A Bidirectional LSTM for Estimating Dynamic Human Velocities from a Single IMU

Tobias Feigl,[†*] Sebastian Kram,[*] Philipp Woller,[*] Ramiz H. Siddiqui,[*] Michael Philippsen,[†]
and Christopher Mutschler[*‡]
{ tobias.feigl | michael.philippsen }@fau.de
{ sebastian.kram | philipp.woller | siddiqrz | christopher.mutschler } @iis.fraunhofer.de

[*]Precise Positioning and Analytics Department,
Fraunhofer Institute for Integrated Circuits IIS,
Nümberg, Germany

[†]Programming Systems Group,
[‡]Machine Learning and Data Analytics Lab,
Friedrich-Alexander University Erlangen-Nürnberg (FAU), Germany

***Abstract*—The main challenge in estimating human velocity from noisy Inertial Measurement Units (IMUs) are the errors that accumulate by integrating noisy accelerometer signals over a long time. Known approaches that work on step length estimation are optimized for a specific application, sensor position, and movement type, require an exhaustive (manual) parameter tuning, and can thus not be applied to other movement types or to a broader range of applications. Moreover, varying dynamics (as they are present for instance in sports applications) cause abrupt and unpredictable changes in step frequency or step length and hence result in erroneous velocity estimates.**

We use machine learning (ML) and deep learning (DL) to estimate a human's velocity. Our approach is robust to varying motion states and orientation changes in dynamic situations. On data from a single un-calibrated IMU, our novel recurrent model not only outperforms the state-of-the-art on instantaneous velocity (≤0.10 m/s) and on traveled distance (≤29 m/km). It can also generalize to different and varying rates of motion and provides accurate and precise velocity estimates.

I. Introduction

Feet- [1], [2] or torso-mounted [3]–[5] 6DoF IMUs (accelerometers and gyroscopes) can be used to estimate stable position and velocity without the need of any additional external sensors and systems [6], [7]. However, for many real-world applications it is inadequate to place IMUs on the feet or the spine. But known estimation methods fail with other more convenient placements, e.g., in a user's pocket.

Velocity and traveled distance of human motion are hard to capture accurately and robustly with only inertial sensors because of rapid and abrupt changes in motion, individual gait, and the dependence on a known, static sensor orientation [1].

Naive approaches integrate the acceleration over time to estimate velocity: By permanently estimating the object's orientation and subtracting the gravitation component from the acceleration signal, this yields the object's acceleration in its navigation frame (and integration over time returns its velocity). But due to sensor noise (and other artifacts) this is only stable for a short while, making such approaches often only useful as add-ons to multi-sensor fusion systems [8].

978-1-7281-1788-1/19/$31.00 © 2019 IEEE

Advanced approaches use (peak) detection for steps and estimate their lengths. Combinations with heading estimation [5] yield more exact velocity estimates [9], [10]. But such approaches require a manual parameter tuning which also makes them highly dependent on the sensor's mounting position [6]. And they are often limited to (walking) motion with little variations and fail to generalize to broader motion types, e.g., running, or to the transitioning between them [10].

This paper considers IMU-based velocity estimation as a regression problem that we address with machine learning. We use a single un-calibrated low-cost IMU and capture movement data in an extensive measurement campaign with a millimeter-accurate optical reference system. We compare several ML- and DL-approaches with state-of-the-art methods and show that our novel bidirectional LSTM (long short-term memory) architecture learns to map the IMU signals to different movement velocities in motion states such as walking, jogging, running, and random (a natural combination of all) of different subjects. This also works robustly under dynamically varying IMU orientations as we only use the signal magnitude vector (SMV).

Our experiments show that while classical methods and ML-approaches (on extracted features) cannot generalize to different and varying rates of motion, our DL-approach (on the raw measurements) provides accurate and precise velocity estimates. Our recurrent DL method directly estimates from the history of raw accelerations and raw angular velocities and outperforms the state-of-the-art on instantaneous velocity ($\leq 0.10 m/s$) and on traveled distance ($\leq 29 m/km$) [8], [11].

The paper is organized as follows. Sec. II reviews related work. Sec. III formalizes the problem. Sec. IV shows our processing pipeline and explains our methods. Sec. V covers the data acquisition and introduces our experimental setup. Sec. VI evaluates our methods and discusses results.

II. Related Work

While early approaches were limited to moving vehicles in military applications [12], low-cost IMUs are now used in many application areas such as robotics and mobile devices [1], [3]–[5]. However, due to limited size and costs their accuracy is limited. Hence, such IMUs are only used together with other sensors such as visual-inertial odometers [13].

