are, thus, not useful for step detection. Comparing both curves in Fig. 1, it is clear that it is much easier to extract the four steps from the filtered acceleration, and using the raw acceleration can result in false step detections. From the filtered acceleration magnitude, we can easily identify the four steps and observe that the boundaries for each of these steps can be found by detecting instants where the acceleration magnitude crosses 1 g with a positive slope. Hence, in theory, the step boundaries can easily be found. However, noise and/or acceleration not related to taking steps (e.g., shaking the IMU) can result in more instants where the magnitude crosses 1 g. An example of this can also be seen in Fig. 1 around 0.4 s, caused by some small arm movements before the test person started to walk.

In previous works, to limit the number of erroneously detected 1-g crossings, an interval between two 1-g crossings was detected as a step only if the maximum (minimum) acceleration during the interval was higher (lower) than a certain

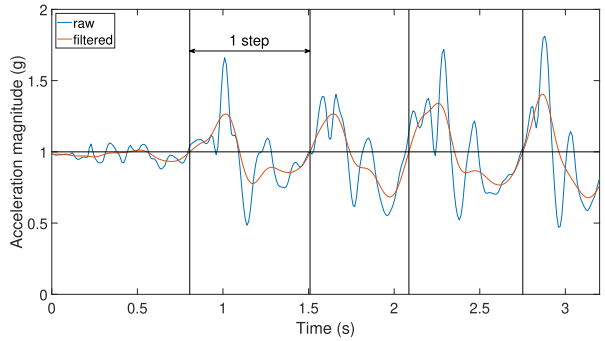


Fig. 1. Example of the raw and low-pass filtered acceleration magnitude for a handheld accelerometer with the boundaries for each step.

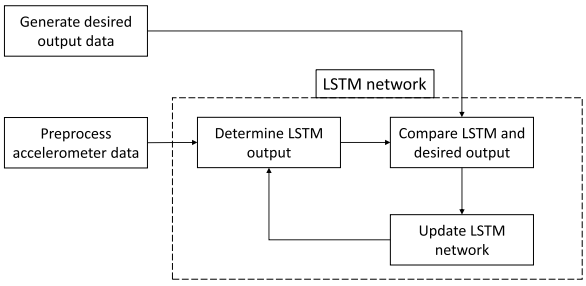


Fig. 2. Block diagram of the training phase.

threshold. The problem with this approach is that the optimal value of these thresholds is user-dependent and, hence, needs to be tuned. For this reason, we want to apply machine learning in the proposed step detector as this can eliminate the need for tuning, provided that the machine learning algorithm is trained on multiple users. Regular machine learning algorithms, such as decision trees, support vector machines, or feedforward neural networks, convert a fixed-length input to a fixed-length output, but, in general, steps do not have the same duration. Although we can solve this fixed-length problem by dividing the raw accelerometer data into shorter fragments with a fixed size, such an approach would not allow us to take into account possible relationships between subsequent fragments of the accelerometer data, resulting in a loss of valuable information.

To tackle the problem of variable length fragments, recurrent neural networks (RNNs) were developed. This type of neural network can handle input data of variable size and is able to detect temporal patterns in the input data by also taking into account information learned from the previous inputs. In this work, we employ an RNN based on the LSTM architecture, which is designed to learn long-term dependencies. We will use the LSTM network to separately predict for each sample of the accelerometer if a step started and/or ended.

In Section II-A, we will provide more details about the architecture of the LSTM network and how it is trained to obtain the desired output. The output of the LSTM network consists of two sequences that provide potential candidate samples for the start/end of a step, respectively. In Section II-B, we will discuss how to extract the start/end time instants of a step from the two output sequences generated by the LSTM network. To this end, we use the relationship between the start and end points of a step to combine the information in order to identify if some starts/ends are missed or detected erroneously. For example, when only a start was detected but no matching end of a step, either the start was wrongly detected or the LSTM network missed the end of the step.

A. LSTM Network

This section discusses the design of the LSTM network that we use to predict if a sample of the accelerometer is the start and/or end of a step. In Fig. 2, we show a schematic representation of the structure of the LSTM network in the training phase. In the remainder of this section, we discuss the different parts of the block diagram in more detail.

1) Preprocessing the Data: The accelerometer data consist of the raw acceleration $\boldsymbol{a} = [a_x, a_y, a_z]$, where a_α is the α component of the measured acceleration with $\alpha \in [x, y, z]$, from the handheld phone in texting position sampled at 100 Hz. To limit the effect of the noise of the accelerometer, we first smoothen the measured acceleration sequence with a third-order low-pass Butterworth filter with a cutoff frequency of 3 Hz, which results in the filtered acceleration sequence $\tilde{\boldsymbol{a}} = [\tilde{a}_x, \tilde{a}_y, \tilde{a}_z]$, where $\tilde{a}_\alpha$ is the Butterworth filtered version of a_α with $\alpha \in [x, y, z]$. In addition, we also determine the magnitude $|\tilde{\boldsymbol{a}}|$ of the filtered acceleration with $|\tilde{\boldsymbol{a}}| = (\tilde{a}_x^2 + \tilde{a}_y^2 + \tilde{a}_z^2)^{1/2}$. To lower the training time and improve the performance of the LSTM network, we scale the acceleration using a MinMaxScaler so that the scaled values of the acceleration are approximately between zero and one. This results in the scaled accelerations $|\hat{\boldsymbol{a}}|$ and $\hat{a}_\alpha$, which are, respectively, the scaled acceleration of the magnitude and the α component with $\alpha \in [x, y, z]$. For each sample of the accelerometer, we provide the scaled acceleration sample, i.e., a 4-D vector $\hat{\mathbf{a}}_i = [\hat{a}_{x,i}, \hat{a}_{y,i}, \hat{a}_{z,i}, |\hat{\boldsymbol{a}}|_i]^T$, where $|\hat{\boldsymbol{a}}|_i$ and $\hat{a}_{\alpha,i}$, respectively, correspond to the ith sample of the magnitude and the α component of the scaled acceleration with $\alpha \in [x, y, z]$, as an input to the LSTM network. To train the network, the acceleration data are then divided into fragments of length n_s, i.e., the $4 \times n_s$ tuples $\mathbf{a}_j^{n_s} = [\hat{\mathbf{a}}_{j,1}, \ldots, \hat{\mathbf{a}}_{j,n_s}]$, where j denotes the jth fragment and $\hat{\mathbf{a}}_{j,\ell} = \hat{\mathbf{a}}_{jn_s+\ell}$, as training data for the network. In the numerical results section, we will further discuss the selection of this parameter.

2) Desired Output: Let us look closer at the desired output. Ideally, the desired output sequences—s_{ideal}: start of the step and e_{ideal}: end of the step—equal one at the instants of the true start and end of steps, respectively, i.e., at the instants where the acceleration magnitude crosses the 1-g threshold with a positive slope and zero otherwise (see Fig. 3). This ideal desired output, however, strongly restricts the potential start/end samples and will result in a high loss when the start/end of a step is not detected at the exact sample. This will make the network hard to train. Therefore, we relax our constraints on the desired sequence by making the desired output equal to one for some period around the true start/end of a step. In this way, the LSTM algorithm learns that the prediction of the start/end of a step does not have to be perfect, as long as it is close to the true start/end. In our simulations,

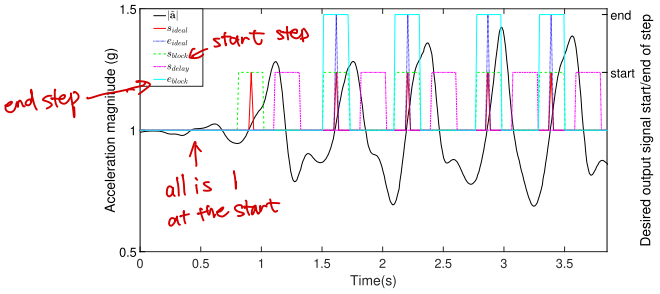


Fig. 3. Example of a fragment of $|\tilde{a}|$ together with different desired output sequences of the LSTM network. s_{block}, s_{delay}, and e_{block} are "1" during 0.2 s, and s_{delay} contains a delay of $d = 0.3$ s.

we found that an interval of 21 samples, i.e., ten samples before and after the start/end of a step, results in the best performance, which corresponds to an uncertainty interval of 0.1 s at 100 Hz. This altered output is also shown in Fig. 3 as s_{block} (start of step) and e_{block} (end of step).

The LSTM network evaluates sample per sample of a fragment, and based on the already processed samples, it needs to determine if the sample corresponds to a start or end of a step. It is clear that, by only using preceding acceleration samples, and not knowing the successive samples, the determination of the start of a step is not obvious. To solve this issue, we suspend the determination of the start of a step until we have sufficient information to take the decision. This is achieved by adding a fixed delay d to the desired output (s_{delay}), as illustrated in Fig. 3. As long as this delay is smaller than the duration of a step, this delay does not impact the real-time character of the decision process. Note that such a delay is not necessary for the decision process of the end of a step: the end of a step can immediately be predicted when the last sample of the step is delivered to the LSTM network. The resulting sequences $s_{\text{desired}} = s_{\text{delay}}$ and $e_{\text{desired}} = e_{\text{block}}$ are then used to train the network.

3) LSTM Algorithm: In Fig. 4, we show the architecture of the LSTM network, where we unfold the network for each of the n_s samples of an accelerometer fragment $\boldsymbol{a}_j^{n_s}$. In this architecture, a row corresponds with a layer of the network and a column with a timestep/sample.

For the ℓth timestep of the network, $\ell = 1, \ldots, n_s$, the input consists of the ℓth preprocessed acceleration sample $\hat{\mathbf{a}}_{j,\ell}$ of fragment j. This acceleration sample $\hat{\mathbf{a}}_{j,\ell}$ is then applied to the ℓth cell of the first LSTM layer, containing n_h hidden states. The LSTM cell transforms the 4×1 input $\hat{\mathbf{a}}_{j,\ell}$ combined with the $n_h \times 1$ hidden states vector $\boldsymbol{h}_{\ell-1}$ produced by the previous cell into the $n_h \times 1$ output $\boldsymbol{h}_\ell$ of the cell. The resulting output is then forwarded to the next cell in the layer, as well as to the next layer for the same timestep. As there might be dependencies between subsequent fragments of the accelerometer $\boldsymbol{a}_j^{n_s}$, we use stateful LSTMs. This means that the hidden states $\boldsymbol{h}_{n_s}$ of the last sample in a fragment are passed as input $\boldsymbol{h}_0$ to the first LSTM cell of the next fragment in the same layer, provided that the next fragment directly succeeds the current fragment. Otherwise, we reset the hidden states vector $\boldsymbol{h}_0$ to an all zeros vector. The second LSTM layer uses as input the $n_h \times 1$ output vectors $\boldsymbol{h}_\ell$ from the previous layer and also contains n_h hidden states to produce the $n_h \times 1$ output vectors $\boldsymbol{h}'_\ell$. We restrict our network to two LSTM layers, as our simulations revealed that adding more layers in the network only increased the computational complexity, while not resulting in a significant performance improvement.