Pedestrian Dead Reckoning (PDR) uses inertial measurements to detect steps and estimate stride length and direction by means of empirical formulas [14]. Approaches that yield errors below 0.4 m/s typically assume a rigid device orientation that is aligned with the direction of travel [10], [15]. Hence, such methods suffer from more complex and varying movements (e.g., abrupt changes or varying velocities) as orientation highly fluctuates. The main challenge is that an incorrect segmentation of the step shift leads to an inaccurate step length estimation. To compensate, a number of parameters must be tuned to the user's walking habits and the specific use-case [16]. Current research focuses on fusing PDR with external sensors such as WiFi [11], UWB [8], magnetic fields [17], and environmental information [18], [19] as the technical challenges of sensor drift and the decomposition of the acceleration are still unresolved. Methods that use frequency-based parameters that are invariant to rotation still suffer from bad accuracy [1], [4].

Biomechanical models exploit knowledge on the mechanics of the human body in the step phase. However, as movements are usually complex such mechanics can only be approximated [1], [20]–[22]. Usually, different models for different IMU positions are used (spine, hand, pocket [1], [23], foot [1], [21], and head [5]). As the mechanics also differ between movement types (e.g., walking and running) they need specific step length estimators [4], [21], [23]. Even clever combinations of step detection with gait analysis suffer from orientation variance [6], [7], [24].

Hybrid **machine learning** methods [2], [9], [23], [25]–[27] first classify steps and then select an adequate step length estimator. Most of them also exploit zero updating velocity moments (ZUPT) when the foot touches the ground [2], [6], [28] which can be used to eliminate the sensor drift (error <0.25 m/s) [2], [19]. Unfortunately, at higher movement velocities both the availability and confidence of ZUPT-moments decrease due to noise and motion artifacts [2], [6], [7], [23]. This often misleads such hybrid methods [7], [9] to use a wrong velocity estimator which results in a low accuracy [27]. While IMU-based velocity estimation has been proposed as a regression problem in [29], [30] the sensor placement was rigid and free from noise which cannot be assumed in the real world. Preliminary work mainly focuses on walking to simplify velocity estimation.

Current DL-methods often combine various sensors to address issues in the field of relative localization, e.g., visual-inertial odometer [13]. Or they fuse at least two [18] or military-grade IMUs [7] to average accumulating errors and to raise the confidence of their algorithms. Others learn intuitive physics [5] and design state-space models [20] or monitor neural networks via physical knowledge [14], [22]. However, they all suffer from the same problem: they either rely on tightly coupled sensor fusion [31] or on the availability of reliable contextual information (like external sensors [1], [8], [17], [31] or maps [11], [18], [19]). Thus, their accuracy heavily depends on the external information source.

Recent studies use deep (recurrent) networks to either denoise a signal [16], [26], [32], detect steps [2], [9], [23], or estimate their length [2], [9], [25]. But since such a split of the problem forces the networks to deal with multiple interdependent errors, these methods cannot learn the important error relationship. In contrast, our method "black-boxes" all errors end-to-end, i.e., from acceleration to reference velocity, keeps their interconnection, and provides direct inertial odometry (with higher reliability and accuracy) on a single orientation-invariant and uncalibrated inertial sensor in dynamic motion.

III. Problem Description

To provide accurate velocities, a raw acceleration acc signal must be decomposed into its linear and gravitation components. As velocity is typically derived from the linear component (per axis) we must estimate the gravity component (per axis) and subtract it from the raw signal (per axis). The accuracy of this estimation depends on the placement of the sensor, its orientation w.r.t. the body part it is attached to, the motion state, and of course noise: When the zero-mean Gaussian noise component c is low (e.g., when the sensor is not moving) we can reliably determine the initial biases (i.e., calibration offsets), by *projecting* the rotation of the acc signal into the navigation frame, subtracting the gravity, and integrating to obtain velocity. However, dynamic movements or a non-rigid mounting of the sensor increase this complexity of the modeling (as c is high) [15]. But as in dynamic movements (e.g., sports), orientation estimation is inaccurate acc cannot be easily used for velocity estimation [25]. In addition, even tactical-grade IMUs only provide stable estimates in the short-run (as the error propagates through integration) and as noise is not low in dynamic situations.

To cope with these problems, we describe the IMU-based velocity estimation as a supervised regression problem [33] where we derive a functional relation between the numerical velocity and the IMU-data (i.e., the pattern) by fitting the mapping to labeled data. Through regression analysis, we derive a function that approximates the mapping from SMV to the reference velocity by employing a set of trained parameters. Furthermore, we show how to circumvent the error-prone *projection* when c is high. As we apply our methods directly on the signal magnitude vector (SMV) or on its characteristic features the calculation is rotation-invariant and does not require an orientation estimator.