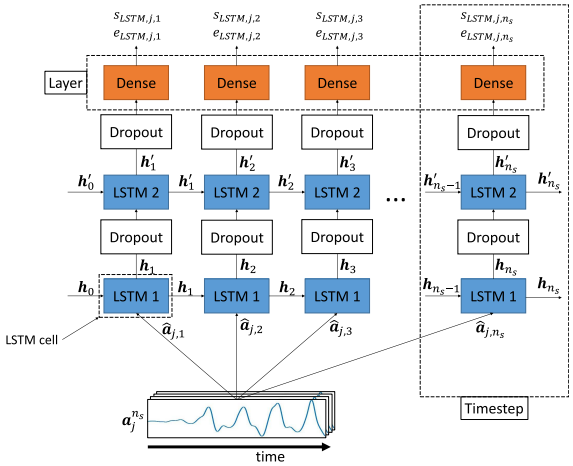


Fig. 4. Architecture of unfolded LSTM network that is used to predict how likely a sample is a start and/or end of a step.

To reduce the risk of overfitting, we add a dropout layer after each LSTM layer. For each epoch in the training phase, the dropout layer randomly selects a fraction p_{dropout} of the outputs of the preceding LSTM layer to be ignored by the next layer. As a result, the LSTM network learns to predict the desired output when it can only process part of the entire input. As the dropout layers are intended to prevent overfitting, they are included only during the training phase, but not during the test phase.

Finally, we use a dense (or fully connected) layer with a sigmoid activation function to transform the outputs of the second LSTM layer to the wanted output sequences, i.e., an output sequence s_{LSTM} corresponding to the start of the steps and an output sequence e_{LSTM} for the end of the steps. The sigmoid function ensures that the values of the output sequences are between 0 and 1. Hence, the outputs $s_{\text{LSTM},j,\ell}$ and $e_{\text{LSTM},j,\ell}$ of the LSTM network at the ℓth timestep of the jth fragment give an indication of the corresponding sample $\hat{\mathbf{a}}_{j,\ell}$ being (close to) the start or end of a step. In the results section, we will find the values for the parameters (n_h, n_s, p_{dropout}, d, and so on) that result in an LSTM network with good performance.

To train the network, the outputs s_{LSTM} and e_{LSTM} of the LSTM algorithm are compared with the desired outputs s_{desired} and e_{desired} using a binary cross-entropy loss function $\mathcal{L}_{\text{ce}}$, which is given by

$$\mathcal{L}_{\text{ce}} = \frac{\mathcal{L}_{s,\text{ce}} + \mathcal{L}_{e,\text{ce}}}{2} \tag{1}$$

where $\mathcal{L}_{\text{ce}}$ is the average of the binary cross-entropy $\mathcal{L}_{s,\text{ce}}$ for the output corresponding to the start of a step and the binary cross-entropy $\mathcal{L}_{e,\text{ce}}$ for the output corresponding to the end of a step. In (2) and (3), we give the expression for, respectively, $\mathcal{L}_{s,\text{ce}}$ and $\mathcal{L}_{e,\text{ce}}$, where n_f is the number of fragments in the

training set and n_s is the length of each fragment, as defined in Section II-A.1

$$\mathcal{L}_{s,\text{ce}} = -\frac{1}{n_s \cdot n_f}\sum_{i=1}^{n_s}\sum_{j=1}^{n_f}\left[s_{\text{desired},j,i}\cdot\log\left(s_{\text{LSTM},j,i}\right) + \left(1-s_{\text{desired},j,i}\right)\cdot\log\left(\left(1-s_{\text{LSTM},j,i}\right)\right)\right] \quad (2)$$

$$\mathcal{L}_{e,\text{ce}} = -\frac{1}{n_s \cdot n_f}\sum_{i=1}^{n_s}\sum_{j=1}^{n_f}\left[e_{\text{desired},j,i}\cdot\log\left(e_{\text{LSTM},j,i}\right) + \left(1-e_{\text{desired},j,i}\right)\cdot\log\left(\left(1-e_{\text{LSTM},j,i}\right)\right)\right]. \quad (3)$$

From these equations and the knowledge that $s_{\text{desired},j,i}$ and $e_{\text{desired},j,i}$ are either zero or one, it is clear that, if $s_{\text{LSTM},j,i} = s_{\text{desired},j,i}$ ($e_{\text{LSTM},j,i} = e_{\text{desired},j,i}$), the contribution to $\mathcal{L}_{s,\text{ce}}$ ($\mathcal{L}_{e,\text{ce}}$) is zero. Hence, the lower the loss $\mathcal{L}_{\text{ce}}$, the better the LSTM network predicts the start and end of a step.

A problem with the current definition of the loss $\mathcal{L}_{\text{ce}}$ is that the desired outputs s_{desired} and e_{desired} are equal to one during only a small part of a step and zero elsewhere. In our training set, the number of samples $N_{0,s}$ and $N_{1,s}$ for which s_{desired} is equal to zero or one are, respectively, $N_{0,s} = 137\,417$ and $N_{1,s} = 65\,583$. As each start of a step also has a corresponding end, the number of samples $N_{0,e}$ and $N_{1,e}$ for which e_{desired} is equal to zero or one is equal to, respectively, $N_{0,s}$ and $N_{1,s}$, i.e., $N_{0,s} = N_{0,e}$ and $N_{1,s} = N_{1,e}$. Hence, the contribution of the samples for which the desired output s_{desired} (e_{desired}) is equal to one to the loss $\mathcal{L}_{s,\text{ce}}$ ($\mathcal{L}_{e,\text{ce}}$) is smaller than for the samples for which the desired output s_{desired} (e_{desired}) is equal to zero. This implies that the network is biased to more accurately predict the zero output. To prevent that the network is not able to accurately predict the more seldom start/end of a step, we apply weight balancing, i.e., during training, samples for which s_{desired} (e_{desired}) are equal to zero contribute less to the loss function than samples for which s_{desired} (e_{desired}) is equal to one. With weight balancing, the losses $\mathcal{L}_{s,\text{ce}}$ and $\mathcal{L}_{e,\text{ce}}$ can be rewritten as

weight balancing

$$\mathcal{L}_{s,\text{ce}} = -\frac{1}{n_s \cdot n_f}\sum_{i=1}^{n_s}\sum_{j=1}^{n_f} w_{s,j,i} \cdot\left(s_{\text{desired},j,i}\cdot\log\left(s_{\text{LSTM},j,i}\right) + \left(1-s_{\text{desired},j,i}\right)\cdot\log\left(\left(1-s_{\text{LSTM},j,i}\right)\right)\right) \quad (4)$$

$$\mathcal{L}_{e,\text{ce}} = -\frac{1}{n_s \cdot n_f}\sum_{i=1}^{n_s}\sum_{j=1}^{n_f} w_{e,j,i} \cdot\left(e_{\text{desired},j,i}\cdot\log\left(e_{\text{LSTM},j,i}\right) + \left(1-e_{\text{desired},j,i}\right)\cdot\log\left(\left(1-e_{\text{LSTM},j,i}\right)\right)\right) \quad (5)$$

where $w_{s,j,i}$ and $w_{e,j,i}$ are the weights that are used to balance the loss. In (6) and (7), the definition of these weights is given, as well as their values for our training set. From these equations, we can see that samples with the desired output of one contribute approximately twice as much to the loss function as the samples with the desired output of zero

$$w_{s,j,i} = \begin{cases} \dfrac{N_{0,s}+N_{1,s}}{2\cdot N_{1,s}} = 1.548, & \text{if } s_{\text{desired}} = 1 \\ \dfrac{N_{0,s}+N_{1,s}}{2\cdot N_{0,s}} = 0.739, & \text{if } s_{\text{desired}} = 0 \end{cases} \quad (6)$$

$$w_{e,j,i} = \begin{cases} \dfrac{N_{0,e}+N_{1,e}}{2\cdot N_{1,e}} = 1.548, & \text{if } e_{\text{desired}} = 1 \\ \dfrac{N_{0,e}+N_{1,e}}{2\cdot N_{0,e}} = 0.739, & \text{if } e_{\text{desired}} = 0. \end{cases} \quad (7)$$

The weighted binary cross-entropy loss $\mathcal{L}_{\text{ce}}$ is then used to update the internal weights of the LSTM network with a gradient descent-based algorithm to improve, i.e., lower, the loss in the next epoch. In this article, we implemented the neural network in the Keras framework [36] and used the Adam optimizer [37] with a learning rate of 0.001 to update the internal weights of the network.

4) Performance Measures: Using the procedure explained above, we can train the LSTM network to determine the two outputs s_{LSTM} and e_{LSTM}, which, respectively, correspond to the output that indicates if an accelerometer sample is close to the true start and/or the end of a step. Both these outputs lie in the interval [0, 1], where a value close to zero indicates that the sample most likely is not a start or end, while a value close to one indicates that the sample is most likely a start and/or end of a step. After we updated the weights of the LSTM network using the binary cross-entropy loss, we now want to evaluate the performance with some more comprehensible metrics. To evaluate the performance of the LSTM network, we first determine the recall and precision on the output corresponding to the start and end of a step combined, i.e., on the concatenated output $b_{\text{LSTM}} = s_{\text{LSTM}} \frown e_{\text{LSTM}}$, where $\frown$ denotes the concatenation operator. The recall expresses how likely the network correctly predicts that a sample is a start (end) of a step, while the precision tells us how likely it is that, if the algorithm predicts a start (end), it actually was a start (end) of a step. Hence, the recall and precision can be determined with the following equation:

$$r = \frac{\text{tp}}{\text{tp}+\text{fn}} \qquad p = \frac{\text{tp}}{\text{tp}+\text{fp}}. \quad (8)$$

where r and p are, respectively, the recall and precision, tp is the number of true positives, fn is the number of false negatives, and fp is the number of false positives.

To find the number of true positives, false negatives, and false positives, we round the values in b_{LSTM} to one if they are larger than 0.5 and to zero in the other case, and check if the rounded values are equal to the desired outputs $b_{\text{desired}} = s_{\text{desired}} \frown e_{\text{desired}}$. From the recall and precision, we can then determine the f-score as in (9), which we use to evaluate the performance of the LSTM network. Note that a high f-score corresponds with high precision and recall and, hence, a good performance. For each epoch during the training phase, we keep track of the achieved f-score and save the

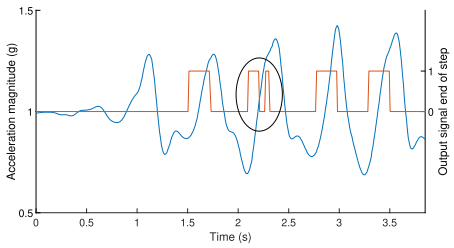


Fig. 5. Illustration of a fragment of the acceleration magnitude $|\tilde{a}|$ with the rounded output for the end of a step that could result in a misdetected end due to the short $1 \rightarrow 0 \rightarrow 1$ transition.

model for the epoch with the best f-score

$$\text{f-score} = 2\frac{r \cdot p}{r + p}. \tag{9}$$

So far, we developed an LSTM network that only indicates for each sample the confidence of being the start/end of a step. In Section II-B, we use the output of this deep learning network to extract/detect the actual time instants a step started and/or ended.

B. Combine Start and End of Steps

Now that we have the outputs s_{LSTM} and e_{LSTM}, we still need to extract the corresponding start and end boundaries, i.e., the time instants where a step started or ended. To do so, we first round these outputs to either zero or one using (10) and (11), where th denotes the threshold that we use to round s_{LSTM} and e_{LSTM}, which results in the rounded outputs $\hat{s}_{\text{LSTM}}$ and $\hat{e}_{\text{LSTM}}$

$$\hat{s}_{\text{LSTM},j,i} = \begin{cases} 1, & \text{if } s_{\text{LSTM},j,i} > \text{th} \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

$$\hat{e}_{\text{LSTM},j,i} = \begin{cases} 1, & \text{if } e_{\text{LSTM},j,i} > \text{th} \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

From these rounded outputs, we already can extract the step boundaries by detecting sequences of ones in the output, where each sequence results in a start or end of a step. However, in some scenarios, this approach will result in the erroneous detection of additional steps. An example of such a scenario is illustrated in Fig. 5, where a fragment of the acceleration magnitude is shown together with the rounded output $\hat{e}_{\text{LSTM}}$ for the end of a step. If we compare this with the true output, which is shown in the cyan curve of Fig. 3, we can see that the estimated ends are identical to the true ends except for the second step where the output shortly becomes zero. Hence, in this situation, detecting sequences of ones to find the end of a step would result in an extra (erroneous) end of a step. To avoid this, we first preprocess the rounded outputs $\hat{s}_{\text{LSTM}}$ and $\hat{e}_{\text{LSTM}}$ as follows: if the time between a $1 \rightarrow 0$ and $0 \rightarrow 1$ transition is smaller than a threshold $w_{1\rightarrow0\rightarrow1}$, we replace this with $1 \rightarrow 1$, i.e., we replace two sequences of ones separated by a short zero-sequence with one large sequence of ones. This step then results in the new outputs $\tilde{s}_{\text{LSTM}}$ and $\tilde{e}_{\text{LSTM}}$, and prevents that, if the output is zero for only a very short time, two separate starts or ends of a step would be detected instead of one. In the results section, we will find the optimal value of $w_{1\rightarrow0\rightarrow1}$.

Finally, from the preprocessed outputs $\tilde{s}_{\text{LSTM}}$ and $\tilde{e}_{\text{LSTM}}$, we extract the sets containing all the time instants where a step starts ($\Omega_{s,d}$) or ends (Ω_e). To this end, we look for sequences of ones in $\tilde{s}_{\text{LSTM}}$ ($\tilde{e}_{\text{LSTM}}$), i.e., we look for sequences where $\tilde{s}_{\text{LSTM},i_{\text{start}}:i_{\text{end}}} = 1$ ($\tilde{e}_{\text{LSTM},i_{\text{start}}:i_{\text{end}}} = 1$), where we omit the j subindex for clarity and with i_{start} and i_{end}, respectively, the start and end timestep of a sequence of ones. For each sequence, we then add the time instant of the start (end) of a step to $\Omega_{s,d}(\Omega_e)$ if the following conditions are met:

$$\begin{aligned} i_{\text{end}} - i_{\text{start}} &> w \\ \max\left(\alpha_{\text{LSTM}}\left(i_{\text{start}} : i_{\text{end}}\right)\right) &> \text{th}_{\max} \end{aligned} \tag{12}$$

where w is the minimal required length of a sequence of ones, α_{LSTM} is either s_{LSTM} or e_{LSTM} depending on whether we are determining the start or end instants of the steps, and $\text{th}_{\max}$ with $\text{th} \leq \text{th}_{\max} \leq 1$ is the threshold for the output of the LSTM network (before rounding) during a sequence of ones in the rounded output. The first condition makes sure that short bursts of ones do not result in a detected start or end of a step, while the second condition verifies that, for at least one sample of the sequence, the predicted (unrounded) output lies above $\text{th}_{\max}$. This ensures that the start or end is detected only when the LSTM network is confident enough that a step started or ended. If these conditions are met, the time instant that corresponds to the middle of the sequence of ones, i.e., $i_m = (i_{\text{start}} + i_{\text{end}}/2)$, is added to $\Omega_{s,d}$ or Ω_e. Finally, we still need to take into account the delay d that was introduced to the output for detecting the start of a step. To this end, we need to subtract the delay d from the time instants for the start of a step in $\Omega_{s,d}$, i.e., $\Omega_s = \{i - d \mid i \in \Omega_{s,d}\}$.

So far, we extracted two sets with the time instants where steps started or ended, i.e., Ω_s and Ω_e, from the output of the LSTM network. However, up until now, we did not consider the relationship between the start and end of the steps. For example, when we detect an end of a step before we have detected any start of a step, either the end of a step was detected incorrectly or the algorithm missed the start of a step. In the remainder of this section, we describe the algorithm that combines the information of the detected start and end of the steps into the sets that contain the final estimate of the step boundaries, which we will call $\Omega_{f,s}$ and $\Omega_{f,e}$, respectively.

First, we derive three sets Ω_1, Ω_2, and Ω_3 from Ω_s and Ω_e.

Ω_1 contains all starts of Ω_s for which no end of a step in Ω_e closer than a threshold m exists, where m is the threshold for the minimal duration of a valid step, while Ω_3 contains all ends for which no start of a step closer than m exists. Hence, ideally, Ω_1 contains only the first start of each sequence of steps, and Ω_3 contains only the last end of a step for each sequence. For Ω_2, we determine, for each start of a step, the closest end of a step, and if they are closer than m, we add the average of the start and end to Ω_2. In short, we look for instants where, at the same time, a step begins and ends, and put these in a separate set Ω_2. As during walking each step follows directly after the other, generally, Ω_2 will contain much more elements than Ω_1 and Ω_3. Then, we transform these three sets back into two (temporary) sets $\Omega_{s,t}$ and $\Omega_{e,t}$

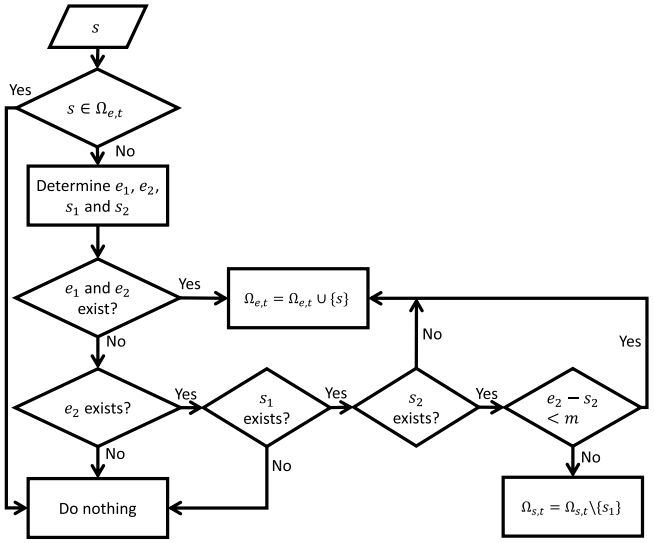


Fig. 6. Flowchart that is used to detect missing ends of a step or wrong starts of a step.

as in (13) and (14) and sort both sets in increasing order

$$\Omega_{s,t} = \Omega_1 \cup \Omega_2 \tag{13}$$

$$\Omega_{e,t} = \Omega_3 \cup \Omega_2. \tag{14}$$

Next, we go through these sets to determine if some starts or ends of a step are missing or detected incorrectly. In Fig. 6, a flowchart is shown, which illustrates how possibly missing ends of a step are added to $\Omega_{e,t}$ or incorrect starts of a step are removed from $\Omega_{s,t}$. To this end, we loop over all the elements $s \in \Omega_{s,t}$. In Fig. 6, e_1, e_2, s_1, and s_2 are defined as

$$e_1 = \{el \in \Omega_e \mid s - M < el < s - m\} \tag{15}$$

$$e_2 = \{el \in \Omega_e \mid s + m < el < s + M\} \tag{16}$$

$$s_1 = \{el \in \Omega_s \mid s - M < el < s - m\} \tag{17}$$

$$s_2 = \{el \in \Omega_s \mid s + m < el < s + M\} \tag{18}$$

with m and M, respectively, being the thresholds for the minimum and maximum durations of a valid step. A similar procedure is followed where we loop over all elements $e \in \Omega_{e,t}$.

As a final step, we again loop over all elements $s \in \Omega_{s,t}$ and determine the end $e \in \Omega_{e,t}$, for which $s+m < e < s+M$, i.e., we determine if there is an end in $\Omega_{e,t}$ that can be the end of a step with start s. If such an end of a step is found, we add s to $\Omega_{f,s}$ and e to $\Omega_{f,e}$. In Section IV, we test our method to detect the start and end instants of a step and evaluate the performance. To evaluate the performance of the algorithm, we compare the final sets $\Omega_{\text{f,s}}$ and $\Omega_{\text{f,e}}$ with the true sets $\Omega_{\text{true,s}}$ and $\Omega_{\text{true,e}}$ for, respectively, the start and end instants of all steps by determining the recall, precision, and the f-score for detecting the start and end instants of a step. To determine if the start (end) of a step is detected at a correct instant, for each element in $\Omega_{\text{f,s}}$ ($\Omega_{\text{f,e}}$), we determine the closest element in $\Omega_{\text{true,s}}$ ($\Omega_{\text{true,e}}$). If the difference between these two elements is smaller than $(m/2)$, i.e., the half of the minimal step length, the predicted start (end) instant of a step is assumed to be correct. In Table I, we, for clarity, summarize the variables used for step detection together with their description.