Moreover, we also assume that only a single sensor is located on a smartphone within the pocket of the subject. Thus, the sensor is not rigidly connected to the leg of the subject, as the phone can move within the pocket and hence, our ML/DL methods must implicitly learn to project the body frame to the navigation frame via multiple coordinate frames (from pocket, over leg, to subject). In addition, our solution is applicable in the real-world for many scenarios as the pocket is a natural position for a smartphone.

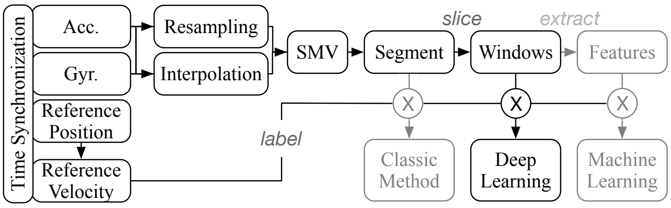


Fig. 1. Our processing pipeline.

IV. Method

A. Processing Pipeline and Pre-Processing

Fig. 1 sketches our processing pipeline. The gray elements are only used to provide the input data for the implementation of state-of-the-art baselines for our benchmarks, see Sec. VI. First, we pre-process the raw input data (from accelerometer acc and gyroscope gyr) by linear interpolation and resampling, and calculate the signal magnitude vectors (SMVs). Second, we process the input data, i.e., the set of acc- and gyr-measurements that represent all activities (walking, jogging, running, and random, of one specific subject) into segments s. Next, we slice each s into consecutive windows w. While our DL approach works directly on w, we extract features f_w from w for the ML methods, see Sec. V-B. Finally, we split the data into train, validation, and test sets and train our estimators on labeled data. At run-time, we predict velocities from unknown input data using the same pipeline.

In a preliminary study, we evaluated different sampling rates f_s (50, **100**, 200, 400) of the sensors and different window sizes N_w (64, **128**, 256, 512) to find combinations of f_s and N_w that are computationally efficient and still yield accurate velocity estimates. We found the bold settings to embed enough prominent characteristics (1.28s already cover long-term relations of human motion that enable the applicability of our DL method) at low computational costs. A segment length of 20 $seconds$ and sliding windows with 50% overlap yield the best performance.

On each resampled and interpolated segment s we calculate *SMVs*. As we have two input streams (acc and gyr) we studied the following two processing options: an intuitive 2-dimensional (2D) representation, i.e., $2 \times N_w \times \#(s)$:

$$SMV(s(acc)) = \sqrt{(acc_x^2 + acc_y^2 + acc_z^2)} \text{ and}$$
$$SMV(s(gyr)) = \sqrt{(gyr_x^2 + gyr_y^2 + gyr_z^2)},$$

and a 1D combination of acc and gyr, i.e., $1 \times N_w \times \#(s)$:

$$SMV(s(acc, gyr)) = \sqrt{(acc^2 + gyr^2)}.$$

Finally, to evaluate our methods we compute two different reference variables: the instantaneous velocity v_{ref} per window w and the covered reference distance d_{ref} per segment s. We calculate v_{ref} by differentiating the 2D velocities, i.e., the translational layer, spanned by the x- and y-axes obtained from the reference system (w.r.t. time and norm). We calculate d_{ref} accordingly, but this time we differentiate all windows of the segment. We determine the errors based on v_{ref} and d_{ref}.

B. Deep Learning Models

Neural networks such as multi-layer perceptrons (MLPs) are composed of an input layer X, one or more hidden layers H, and an output layer Y. Input signals propagate through the network, a loss-function compares the output to the ground truth label to compute the error, error signals propagate backward through the network, and weights are adjusted to reduce the error [34]. As all MLP-variants only take a fixed length of input measurement values m, i.e., acc and gyr samples, at time step i to predict a velocity v_i, they cannot describe the relationship of time and context across consecutive measurements. But those are relevant for motion as it is a temporal-dynamic process that depends on different factors like acceleration, velocity, and direction.

Thus we propose to use time- and context-sensitive neural networks to learn such dependencies within a single window and across several windows. Since recurrent neural networks (RNN) such as Long-Short-Term-Memory LSTM- or bi-directional LSTMs (BLSTMs)-networks learn both short-term dependencies (within a single window) and long-term dependencies (between windows) they capture (a) the relation of each measurement in a window to all other measurements in the same window (i.e., how each measurement affects other measurements in the same window) and (b) their dependencies within the full segment (i.e., how each measurement affects the measurements in another window, e.g., how movements change between windows) [32].