TABLE I
DESCRIPTION OF THE VARIABLES USED FOR STEP DETECTION

Variable	Description
a_x, a_y, a_z	x, y and z component of the raw acceleration in the body frame
$\tilde{a}_x$, $\tilde{a}_y$, $\tilde{a}_z$, $\lvert\tilde{\boldsymbol{a}}\rvert$	x, y, z and magnitude component of the low-pass filtered acceleration in the body frame
$\hat{a}_x$, $\hat{a}_y$, $\hat{a}_z$, $\lvert\hat{\boldsymbol{a}}\rvert$	x, y, z and magnitude component of the scaled acceleration in the body frame
$\boldsymbol{a}_j^{n_s}$	Fragment of the scaled acceleration with length n_s
s_{ideal}, e_{ideal}	Ideal output for detecting start/end of a step (1 for only 1 sample per step)
s_{block}, e_{block}	Output for detecting start/end of a step where the output is 1 for 21 samples per step
s_{delay}	s_{block} delayed with a delay d
$s_{desired}$, $e_{desired}$	Output for detecting start/end of a step that the LSTM network must learn
h_ℓ, h'_ℓ	Hidden state of the first and second LSTM layer at the ℓ^{th} timestep
s_{LSTM}, e_{LSTM}	Predicted output of the LSTM network for detecting start/end of a step
$\mathcal{L}_{ce}$, $\mathcal{L}_{s,ce}$, $\mathcal{L}_{e,ce}$	Binary cross-entropy loss for the output corresponding to the LSTM network, the start of a step and the end of a step
n_f	Number of fragments in the training set for the step detection network
$N_{0,s}$, $N_{1,s}$, $N_{0,e}$, $N_{1,e}$	Number of samples in the acceleration fragments of the training set for which the desired output is 0 and 1 for respectively the start and end of a step
w_s, w_e	Weights that determine how much an input of the network contributes to the loss for the start/end of a step
$b_{desired}$, b_{LSTM}	Concatenation of respectively $s_{desired}$ and $e_{desired}$, and s_{LSTM} and e_{LSTM}
r, p, f-score	Recall, precision and f-score of the LSTM network respectively
$\hat{s}_{LSTM}$, $\hat{e}_{LSTM}$	Rounded version of s_{LSTM} and e_{LSTM}; 1 if the original value was larger than th and 0 otherwise
$\tilde{s}_{LSTM}$, $\tilde{e}_{LSTM}$	Output of the LSTM network after preprocessing
$\Omega_{s,d}$, Ω_e, Ω_e	Sets that respectively contain the start (with delay), the start (without delay) and the end of the steps
Ω_1, Ω_2, Ω_3	Sets that respectively contain the start of steps that do not coincide with an end, the start/end of steps that coincide and the end of steps that do not coincide with a start
s_1, s_2	The start of a step before and after start s
e_1, e_2	The end of a step before and after start s
$\Omega_{f,s}$, $\Omega_{f,e}$	Sets that contain the final starts and ends of the detected steps
$\Omega_{true,s}$, $\Omega_{true,e}$	Sets that contain the true starts and ends of the steps

III. STEP LENGTH

In Section II, we designed an algorithm that detects the boundaries, i.e., the start and the end instant, of each step that a user takes and, hence, allows to extract all the acceleration samples that correspond to a certain step. In this section, we now want to estimate the length of a step using the extracted samples. In this work, we combine the step detection algorithm of Section II with a ridge-regression-based step length estimation algorithm, which resulted in the best performance among the considered algorithms from [31]. Although this ridge regression step length estimator performs well, the drawback is that it is obtained by training a machine learning network on a subset of a larger ad hoc selected feature

set that was not optimized for step length estimation. While the performance of the estimator is improved by optimally selecting the features, this comes at the cost of higher complexity. Therefore, we opt in this article to propose a novel step length estimator based on a deep learning network that consists of several LSTM, convolutional, and dense (or fully connected) layers. In this network, the LSTM layers are used to detect temporal patterns in the input data, whereas the convolutional layers serve to detect spatial patterns. Such a convolutional layer is commonly used in image processing to automatically extract useful features from the input data, i.e., feature extraction and selection are integrated into the proposed algorithm. This is in contrast to the algorithm from [31], where the features from the large initial set needed to be extracted manually, and a feature selection procedure is used to select the optimal subset of features. In the next section, the architecture of this network is discussed.

A. Deep Learning Network for Step Length Estimation

Similarly as for the LSTM network that determines how likely an accelerometer sample is at the start/end of a step, several actions are required to obtain a trained deep learning network for estimating the step length. In the remainder of this section, we discuss these actions in more detail.

1) Preprocessing the Data: First, we need to preprocess the accelerometer data before it enters the deep learning network, i.e., similarly as for the step detection algorithm, we feed the scaled acceleration samples $\hat{\mathbf{a}}_i = [\hat{a}_{x,i}, \hat{a}_{y,i}, \hat{a}_{z,i}, |\hat{\boldsymbol{a}}|_i]^T$ to the network. Furthermore, we use the output of the step detector, consisting of each step j in the measurement of the start s_j and end e_j of the step, to divide the scaled accelerometer data into fragments $\boldsymbol{a}_j^{\text{step}} = [\hat{\boldsymbol{a}}_{s_j}, \ldots, \hat{\boldsymbol{a}}_{e_j}]$ containing all acceleration samples of the jth step. In general, this fragment contains a variable number of samples. However, the deep learning network that we use requires an input fragment of a fixed length. Therefore, we transform the input fragments to a sequence of fragments of fixed size $4 \times n_{s,\text{fixed}}$, by either zero-padding or truncating the acceleration fragments. In this work, we set $n_{s,\text{fixed}} = 100$ samples, i.e., at a sample rate of 100 Hz, this would correspond to an accelerometer fragment of 1 s.

2) Desired Output: The desired output of the deep learning network ideally consists of the step length $L_{\text{desired},j}$ for each accelerometer fragment $\boldsymbol{a}_j^{\text{step}}$ that corresponds with a step. However, to predict the (unscaled) length of a step, the final layer of the deep learning network needs to have larger weights to transform the output of the intermediate layers—often these outputs are in the range $[-1, 1]$ or $[0, 1]$—to the step length. To learn these larger weights, the network generally needs to be trained longer, and the gradient-descent-based algorithm that is used to determine the weights of the deep learning network may become unstable. Therefore, in this work, we will train the network to predict the *scaled* step length $\hat{L}_j$. To this end, we apply a MinMaxScaler to the desired step length $L_{\text{desired},j}$, which results in the scaled step lengths $\hat{L}_{\text{desired},j}$. From the scaled step length predicted by the network, the inverse of the scaling operation can then be used to convert the predicted scaled step length to the unscaled one.

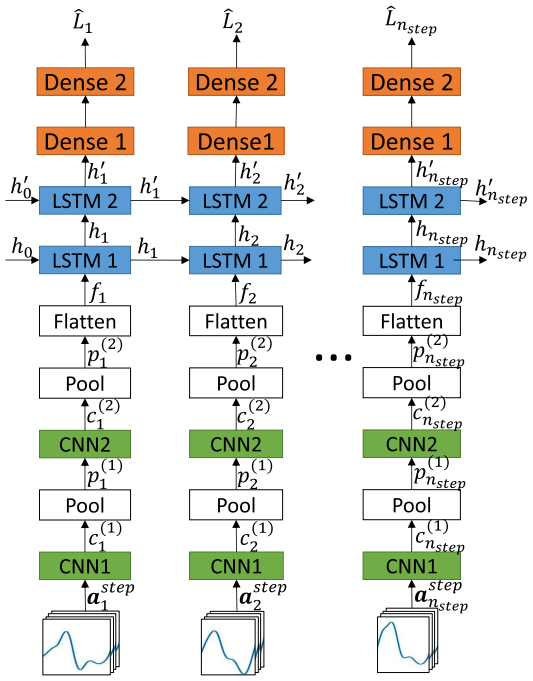


Fig. 7. Unfolded architecture of deep learning network for step length estimation.

3) Deep Learning Algorithm: In Fig. 7, we show the architecture of the deep learning network that we use to determine the scaled length of a step, where we unfold the network for n_{step} subsequent steps. To predict the step length for an acceleration fragment $\boldsymbol{a}_j^{\text{step}}$, in the first phase, the network needs to extract useful features from this fragment. This is done by the CNN, where every additional convolutional layer is capable of automatically extracting features with increasing complexity. For example, in a 2-D convolutional network for image classification, the first layer(s) can detect edges, while the following layers can use the edges to, e.g., detect shapes. In this work, we restricted the number of convolutional layers to two, as our simulations showed that more layers did not result in a better performance. Each convolutional layer is followed by a pooling layer to reduce the output size of the convolutional layer and, consequently, the number of learnable parameters so that the extracted features depend less on the place they are detected in the input, implying that the network becomes more robust and less susceptible to overfitting.

The transformation of the input fragments into fragments of fixed size in the preprocessing step (see Section III-A.1) is necessary due to the convolutional layers, which require inputs of fixed size. Hence, for the ℓth timestep of the network, $\ell = 1, \ldots, n_{\text{step}}$, the input consists of the ℓth acceleration fragment $\boldsymbol{a}_\ell^{\text{step}}$ of size $4 \times n_{s,\text{fixed}}$. The first convolutional layer consisting of $n_{\text{filt},1}$ filters of length $n_{l,1}$ transforms the ℓth acceleration fragment into a new output vector $c_\ell^{(1)}$ of size $n_{\text{filt},1} \times n_{s,\text{fixed}}$ by taking the convolution of the acceleration fragment and the $n_{\text{filt},1}$ filters. Next, $c_\ell^{(1)}$ is applied to a max pooling layer, which reduces its input vector to a vector $p_\ell^{(1)}$ of size $n_{\text{filt},1} \times \lceil (n_{s,\text{fixed}}/n_{\text{pool},1}) \rceil$ by only keeping from its input the maximum in nonoverlapping windows of $n_{\text{pool},1}$

(subsequent) samples. The output of the first pooling layer is then applied to a second convolutional and pooling layer, which ultimately results in an output $p_\ell^{(2)}$ of size $n_{\text{filt},2} \times \lceil (n_{s,\text{fixed}}/n_{\text{pool},1}n_{\text{pool},2}) \rceil$.

After the convolutional and pooling layers, the extracted feature map is sent to the LSTM layers. These LSTM layers enable the network to also take into account previous acceleration fragments to better estimate the step length for the current acceleration fragment. As the LSTM layers require a 1-D input, we first need to pass the 2-D output $p_\ell^{(2)}$ to a flatten layer, which transforms the 2-D input into a 1-D output f_ℓ. This flattened output is then applied to two LSTM layers, which, respectively, transform their input to an output of size $n_{\text{LSTM},1} \times 1$ and $n_{\text{LSTM},2} \times 1$. In this work, we chose to use two LSTM layers as more layers did not result in a significant performance improvement. Finally, we still need to extract the predicted scaled step length from the output of the last LSTM layer. To this end, generally, one or more dense (or fully connected) layers are used. In this work, we achieved the best performance with two dense layers, which transform their input to an output of, respectively, size $n_{\text{dense},1} \times 1$ and $n_{\text{dense},2} \times 1$. As the final dense layer needs to return a scalar, i.e., the scaled step length estimate $\hat{L}_\ell$ of the current step, the length $n_{\text{dense},2}$ of the dense layer output must be equal to one. Similarly as for the network used for the step detection, we add a dropout layer with dropout probability $p_{\text{dropout},i}$, where the subindex i corresponds to the ith dropout layer, after each pooling and LSTM layer (not shown in Fig. 7), in order to minimize the risk of overfitting.

To train the network, we divide the training data into batches, where one batch contains b_s examples. Each of these examples contains the accelerometer fragments $\boldsymbol{a}_{b,i,\ell}^{\text{step}}$ of n_{step} successive steps together with the corresponding desired scaled step lengths $\hat{L}_{\text{desired},b,i,\ell}$, where b and i denote that the example is the ith example in the bth batch, and $\ell = 1, \ldots, n_{\text{step}}$ denotes the ℓth step in the example. For each example in the bth batch, we then apply the data of the n_{step} steps to the deep learning network of Fig. 7 to predict the corresponding n_{step} scaled step lengths $\hat{L}_{b,i,\ell}$. These outputs are then compared with the desired output using a mean squared error loss function $\mathcal{L}_{\text{mse}}^b$ [see (19)], which allows updating the weights of the deep learning network with a gradient descent approach every time a batch has been processed

$$\mathcal{L}_{\text{mse}}^b = \frac{1}{b_s \cdot n_{\text{step}}} \sum_{i=1}^{b_s} \sum_{\ell=1}^{n_{\text{step}}} \left(\hat{L}_{\text{desired},b,i,\ell} - \hat{L}_{b,i,\ell} \right)^2. \quad (19)$$

This training process is applied to all batches of the training data for 1000 epochs. More specifically, during an epoch, we start from the model that was trained in the previous epoch (and not the model of the best epoch so far) and apply the training data batch by batch so that the weights of the deep learning network can be updated once again. At the end of each epoch, the performance (see Section III-A.4) of the model was evaluated on the test set, and if the performance of the model on the test set for the current epoch was better than the models for all previous epochs, then the model is saved. By using the test set to determine the best epoch, we avoid that the deep learning network would overfit on the training data. To implement the step length estimation network, we use, similarly as for the step boundary detection, Keras and the Adam optimizer with a learning rate of 0.001 to update the internal weights of the network.

TABLE II
DESCRIPTION OF THE VARIABLES USED FOR STEP LENGTH ESTIMATION

Variable	Description
$\boldsymbol{a}_j^{step}$	Fragment of the scaled acceleration containing all the samples of a step
$L_{desired,j}$,$\hat{L}_{desired,j}$	True length and the scaled true length of a step
L_j ,$\hat{L}_j$	Length and the scaled length of a step predicted by the deep learning network
$c_\ell^{(1)}$, $c_\ell^{(2)}$	Output of respectively the first and second convolutional layer at timestep ℓ
$p_\ell^{(1)}$, $p_\ell^{(2)}$	Output of respectively the first and second pooling layer at timestep ℓ
h_ℓ, h'_ℓ	Output of respectively the first and second LSTM layer at timestep ℓ
f_ℓ	Output of the flatten layer at timestep ℓ
$\mathcal{L}_{mse}^b$	Mean squared error loss of the step length network on batch b

4) Performance Measure: To evaluate the performance of the step length estimator, we determine the mean absolute value of the error between the estimated step length and the true step length extracted from the ground-truth data. So far, we assumed that we knew the step boundaries s_j and e_j of the steps to extract the acceleration fragment. In practice, however, these step boundaries are not known in advance. Hence, in the results section, we also investigate the performance when we combine the step length estimator with the step detector from Section II to extract the accelerometer samples that correspond to a step instead of assuming that the boundaries are known.

In Table II, we, for clarity, summarize the variables used for step length estimation together with their description.

IV. RESULTS

In this section, we discuss the results of the deep-learning-based step detection and step length estimation algorithms. First, we introduce the different datasets that are used to train and evaluate the different neural networks. Next, we then evaluate the performance of the step detector and compare it with a state-of-the-art algorithm [21]. This involves the selection of the parameters specified in Section II-A in order to obtain the best recall, precision, and f-score, as well as training the LSTM network. After training the network, we use the algorithm from Section II-B to extract the time instant of the start and end of each step from the output of the LSTM network and evaluate the accuracy of the step detector.

Second, we assess the performance, i.e., the mean absolute error (mae), of the step length estimator from Section III, assuming the start and end of the step are known. Also, this network involves the selection of some parameters, so we first determine parameter values that lead to a good, though not necessarily optimal, performance as testing all possible parameter value combinations is infeasible. After training the

network, we determine the performance of the estimator and compare the results with the step length estimator from [31], where the feature selection was not included in the network and the state-of-the-art algorithm of [28].

Finally, we combine the deep-learning-based step detector and the step length estimation algorithm. To this end, we first use the step detector to detect the start and end of a step and use these to extract the accelerometer fragment that corresponds to that step. Next, we use this fragment as an input for the step length estimator and compare the performance with the performance of the step length estimator, where we assumed that the start and end of a step are known.

A. Datasets

In this network, we use two different datasets, i.e., one for training and evaluating the step detection algorithm and one for the step length estimation. For the LSTM network/step detection algorithm, we use the dataset that was introduced in [19]. For this dataset, we gathered the IMU data of three users that, while holding a smartphone in a texting position, perform a measurement where they are free to choose the trajectory inside a room. For these measurements, two different smartphones are used, i.e., a Samsung Galaxy IV and a Motorola Moto Z Play. The Samsung device contains a K330 IMU from STMicroelectronics that samples the acceleration at 100 Hz and has a range of ± 2 g. The Motorola device, on the other hand, contains a BMI160 IMU from Bosch that samples the acceleration at 100 Hz and has a range of $\pm 16g$. To obtain the ground-truth data, we manually extracted the step boundaries from the low-pass filtered acceleration magnitude by determining the instants at which the magnitude crosses 1 g, i.e., the gravitational force, with a positive slope. During the measurements, also, the total number of steps was counted as verification that the right amount of steps was annotated in the dataset. This resulted in a dataset with 6226 steps, of which 3120 steps are used to train the step detection algorithm, and 3106 steps to test it.

For the step length estimation, we use the dataset from our previous work [31] to train and test the network. During a measurement, the user walked on a straight path of approximately 5 m next to a tape measure with a smartphone in a texting position, while a ceiling-mounted camera captured the experiment. Based on these camera images, it was then possible to extract the ground-truth step length from the position of the foot relative to the tape measure. For this dataset, we gathered the IMU data with the same devices that were used for the step detection dataset and with the same three test persons. This resulted in a dataset with in total of 837 annotated steps, where each measurement contributed three to four subsequent steps. Approximately 80% of this dataset is used to train the network, while the remaining 20% is used to test the trained algorithm. As, in the dataset, each experiment resulted in minimally three successive steps, we set $n_{\text{step}} = 3$.

B. Results Step Detection

For the LSTM network for the step detection, as discussed in Section II-A, we first need to optimize the parameters $(n_s, n_h, d, p_{\text{dropout}})$. As these parameters are not related to each other and training the network is an expensive operation, testing every possible combination is very time-consuming. Therefore, in this work, we determine a (suboptimal) parameter set that leads to a good performance. To determine the values for the parameters, we sequentially update the parameter set, where each new parameter set differs in only one parameter compared to the parameter set that led so far to the best performance. In this tuning process, we started with tuning n_s followed by d, n_h and p_{dropout}. With this order, we first optimize the parameters that determine the input of the network (n_s, d), then the parameters of the network itself (n_h), and, finally, the parameters that regularize the network, i.e., prevent overfitting (p_{dropout}). Note that this order is arbitrary, and another order could lead to better performance. For each of the approximately 100 considered parameter sets, we train the LSTM network for 200 epochs on the training data and evaluate the performance of the trained LSTM network by computing the f-score, as discussed in Section II-A. This f-score is determined for each measurement in the training set, and the final f-score for an epoch is found by averaging over all the measurements in the training set. For each parameter set, we save the f-score and the LSTM network for the epoch that resulted in the best f-score. The parameter set resulting in the highest f-score is used for the final results. To select the values for the parameters, we need to take into account the following considerations.

1) *Number n_s of Samples in a Fragment:* As it is computationally hard to analyze long acceleration fragments to find the steps included in the signal, we divide the acceleration signal in shorter fragments, i.e., n_s may not be too large. However, to be able to detect the start and end of a step, all samples of the step need to be included in the fragment, implying that n_s may not be too small and at least large enough to contain all samples of one step. Hence, n_s depends on the sample frequency of the IMU and the walking frequency. As, in this work, the IMU samples at 100 Hz and a walking user usually takes at least one step per second, n_s should be at least 100 samples.
2) *Number n_h of States in an LSTM Layer:* A higher n_h enables the LSTM network to learn a more complicated model. However, choosing a very large n_h can result in overfitting.
3) *Delay d (in Number of Samples) in the Decision Process for the Start of a Step:* A higher value gives the LSTM network more time/information to decide how likely a sample of the acceleration data was the start of step. However, to extract a step in real time, the delay must be shorter than the duration of a step. Hence, d also depends on the sample and walking frequency. As the sample frequency in this work is 100 Hz and a user usually takes less than 2.5 steps per second, d should be less than 40 samples.
4) *Probability $p_{dropout}$ That an Output of a Layer Is Not Used in the Next Layer:* As we have two dropout layers in the LSTM network for step detection, we consider that each of the dropout layers can have a different

TABLE III
OPTIMAL PARAMETERS FOR THE LSTM NETWORK

n_s(samples)	n_h	d(samples)	$p_{dropout}$
200	400	30	[0, 0.2]

TABLE IV
PRECISION, RECALL, AND F-SCORE OF THE LSTM NETWORK FOR STEP DETECTION ON TRAINING AND TEST SETS

	precision(%)	recall(%)	f-score(%)
train	98.8	98.4	98.6
test	96.0	94.8	95.4

dropout probability. In general, a low dropout probability increases the risk of overfitting, while a high probability can prevent the LSTM network from learning anything.

In Table III, the parameters for the LSTM network are given, which resulted in the best performance. As n_s is equal to 200 samples, this means that, for walking frequencies between 1 and 2.5 Hz, each fragment contains at least two steps and can contain up to five steps. For the number of hidden states n_h in the LSTM layers, we find that a value of 400 results in the best performance. During our experiments, however, we noticed that a lower value does not severely degrade the performance.

In Table IV, we show the precision, recall, and f-score of the LSTM network on the training and test sets when the parameters from Table III are used to train the network. For the training set, we see that precision, recall, and, hence, f-score are very close to each other. The precision of 98.8% is slightly higher than the recall of 98.4%, which results in an f-score of 98.6%. This means that the LSTM network more frequently fails to predict the start or end of a step, i.e., a false negative, than it predicts a nonexisting step, i.e., a false positive. On the test set, the difference between precision and recall is larger than on the training set. When we compare the performance for the training set to the performance for the test set, we can see that the difference is around 3%, e.g., the f-score decreases from 98.6% to 95.4%.