An obvious approach is to take hand-crafted features (that we also use for ML-methods, see Sec. V-B) and to use them as an input to the recurrent network. However, convolutional neural networks (CNNs) proved to be much better feature extractors (while they cannot set these features into time-context). Thus we propose a model that uses a CNN to extract high-level features from raw input signals and connect them trough time with an RNN model.

Fig. 2 shows our CNN-BLSTM model. Data propagates from the bottom to the top. After an initial unfolding of the data from our batch, 1D-convolution kernels extract high-level features, while a 1D filter kernel f_k slides over a sequence (i.e., window vector w_v) to detect features at different positions [35]. The idea behind the 1D convolution is to perform element-by-element multiplication of a 1D filter kernel f_k with each 1D w_v on d windows to obtain a time-sensitive feature map [36]. We use two such layers to obtain high-level features.

Now that we have extracted high-level features from the SMV, we directly put them into our BLSTM without any added pooling (max. and average pooling would remove valuable sequential organization [36], [37]). Thus, the BLSTM tracks the emergence of features over time. We extract bottleneck features with a many-to-one (i.e., $X > Y$=1) BLSTM architecture. Unlike LSTM, our BLSTM processes information about the future and the past as it retrieves short-term long-term contexts of input to the current output in both the forward and backward directions [35]. The BLSTM acts as a time- and context-sensitive detector and tracker of high-level motion

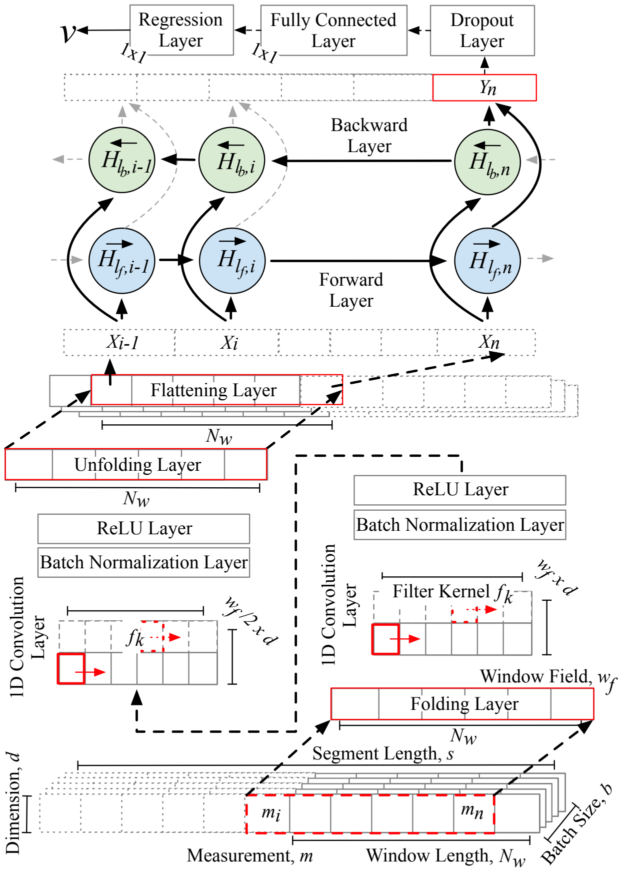


Fig. 2. The architecture of our CNN-BLSTM network. From bottom to top: First we fold windows with dimension $d \times N_w$ each to a corresponding array ($w_f \times d$); the first 1D convolution layer applies k=128 filter kernels $f_k = N_w$ each with size 1×3 (i.e., $\#k = N_w$; we found that one filter kernel f_k, i.e., one feature [35], per measurement value m yields accurate results) followed by a batch normalization (BN) and a rectified linear unit (ReLU) layer; the second 2D convolution layer ($w_f/2 \times d$) applies N_w filter kernels with size 1×3, again followed by BN and ReLU; if d==2, an unfolding layer projects the feature map back to its initial sequence structure according to the batch's info that is passed trough the (un)folding layers; a flattening layer provides a flat sequence to the BLSTM's input layer; the BLSTM's last layer acts as a bottleneck and yields the latest Y_n that is processed through a dropout layer; a fully connected layer provides a 1×1 output to the regression layer that computes the loss with a half-mean-squared error.

features (which the convolution layers extract) on the signal embedding (acc and gyr). Two hidden layers (H_l and H_{l+1}) provide forward and backward directions. Following previous work [16], we use the last hidden layer as the bottleneck layer.

Note, that the sequence length s defines how many consecutive measurement values m the model processes in each iteration. The longer the sequence is, the more motion dependencies the model learns to interpret correctly, but the more complex the model gets [32]. The model learns how to obtain a single velocity from a single set of measurement values and to predict consecutive velocities. While X_i represents an input set at time-step i, the resulting velocity estimation is Y_i. With a randomly initialized initial state H_0, the model does not have prior knowledge about the motion and consequently learns to estimate the velocity from local information.

During the training phase, our model learns to leverage latent motion characteristics from a consecutive sequence s=w of data points m per window w_i and their corresponding reference velocity value v_{ref}. At run-time it predicts the current accurate and precise velocity of a subject on a window w_i of raw acc and gyr measurements. We take the last velocity estimate Y_n as the final velocity estimate v per window that is estimated by a final regression layer.

V. Experimental Setup

Measurement Area. We record the motion data (i.e., acc and gyr) at the Fraunhofer IIS L.I.N.K. (localization, identification, navigation, communication) testing facility in Nuremberg. It provides a unique test ground on 1,400 m^2, see Fig. 3 (left) [38]. We used the following measurement and reference systems to collect and label our training, test, and evaluation data, and to capture the properties of different movement types (i.e., walking, jogging, running, and random).

Reference System. We recorded reference data with 28 cameras of the millimeter-accurate optical Qualisys motion tracking system (spherical error probable (SEP_{95}) $\leq 5mm$ and $\leq 0.1°$). The cameras are mounted on the edges of the upper side of the test area and cover a volume of 11.025 m^3 (45 × 35 × 7 m). Subjects wear 4 small trackable reflective markers, attached to an elastic ribbon, see Fig. 3 (right). All cameras permanently had a clear field of view when we tracked each subject's position and orientation at 100 Hz.

IMU Measurement Device. We use two Samsung Galaxy S7 smartphones with their acc and gyr sensors (STMicroelectronics LSM6DS3 samples acc at ±16 G and gyr at ±1000 dps at 100 Hz) to measure the subject's accelerations and angular rates. To cover motion differences between the two legs we loosely placed a phone (inverse portrait) in each of the pockets, see Fig. 3 (middle). Note: we use two phones to collect more data per subject, but our methods only use the data from a single phone.

IMU Software. We access the raw IMU sensor data of the smartphones via the Android API (Version 6, 2019) [27]. We store them along with timestamps that are globally NTP-time-synchronized. Our API also receives the NTP-time-synchronized 6DoF state information data as a stream from Qualisys via 5 GHz WiFi. We labeled each activity and measurement (acc, gyr, and reference data) and stored separate files per subject. Thus, our dataset can be used for activity classification as well.

A. Data Acquisition

16 people (male: 12, female: 4, avg. age: 23.4, height min. 146 cm, max. 186 cm, SD 17 cm) were asked to do 4 different

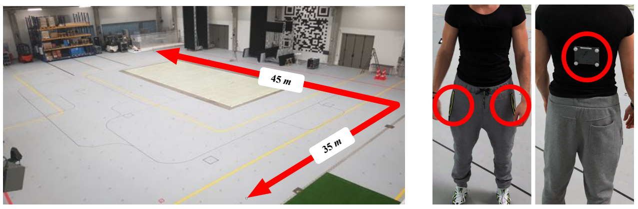


Fig. 3. Setup. Left: Lateral view of the data acquisition environment; Middle: Smartphones in the pockets of a trial subject; Right: Trackable object of the reference system.

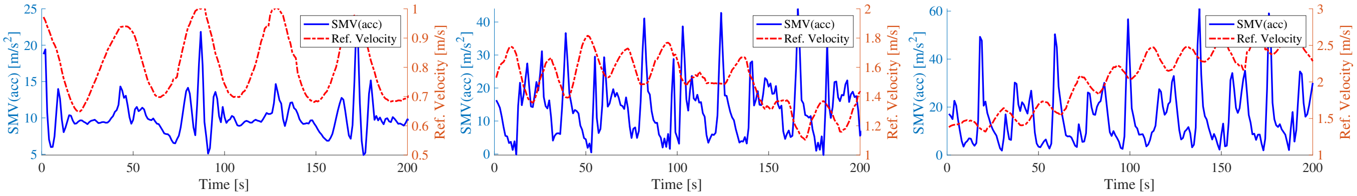


Fig. 4. Raw *acc* signals: SMV (blue), reference velocity (red).

types of movement activities (walking, jogging, running, and random movements, a natural combination of all) within our tracking area (one activity after another). After the completion of each activity, the recordings were analyzed for completeness and manually labeled (class according to activity). Each activity took 6 minutes on average. The walking, jogging, and running activities were performed at different speeds (*walking* on avg. 0.7; min. 0.5; max. 0.9; SD 0.2; *jogging* on avg. 1.3; min. 0.8; max. 2.4; SD 0.34; *running* on avg. 1.9; min. 1.2; max. 3.5; SD 0.6; $\frac{m}{s}$) but along similar movement trajectories. The random movements were performed differently by each participant (changing between walking, jogging, and running randomly during the measurement). Hence, the movement speed and trajectory was chosen individually by each subject (on avg. 0.8; min. 0.1; max. 3.4; SD 0.8; $\frac{m}{s}$).