In the second part of the step detection algorithm, we use the output of the LSTM network from the first part to extract the time instants where a step started (ended) and combine them to obtain more accurate estimates, as described in Section II-B. To obtain good performance, we first need to optimize the parameters $(\text{th}, m, M, w, w_{1\to0\to}, \text{th}_{\max})$ of the algorithm. In contrast to the training of the LSTM network, the algorithm that extracts the start and end of a step has much lower complexity. As a consequence, it is easier to find the optimal parameter settings. To determine the performance of a parameter set, we extract the time instants when a step started (ended) and determine the precision, recall, and f-score, where the start (end) is considered to be detected correctly when the time between the predicted and true start (end) of a step is smaller than half the minimum duration of a step, i.e., $(m/2)$. To find the optimal value for the parameters, we use a grid search over all parameters, where we take into account the following considerations.

1) *Threshold th Used in* (10) *and* (11) *to Determine the Rounded Outputs* $\hat{s}_{LSTM}$ *and* $\hat{e}_{LSTM}$*:* As the final dense layer uses a sigmoid activation, i.e., the outputs of the network lay in the interval $[0, 1]$, an intuitive choice would be to set the threshold as the center of this interval, i.e., $\text{th} = 0.5$. When the threshold reduces, the LSTM network will decide that a start/end of a step occurred for smaller values of s_{LSTM} and e_{LSTM}, implying that the probability of a false negative would reduce, but, at the same time, the probability of false positive increases. When the threshold increases, the opposite occurs. Taking into account that the recall of the LSTM network is lower than the precision, i.e., the number of false negatives is slightly larger than the number of false positives (see Table IV), we expect that a value for th lower than 0.5 will result in the best performance. In our grid search, we considered values for $\text{th} \in \{0.3 : 0.05 : 0.6\}$.
2) *Thresholds m and M for, Respectively, the Minimum and Maximum Durations of a Valid Step (in Number of Samples):* Humans walking in general take between approximately 60 and 150 steps per minute, i.e., a step frequency between 1 and 2.5 Hz. Hence, a sampling frequency of 100 Hz results in steps between 40 and 100 samples in length. This is also confirmed by the steps in our dataset that has a duration between 34 and 131 samples and for which 99.8% of the steps have a duration between 37 and 102 samples. As m and M determine whether a fragment can correspond to a step, they will have an impact on the accuracy of the network, i.e., reducing m/increasing M increases the probability of false positives, while increasing m/reducing M will result in a higher probability of false negatives. In our grid search, we considered values for $m \in \{32 : 2 : 46\}$ and $M \in \{142 : 2 : 152\}$.
3) *Minimum length* w *(in Number of Samples) of a Sequence of Ones in* (12)*:* A large value for w can lead to not detecting the start (end) of some steps, while a small value can lead to false detections. As the LSTM network is trained to generate an output containing 21 successive samples "1" when a step started or ended, the minimum length w may not exceed 21. Note that the LSTM network generates the output "1" if its output is above the threshold th. Due to noise and fluctuations, one or more of the outputs of the network may not exceed the threshold, even if a start or end of a step occurred. To avoid that this results in a false negative, we set the minimum length w as roughly half of the expected length of the sequence of "1," i.e., in our grid search, we considered values for $w \in \{6 : 2 : 14\}$.
4) *Threshold* $w_{1\to0\to1}$ *(in Number of Samples) Below Which a* $1 \to 0$ *and* $0 \to 1$ *Transition Is Replaced by a Sequence of Ones* $1 \to 1$*:* The occurrence of a $1 \to 0$ and $0 \to 1$ transition is closely related to the disturbances mentioned for the minimum length w. Therefore, the threshold $w_{1\to0\to1}$ and the minimum length w are correlated. Taking into account the minimum duration m of a valid step and the minimum length

TABLE V
OPTIMAL PARAMETERS TO EXTRACT TIME INSTANTS OF START AND END

th	th_{max}	$w_{1\to0\to1}$ (samples)	w (samples)	m (samples)	M (samples)
0.4	0.75	14	12	36	150

w of a sequence of "1," the maximum duration $w_{1\to0\to1}$ is upper bounded by $w_{1\to0\to1} < m - w$. Violating this restriction can result in replacing two valid step detections with one false detection. Too large $w_{1\to0\to1}$ will, therefore, result in a higher number of false negatives, but, on the other hand, too small $w_{1\to0\to1}$ results in erroneous detection of start/end instants of steps, i.e., false positives. In our grid search, we considered values for $w_{1\to0\to1} \in \{10 : 2 : 18\}$.

5) *Threshold th_{max} That Ensures That a Start (End) of a Step Is Only Detected When the Unrounded Output of the LSTM Network Is Large Enough:* Choosing th_{max} too large results in false negatives (a lower recall), while a too small value results in false positives (a lower precision). As mentioned in Section II-B, this variable must take values in the interval [th, 1]. In our grid search, we considered values for $\text{th}_{\text{max}} \in \{0.7 : 0.05 : 0.95\}$.

Table V shows the optimal parameters obtained with the grid search. However, several other parameter sets resulted in the same or slightly worse performance, indicating that the algorithm is robust to slight variations of these parameters. From the table, we can see that th $= 0.4$, which is indeed as we expected slightly lower than 0.5. Although this lower value for th increases the risk of false detections, thanks to the values for $\text{th}_{\text{max}} = 0.75$ and $w = 12$, we only detect the start (end) of a step when the sequence of ones is long enough, and the corresponding unrounded output of the sequence is large enough, which eliminates most false detections.

In Tables VI and VII, we show the precision, recall, and f-score of the step detector on the training and test sets, respectively, before and after combining the start and end of a step, for the parameter settings from Table V. In Table VI, where we show the performance of the algorithm before combining the start and end of the steps, we observe that, in most cases, the precision, recall, and, hence, the f-score, are very similar. Only for detecting the end of a step in the test set, the difference between the recall and precision is slightly larger. From this table, we also see that the precision is higher than the recall, implying that the step detector more likely does not detect a step than it detects a wrong step. Comparing the performance on the training and test sets, it is clear that, as expected, the training set results in the best performance, i.e., on average, the precision, recall, and f-score for the training set are approximately 1% better than for the test set.

In Table VII, we show the performance of the algorithm after combining the start and end of steps. In comparison with Table VI, we see that, for a given set (start/end and training/test), the difference between precision and recall is smaller. Furthermore, for the training set, the performance of the step detector hardly improves by using the relationship between the starts and ends of steps, while, for the test set, there is a clear improvement. For the start of a step, precision and recall, respectively, increase from 99.1% and 98.9% to 99.2% and 99.3%, and for the end of the step, the precision and recall increase from, respectively, 98.7% and 97.9% to 99.0% and 99.1%.

TABLE VI
PRECISION, RECALL, AND F-SCORE ON TRAINING/TEST FOR DETECTING START AND END OF STEP BEFORE COMBINING START AND END

	precision(%)	recall(%)	f-score(%)
start/end			
train	99.8 / 99.7	99.7 / 99.5	99.8 / 99.6
test	99.1 / 98.7	98.9 / 97.9	99.0 / 98.3

TABLE VII
PRECISION, RECALL, AND F-SCORE ON TRAINING/TEST FOR DETECTING START AND END OF STEP AFTER COMBINING START AND END

	precision(%)	recall(%)	f-score(%)
start/end			
train	99.8 / 99.8	99.7 / 99.7	99.8 / 99.8
test	99.2 / 99.0	99.3 / 99.1	99.3 / 99.1

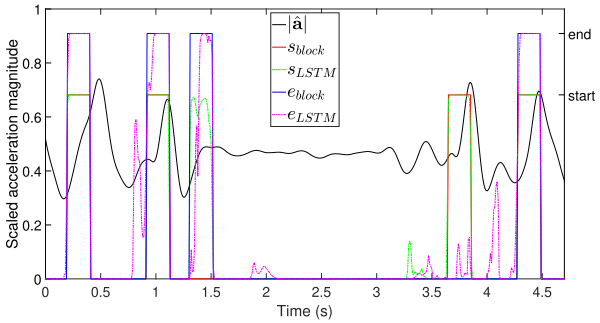


Fig. 8. Example of a fragment of the acceleration magnitude where the LSTM network erroneously detects the start of a step (around 1.4 s). This wrong start is discarded by our algorithm.

In Figs. 8–10, we show several examples of fragments of the output of the LSTM network. In Fig. 8, the network incorrectly detects a start of a step around 1.4 s. However, as there is no end of a step within M samples, the step detection algorithm ignores this start of the step. In Fig. 9, the LSTM network fails to detect the start and end of the first step. In this case, the step detection algorithm is not capable of correcting these errors as the networks are nothing around 0.7 s. If the LSTM network would have detected the first start of a step, the step detection algorithm would correct the missing end of the first step. In Fig. 10, the LSTM network does not detect the start and end of the step from 1.1 to 1.9 s. However, as the starts (ends) of the previous and next steps are detected correctly, the step detection algorithm adds the missing start (end).

Finally, we also compare the step detection algorithm with [21]. In that work, the authors considered an LSTM, CNN, and WaveNet network to detect steps. To this end, the networks were given accelerometer fragments of a fixed

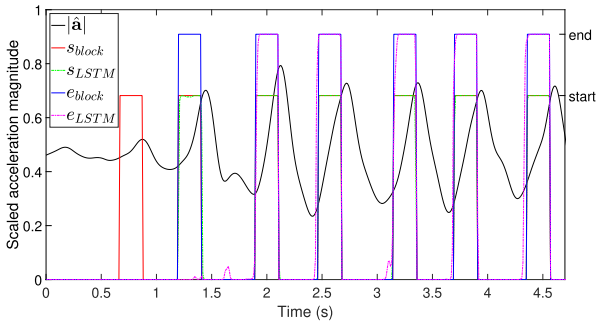


Fig. 9. Example of a fragment of the acceleration magnitude where the LSTM network fails to detect the start and end of the first step (not corrected by our algorithm).