The participants were asked to wear typical running sports clothes and to take the study as a serious workout. As different subjects wore different pants, with wide or tight pockets, this introduced various sensor noise and motion artifacts.

In total, we recorded approx. 800 min (approx. 24 min per subject) of motion data, see Fig. 5. Broken data records were both visually and statistically identified. Hence, to also equally distribute our dataset, 3 (out of 19) runs were completely removed from the dataset, short dropouts were reconstructed by means of resampling and interpolation. The synchronization of the time-stamps had an error of <1 ns/h due to clock drifts between the reference and the measuring sensor.

Fig. 4 shows the acceleration data (blue) and velocity data (red) for walking, jogging, and running. The signal complexity increases at higher velocities (i.e., frequency and noise increase while the peak prominence decreases from low/left to high/right velocities). The low peak prominence also explains the inefficiency of peak-detecting approaches (see Sec. VI) as they vanish in high and noisy velocities.

B. Parameterization of Velocity Estimators

Baseline I – Classic Step Length Estimation requires an additional pre-processing. We use a Butterworth low-pass filter with a cutoff frequency of 15 Hz to remove high-frequency noise and then detect steps in each segment s to estimate their length. Our step detection detects the peaks in either the acc or the gyr signals or both. For sensors in pockets, we found that the peak detection works best on the acc's SMV. We dropped complex hybrid models as they suffer from specific parameter selections which are especially problematic for noisy data or varying subjects with unknown K_s. Instead, we achieve the best estimates for the step length with frequency model $l = Kh\sqrt{f_s}$, where h is the height of the person, K is a calibration parameter, and f_s is the step frequency [16].

For our experiments we use two such models. First, we combine K and h into a *subject-activity*-specific calibration parameter K_s that we calibrate using ground truth data. We use the first few steps (i.e., 1 min) of each activity and determine K_s with the help of the reference system, i.e., using the difference between the true and the estimated traveled distance. Second, we derive a general calibration parameter K_g=1.2771, which is the average of all K_s calibration parameters for all our subjects (except for the left-out test subject). We use K_g as our *generalized* model. This allows a comparison of all our methods and is close to the typical use case in practice. We estimate the velocity by multiplying the step length (calculated by the model) with the delta step time, i.e., the difference between two consecutive steps.

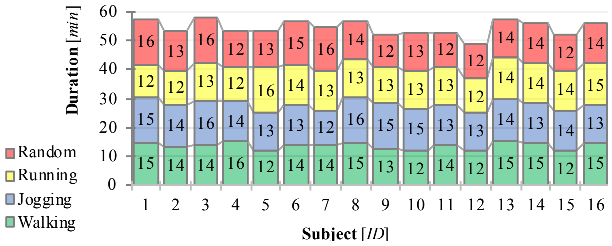


Fig. 5. Dataset statistics with a *quasi-equal* distribution.

Baseline II – Machine Learning with Gaussian Processes (ML-GP). For our ML-based baseline, we evaluate different regression methods on a selection of hand-crafted features. We carefully select and extract features from the windows (it is an elaborate engineering task to find adequate features that best fit the use case).

We extract both statistical and frequency domain [30] features per window, inspired by [33]. Frequency domain feature selection: spectral bandwidth, spectral flatness, i.e., the ratio of the geometric mean to the arithmetic mean of the magnitude spectrum of the signal, spectral roll-off, i.e., frequency below which a specified percentage of the total spectral energy lies, spectral centroid, i.e., the spectral centroid (mean), variance, skew, and kurtosis of the absolute Fourier transform spectrum.

Time-domain features: absolute energy, i.e., the absolute energy of the time series which is the sum over the squared values, maximum, minimum, mean, median, variance. While all time-domain features are straightforward to calculate, for the frequency domain features we first apply a short-time Fourier transform (STFT) to N_w. From the STFT-like representation of the time-frequency distribution, we then extracted the features. A handcrafted feature analysis and a dimensionality reduction (principal component analysis, PCA) reduce the number of features from 30 to only 2.