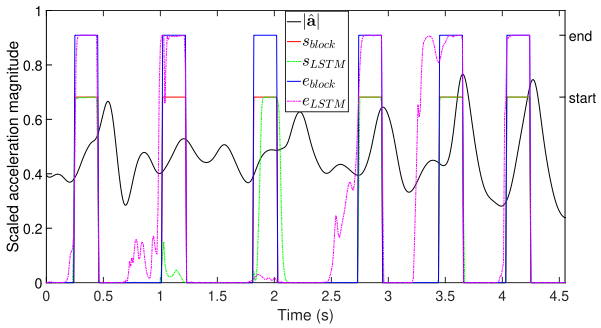


Fig. 10. Example of a fragment of the acceleration magnitude where the LSTM network fails to detect a start (around 1.1 s) and end (around 1.9 s) of a step. With our algorithm, this is corrected.

length for which the networks needed to predict if the last sample of the fragment was part of a left or right step. With a sliding window approach, we can then determine, for each sample of the accelerometer, if it was from a left or right step, and a transition between can be seen as the start/end of a step. As the authors found that a CNN resulted in the best performance, we only compare our step detection algorithm with the CNN-based step detector from [21]. However, as our step detection dataset also contained moments where the user was not walking, we need to slightly modify the algorithm so that it can detect a left step, a right step, and no step. Furthermore, we also needed to apply weight balancing to train the network as the number of examples for a left/right step was much higher than the number of examples with no step. In Table VIII, we show the resulting precision, recall, and f-score of this method for detecting the start and end of the step. Comparing this with Tables VI and VII, it is apparent that the presented step detection algorithm outperforms the step detector from [21], especially in terms of precision. The main reason for this worse performance is that the output of the CNN network often contained short bursts where it predicted a left (right) step instead of a right (left) step. This leads to many false step detections and, hence, a lower precision. Therefore, postprocessing the output of the CNN network, similarly as in our algorithm, can potentially result in a much better performance.

C. Results Step Length

In this part, we evaluate the performance of the deep-learning-based step length estimator and compare it with the

TABLE VIII
PRECISION, RECALL, AND F-SCORE ON TRAINING/TEST FOR DETECTING START AND END OF STEP WITH CNN NETWORK [21]

	precision(%)	recall(%)	f-score(%)
start/end			
train	90.5 / 90.5	99.0 / 99.0	94.6 / 94.6
test	81.4 / 81.3	97.7 / 97.3	88.7 / 88.6

TABLE IX
TUNED PARAMETERS FOR DEEP LEARNING NETWORK FOR STEP LENGTH ESTIMATION

b_s	n_{filt}	n_l	n_{pool}	n_{dense}	n_{LSTM}	$p_{dropout}$
75	[128, 128]	[42, 7]	[6, 6]	[512, 1]	[100, 100]	[0.2, 0.0, 0.0, 0.3]

step length estimator in our previous work [31] and with [28], where the authors use a CNN to estimate the step length from accelerometer and gyroscope data. Similar to the proposed step detector, the proposed step length estimation network requires the tuning of several parameters, related to the convolutional, LSTM, dense, pooling, and dropout layers. As the number of parameters is high and training a deep learning network requires a considerable amount of time, an exhaustive search for the optimal parameters is not feasible.

Therefore, we determine a parameter set that results in satisfactory performance, though not necessarily optimal. To find suitable values for these parameters, we trained and evaluated the step length estimator with a limited number of parameter sets, where we sequentially update the parameter set by changing one parameter compared to the parameter set that, so far, led to the best performance. For the step length estimation network, we first tuned the parameters that determine how the network is trained (b_s), followed by the parameters of the network itself ($n_{\text{pool}}, n_l, n_{\text{filt}}, n_{\text{LSTM}}, n_{\text{dense}}$), and, finally, the parameters that regularize the network (p_{dropout}). The final parameter set that results in the best performance, where the performance is determined by the mae on the step length error, is given in Table IX. In this table, parameters for similar layers are grouped in an array, where the ith element of the array corresponds to the ith layer of that type in the network architecture. For example, $n_l = [42, 7]$ indicates that the first convolutional layer uses filters with length $n_{l,1} = 42$, while the second convolutional layer has filters with length $n_{l,2} = 7$.

In Table X, we give the mae on the step length that is achieved on both the training and test sets using the parameters from Table IX. In addition, we also compare the performance of the proposed deep learning network with the performance that we achieve with the feature selection method from our previous work [31] and step length estimator of [28], which only uses convolutional layers and no LSTM layers. Hence, the latter work in comparison with the proposed approach is not capable of using information about previous steps. The proposed deep learning approach results in an mae of 1.63 and 3.21 cm on, respectively, the training and test sets, while the feature selection approach resulted in an mae of, respectively, 4.07 and 5.15 cm on the training and test sets. Hence, it is clear that the deep learning approach outperforms the feature selection approach in terms of mae. The feature

TABLE X
MAE FOR STEP LENGTH ESTIMATION USING FEATURE SELECTION [31], CNN [28], AND PROPOSED DEEP LEARNING APPROACH ON TRAINING/TEST SET

mae(cm)	feature selection	deep learning	[28]
train	4.07	1.63	1.76
test	5.15	3.21	3.30

selection approach, on the other hand, has a lower complexity, which results in faster training and prediction time. From our tests, however, we can conclude that even the deep learning approach is capable of providing step length estimates in near real time. Comparing the proposed approach with [28], we notice that the latter performs slightly worse compared to the proposed algorithm even though the latter also uses data from the gyroscope.

D. Results Step Detection and Length Combined

In the previous section, we discussed the results of the proposed deep-learning-based step length estimator. For these results, however, we silently assumed that the boundaries, i.e., the start and the end, of a step are known, which, in reality, is not the case. Hence, in this section, we evaluate the performance of the step length estimator from Section III combined with the step detector from Section II to extract the start and end of each step.

The LSTM step detection algorithm from Section II was trained and evaluated in Section IV-B using the *step detection* dataset from [19], where it was used for step counting instead of step detection. This dataset, however, does not contain ground truth for the step length. On the other hand, the dataset used to train and evaluate the performance of the step length estimator (see Section IV-C), i.e., the *step length* dataset, consists of many short fragments. As most errors in the step detection occur when the user starts or stops walking, this *step length* dataset is less suitable to optimize the parameters of the step detection algorithm than the *step detection* dataset, which contains longer fragments with more steps. Therefore, in this section, although we will apply the combined step detection and step length estimation algorithm on the step length dataset, we use, for the parameters of the step detection algorithm, the values given in Table V, i.e., optimized on the step detection dataset. The detected start and end instants of the steps are used to divide the measured acceleration fragments of the *step length* dataset into shorter fragments containing a single step. The resulting short fragments are then fed (after padding the fragments to length $n_{s,fixed}$) to the step length estimation algorithm from Section III, where we use the parameters from Table IX (which are obtained by assuming that the boundaries of the steps are known) to estimate the step length.

In Table XI, we show the performance of the proposed step detector on the training and test sets of the *step length* dataset. Comparing Tables VII and XI, we observe that, as expected, the performance of the step detector on the step length dataset is slightly worse (for both the training and test sets) compared to the performance on the step detection dataset due to the short fragments in the step length dataset. The only exception is the recall on the test set of the step length data, which is equal to 100%, i.e., all steps in that dataset are detected.

In Table XII, we compare the performance of the step length estimator on the training and test sets using a feature selection, a CNN-based [28], and a deep learning approach. To this end, we determine the mae on the step length in two ways. The first way calculates the mae only on the steps that were detected correctly, i.e., false positive and false negative step detections are ignored. Using the feature selection approach, this results in an mae of 4.20 and 5.14 cm on, respectively, the training and test sets. The proposed deep learning approach outperforms the feature selection approach and results in an mae of, respectively, 2.46 and 3.54 cm on the training and test sets. With the CNN-based approach of [28], we again see that it performs slightly worse even though it also uses gyroscope data. Comparing Tables X and XII, we see that the performance using the feature selection approach barely degrades when estimated step boundaries are used. For the deep learning and CNN-based approach, on the other hand, we notice a more significant degradation of performance, especially on the training set. With the deep learning approach, the mae on the training set increases from, respectively, 1.63 to 2.46 cm when, instead of using the true step boundaries, the proposed step detector is used. A possible explanation for this observation is that the feature selection approach builds a less complex model for the step length than the deep learning approach. While the less complex model of the feature selection approach, in general, results in worse performance, it also makes the method less susceptible to changes in the input. On the other hand, the degradation of the deep learning model is due to the dataset that is used to train the model. During the training phase, it was assumed that the step boundaries are known. However, when we combine the step length estimator with the step detection network, the fragments fed to the step length estimator will (slightly) differ from the fragments with known step boundaries. As the step length network is not trained on data with uncertainty on the step boundaries, it is overfitted to the case with known step boundaries. Although we expect that we can reduce the degradation by training the step length estimator network on fragments with uncertain step boundaries, this would come at the cost of higher training complexity. As the test data are not used to train the step length estimator network, the impact of the overfitting is less prominent than on the training set, implying that the degradation for the test set is lower than for the training set.

For the second way to determine the mae, we also take into account false negative and false positive step detections, i.e., steps that were not detected by the step detector and steps that were detected when no step should have been detected. To take these events into account, we set the true step length for a falsely detected step equal to 0 cm, and for a step that was not detected, we set the step length estimate equal to 0 cm. In Table XII, the resulting mae is given. For the feature selection approach, this results in an mae of 6.49 and 7.71 cm on, respectively, the training and test sets. On the other hand, when using the deep learning approach, an mae of 4.91 and 6.56 cm is achieved on, respectively, the training

TABLE XI
RECALL, PRECISION, AND F-SCORE FOR STEP DETECTION ALGORITHM ON STEP LENGTH TRAINING/TEST SET

	precision(%)	recall(%)	f-score(%)
start/end			
train	97.4 / 97.1	98.1 / 97.8	97.7 / 97.4
test	94.9 / 94.9	100 / 100	97.4 / 97.4

TABLE XII
MAE FOR STEP LENGTH ESTIMATION USING FEATURE SELECTION [31], CNN [28], AND PROPOSED DEEP LEARNING APPROACH ON TRAINING/TEST SET TAKING INTO ACCOUNT STEP DETECTION RESULTS

mae(cm)	feature selection	deep learning	[28]
train: 1^{st} / 2^{nd} way	4.20 / 6.49	2.46 / 4.91	2.66 / 4.94
test: 1^{st} / 2^{nd} way	5.14 / 7.71	3.54 / 6.56	3.70 / 6.00

and test sets. In contrast with the first way, we now notice that the CNN-based network results in a better performance on the test set, while, for the training set, the performance is comparable to the proposed algorithm. The reason for this is that Hannink et al. [28] also use the gyroscope data to predict the step length. Without the gyroscope data, we observed that the proposed step length estimator performs better than the CNN-based estimator. Hence, if performance is of the utmost importance, it could be beneficial to also include the gyroscope data in the proposed algorithms. Comparing the second with the first way to determine the mae, we observe that, as expected, the performance degrades for all approaches. Taking all results into account, we can conclude that the deep learning results in the best, i.e., lowest, mae on the step length.