For our ML-based baseline, we also evaluate different regression methods: Classification and Regression Tree (CART), Support Vector Regression (SVR) [33] and Gaussian Processes [30], [39] (GP). Through a grid search, we obtained optimal hyper-parameters.[1] Among the estimators, GP with a Matern52 kernel yields the highest accuracy (but at higher computational costs). To cope with the high dimensionality of the data, we use a sparse GP. [30] We use this best configuration as the baseline of our ML-methods.

CNN-BLSTM Parameterization. For our evaluation, we also investigated different architectures such as LSTM, BLSTM, and CNN-LSTM. It turned out, that our CNN-BLSTM performed best. In a preliminary experiment, we kept all LSTM and BLSTM blocks identically w.r.t. the number of hidden layers and the number of LSTM cells to find the optimal architecture. For our experiments, two convolution layers and one BLSTM block with 128 LSTM cells yield the best accuracy at short inference times. A convolution with 128 filter vectors f_v=1×3 balances best between accuracy and inference time. We do not pad as our input windows have always a constant length. We set the dropout rate to 50%.

We evaluated different sample rates ($50, 100\ Hz$) and window lengths ($64, 128$) on either 1D or 2D SMVs of w. The sequence length s (i.e., s=w) at N_w=128 holds a maximum of 2.56 s (128/50 Hz) of motion data. We trained each set of parameters for 33 epochs, with early stopping based on the validation set performance to prevent over-fitting.[2]

VI. Results

We compare our DL-model with our baselines along with two variables: (1) the error of the velocity and (2) the error of the covered distance. To quantify them we calculate circular error probabilities (CEP), distance error per meter (DEPM), root mean square error (RMSE), mean absolute error (MAE), and mean squared error (MSE) on the velocities of both our validation and *unknown* test segment. We only discuss the results of the best models, see Sec. V-B.

A. Accuracy Estimation

To evaluate the accuracy we train our models on 80% and validate them on 20% of all our data.

From Fig. 6(a) we see that 50% of all estimates yield an absolute velocity error below 0.50 m/s for both the ML-GP and DL-methods, while with the Classic step length estimation 50% of all velocities have an error of more than 1 m/s and more than 2 m/s in 95% (at the same time ML-GP estimates velocities with an error below 1 m/s for 80% of the samples). DL outperforms ML in the interval between 80 and 95%, where the errors for ML-GP rises to 1.46 m/s whereas DL still only has an estimation error of 0.71 m/s. This shows the robustness of our DL-approach as it is not affected by outliers that much (in line with an RMSE of 0.35 m/s).

TABLE I
Error statistics. CEPx, RMSE, MSE, MAE in $[m/s]$ and $[m]$ and DEPM in $[m]$.

	N_W=128 f_S =100 *Hz*	CEP 25	 50	 75	 95	RMSE	MSE	MAE	DEPM
Validation	Classic	0.47	1.01	1.68	2.89	1.45	2.12	1.64	0.20
	ML-GP	0.13	0.31	0.69	1.46	0.70	0.50	0.49	0.11
	CNN-BLSTM	**0.04**	**0.11**	**0.23**	**0.71**	**0.35**	**0.12**	**0.20**	**7e-4**
Test	Classic	0.62	1.41	2.11	3.05	1.70	2.89	1.42	0.43
	ML-GP	0.13	0.33	0.71	1.47	0.68	0.46	0.49	0.21
	CNN-BLSTM	**0.09**	**0.18**	**0.33**	**0.63**	**0.32**	**0.10**	**0.24**	**0.03**

Fig. 6(b) shows the probability density function (PDF). The error distribution in the case of DL is nearly symmetric and forms a Gaussian-like distribution with a small SD while ML-GP has (only) a slight bias and a (slightly) larger SD. The Classic method instead shows a much larger SD with a low distribution around zero which indicates more errors.

Fig. 6(c) shows the deviation of the estimated from the reference traveled distances (remember that the sum of every window's velocity yields the total distance). Impressively, even after a total travel distance of 28,455 m, the DL method nearly fits the baseline as the error is only -20 m, while the Classic and the ML-GP methods are off by 4,955 m and -3955 m.

Table I shows the quintessence of velocity estimation errors (CEPx, RMSE, MSE, MAE) and the traveled distance error (DEPM) for both our validation and test dataset.