V. CONCLUSION

In this work, we propose a deep-learning-based step detector and a step length estimator that only use the accelerometer data of a handheld IMU. For the step detector, a network consisting of several LSTM layers is used to predict if a step starts or ends. Using the output of this network and taking into account that the detection of the start and end of a step are related, we determine the final prediction of the time instants where a step starts and/or ends, which allows extracting the accelerometer data that correspond to a step. In most previous works, generally, only the performance on the number of predicted steps is investigated and not if the steps are detected at the right time, while, in this article, we also take into account if the steps are detected at the right time. This approach results in an f-score of 99.3% for detecting the start of a step and an f-score of 99.1% for detecting the end of a step.

For the deep-learning-based step length estimator, we use a network that consists of several convolutional and LSTM layers, where the idea is that the convolutional layers extract useful features from the measured acceleration to predict the step length, and the LSTM layers take into account the information gathered from previous steps. The results of this step length estimator are then also compared with the feature selection-based step length estimator of our previous work [31] and a CNN-based state-of-the-art approach [28]. Assuming that the boundaries of each step are known, this results in an mae of 3.21 cm with the deep learning approach, while the feature selection and CNN-based approach result in an mae of, respectively, 5.15 and 3.30 cm. In practice, however, we also need to take into account that a step detector can make errors, and hence, the step boundaries are not perfectly known. To this end, we combine the proposed deep-learning-based step detector with multiple step length estimators and evaluate the performance. Taking into account false positive and false negative step detections to evaluate the step length estimator, we obtain an mae of 6.56 cm with the proposed deep-learning-based step length estimator, while the feature-selection-based step length estimator and the CNN-based step estimator, respectively, result in an mae of 7.71 and 6.00 cm. Hence, we can conclude that, although the performance of the proposed step length estimator degrades more when we take into account false negative and positive step detections, it still outperforms the feature selection-based step length estimator. Furthermore, we observed that Hannink et al. [28] perform better than the proposed algorithm when we combine it with the step detection algorithm due to the use of the gyroscope data.

REFERENCES

[1] T. Do-Xuan, V. Tran-Quang, T. Bui-Xuan, and V. Vu-Thanh, "Smartphone-based pedestrian dead reckoning and orientation as an indoor positioning system," in *Proc. ATC*, Oct. 2014, pp. 303–308.

[2] H. Zhang, W. Yuan, Q. Shen, T. Li, and H. Chang, "A handheld inertial pedestrian navigation system with accurate step modes and device poses recognition," *IEEE Sensors J.*, vol. 15, no. 3, pp. 1421–1429, Mar. 2015.

[3] P. Kasebzadeh et al., "Improved pedestrian dead reckoning positioning with gait parameter learning," in *Proc. 19th Int. Conf. Inf. Fusion (FUSION)*. ISIF, 2016, pp. 379–385.

[4] S. Beauregard et al., "Pedestrian dead reckoning: A basis for personal positioning," in *Proc. WPNC*, 2006, pp. 27–35.

[5] J.-O. Nilsson, A. K. Gupta, and P. Handel, "Foot-mounted inertial navigation made easy," in *Proc. IPIN*, Oct. 2014, pp. 24–29.

[6] A. Perttula, H. Leppäkoski, M. Kirkko-Jaakkola, P. Davidson, J. Collin, and J. Takala, "Distributed indoor positioning system with inertial measurements and map matching," *IEEE Trans. Instrum. Meas.*, vol. 63, no. 11, pp. 2682–2695, Nov. 2014.

[7] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," *IEEE Trans. Syst., Man Cybern. C, Appl. Rev.*, vol. 37, no. 6, pp. 1067–1080, Nov. 2007.

[8] Y. Gu, A. Lo, and I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Commun. Surveys Tuts.*, vol. 11, no. 1, pp. 13–32, 1st Quart., 2009.

[9] J. Gao, P. Gu, Q. Ren, J. Zhang, and X. Song, "Abnormal gait recognition algorithm based on LSTM-CNN fusion network," *IEEE Access*, vol. 7, pp. 163180–163190, 2019.

[10] M. Z. Arshad, D. Jung, M. Park, H. Shin, J. Kim, and K.-R. Mun, "Gait-based frailty assessment using image representation of IMU signals and deep CNN," in *Proc. EMBC*, Nov. 2021, pp. 1874–1879.

[11] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao, "A reliable and accurate indoor localization method using phone inertial sensors," in *Proc. UbiComp*, 2012, pp. 421–430.

[12] M. Alzantot and M. Youssef, "UPTIME: Ubiquitous pedestrian tracking using mobile phones," in *Proc. WCNC*, Apr. 2012, pp. 3204–3209.

[13] H.-H. Lee, S. Choi, and M.-J. Lee, "Step detection robust against the dynamics of smartphones," *Sensors*, vol. 15, no. 10, pp. 27230–27250, 2015.

[14] E. M. Diaz and A. L. M. Gonzalez, "Step detector and step length estimator for an inertial pocket navigation system," in *Proc. IPIN*, Oct. 2014, pp. 105–110.

[15] A. Brajdic and R. Harle, "Walk detection and step counting on unconstrained smartphones," in *Proc. UbiComp*, Sep. 2013, pp. 225–234.

[16] J. Lin et al., "A decision tree based pedometer and its implementation on the Android platform," *Comput. Sci. Inf. Technol.*, vol. 5, pp. 73–83, Feb. 2015.

[17] J. Kupke, T. Willemsen, F. Keller, and H. Sternberg, "Development of a step counter based on artificial neural networks," *J. Location Based Services*, vol. 10, no. 3, pp. 161–177, Jul. 2016.
[18] N. Al Abiad, Y. Kone, V. Renaudin, and T. Robert, "SMARTphone inertial sensors based STEP detection driven by human gait learning," in *Proc. IPIN*, Nov. 2021, pp. 1–8.
[19] S. Vandermeeren, S. Van De Velde, H. Bruneel, and H. Steendam, "A feature ranking and selection algorithm for machine learning-based step counters," *IEEE Sensors J.*, vol. 18, no. 8, pp. 3255–3265, Apr. 2018.
[20] M. Edel and E. Koppe, "An advanced method for pedestrian dead reckoning using BLSTM-RNNs," in *Proc. IPIN*, Oct. 2015, pp. 1–6.
[21] L. Luu, A. Pillai, H. Lea, R. Buendia, F. M. Khan, and G. Dennis, "Accurate step count with generalized and personalized deep learning on accelerometer data," *Sensors*, vol. 22, no. 11, p. 3989, May 2022.
[22] Z. Chen, "An LSTM recurrent network for step counting," 2018, *arXiv:1802.03486*.
[23] B. Lin, "Machine learning and pedometers: An integration-based convolutional neural network for step counting and detection," Ph.D. thesis, Dept. Elect. Comput. Eng., Clemson Univ., Clemson, SC, USA, 2020.
[24] H. Weinberg, "Using the ADXL202 in pedometer and personal navigation applications," Analog Devices, Norwood, MA, USA, Appl. Note AN-602, 2002.
[25] J. W. Kim, H. J. Jang, D.-H. Hwang, and C. Park, "A step, stride and heading determination for the pedestrian navigation system," *J. Global Positioning Syst.*, vol. 3, nos. 1–2, pp. 273–279, Dec. 2004.
[26] Q. Tian, Z. Salcic, K. Wang, and Y. Pan, "A multi-mode dead reckoning system for pedestrian tracking using smartphones," *IEEE Sensors J.*, vol. 16, no. 7, pp. 2079–2093, Apr. 2016.
[27] V. Renaudin, M. Susi, and G. Lachapelle, "Step length estimation using handheld inertial sensors," *Sensors*, vol. 12, no. 7, pp. 8507–8525, 2012.
[28] J. Hannink et al., "Mobile stride length estimation with deep convolutional neural networks," *J. Biomed. health Informat.*, vol. 22, no. 2, pp. 354–362, Mar. 2018.
[29] Z. Ping, M. Zhidong, W. Pengyu, and D. Zhihong, "Pedestrian stride-length estimation based on bidirectional LSTM network," in *Proc. CAC*, Nov. 2020, pp. 3358–3363.
[30] Q. Wang, L. Ye, H. Luo, A. Men, F. Zhao, and Y. Huang, "Pedestrian stride-length estimation based on LSTM and denoising autoencoders," *Sensors*, vol. 19, no. 4, p. 840, Feb. 2019.
[31] S. Vandermeeren, H. Bruneel, and H. Steendam, "Feature selection for machine learning based step length estimation algorithms," *Sensors*, vol. 20, no. 3, p. 778, Jan. 2020.
[32] S. Herath, H. Yan, and Y. Furukawa, "RoNIN: Robust neural inertial navigation in the wild: Benchmark, evaluations, & new methods," in *Proc. ICRA*, May 2020, pp. 3146–3152.
[33] C. Chen et al., "IONet: Learning to cure the curse of drift in inertial odometry," in *Proc. AAAI*, vol. 32, 2018, pp. 1–9.
[34] H. Yan et al., "RIDI: Robust IMU double integration," in *Proc. ECCV*, 2018, pp. 621–636.
[35] O. Asraf, F. Shama, and I. Klein, "PDRNet: A deep-learning pedestrian dead reckoning framework," *IEEE Sensors J.*, vol. 22, no. 6, pp. 4932–4939, Mar. 2022.
[36] F. Chollet et al. (2015). *Keras*. [Online]. Available: https://keras.io
[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

Stef Vandermeeren was born in Jette, Belgium, in 1992. He received the B.E. degree, the M.Sc. degree in electrical engineering, and the Ph.D. degree from Ghent University, Ghent, Belgium, in 2013, 2015, and 2022, respectively.

His research interests are in the general area of sensor fusion, indoor localization, and machine/deep learning.

Heidi Steendam (Senior Member, IEEE) received the M.Sc. degree in electrical engineering and the Ph.D. degree in applied sciences from Ghent University, Ghent, Belgium, in 1995 and 2000, respectively.

Since September 1995, she has been with the Digital Communications (DIGCOM) Research Group, Department of Telecommunications and Information Processing (TELIN), Faculty of Engineering, Ghent University, first in the framework of various research projects, where she has been a Professor of Digital Communications since October 2002. In 2015, she was a Visiting Professor with Monash University, Clayton, VIC, Australia. She is the author of more than 150 scientific papers in international journals and conference proceedings, for which she received several best paper awards. Her main research interests are in statistical communication theory, carrier and symbol synchronization, bandwidth-efficient modulation and coding, cognitive radio and cooperative networks, positioning, and visible light communication.

Dr. Steendam was active in various international conferences as a technical program committee chair/member and the session chair. Since 2002, she has been an Executive Committee Member of the IEEE Communications and Vehicular Technology Society Joint Chapter, Benelux Section, where she has been the Vice-Chair since 2012 and the Chair since 2017. In 2004 and 2011, she was the Conference Chair of the IEEE Symposium on Communications and Vehicular Technology in Benelux. From 2012 to 2017, she was an Associate Editor of IEEE TRANSACTIONS ON COMMUNICATIONS and *EURASIP Journal on Wireless Communications and Networking*.