B. Generalization of our Methods

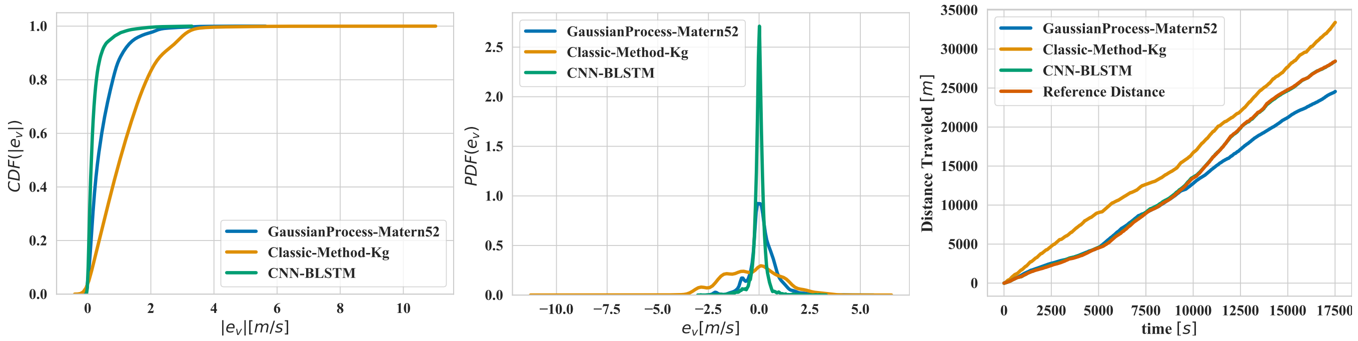
Left-out Subjects. To evaluate the generalization of our models, we used the data of 15 subjects for training and validation and tested on the data of a single subject. The test subject performed the same activities. The lower part of Table I shows the errors of the test subject. The Classic method (with K_g=1.2771) yields an RMSE of 1.7 m/s and an MAE of 1.4 m/s. The results show that the test subject's movement is quite *average* and fits the training data well, but still lags far behind ML-GP and DL. While ML-GP performs quite well on the test subject with an RMSE of 0.68 m/s and an MAE of 0.49 m/s (which is even better than on the validation set), the CNN-BLSTM outperforms this with an RMSE of 0.32 m/s and an MAE of 0.24 m/s. These results show that our novel CNN-BLSTM model generalizes well even to unknown data.

Left-out devices. To evaluate the generalization of our models, we used the Samsung Galaxy S7 data for training and validation and tested it on data of a Samsung Galaxy Note 4 (InvenSense MPU-6500, 6 DOF IMU). Although the test results showed no differences (∅SD <0.01%), data normalization of the sensors of different phones is necessary to cope with sensor-specific alignment and calibration offsets.

Distance Traveled. In a direct comparison of the traveled distance (validation and test) the Classic method quasi-linearly increases its error with growing distance, i.e., it overshoots. (Validation set: 33,410/28,455=1,174 m/km; test

[1] Avail. ML gridsearch parameters: SVR Linear C=100 [1,10,100,1000], SVR Poly. C=100 [1,10,100,1000], degree=3 [2:1:6], SVR RBF gamma=0.001 [0.001,0.0001], C=100 [1,10,100,1000] for every combination of f_s (50,100) and N_w (64,128), DT max. depth=97 [1:1:150], max. features=27 [1:1:30], max. leaf nodes=15 [1:1:20] [39].

[2] Avail. DL gridsearch parameter: solver: Adam [SGD, Adam, rmsprop] β_1, β_2=0.01, Momentum=0.9, B/LSTM layers=2 [1,2,3,4], cells/layer=128 [16,32,64,128,256], initial learning rate (lr)=0.01 [1.0:0.1:0.00001], lr drop rate per epoch=0.9, batch size=32 [1,10,16,32,64,128,256,500]; CNN f_v=1×3 [37].



(a) Absolute error CDF.

(b) Error PDF.

(c) Distance Traveled.

Fig. 6. Results on the validation set from left to right: absolute error CDF, error PDF, and distance traveled.

set: 9,666/5,184=1,864 m/km). However, while ML-GP can cope with the short distance of the test set (5,361/5,184=34 m/km), it undershoots significantly on the long validation run (-160 m/km). This implies that ML struggles in learning the physical relations of motion from handcrafted features. Instead, the DL method (-20 m) fits the traveled distance of 28,455 m and yields low error rates on both validation (0.7 m/km) and test (28 m/km). This implies that our DL method does not over-fit the data, as the test set is unknown and test and validation sets are different.

Random Movement. Although the results of the methods on our validation and test set already embed random activities, we explicitly evaluated our methods on the test subject's random movement. Since the results do not show anomalies, we only discuss the traveled distance error: The Classic method again overshoots by 759 m (1931/1173 m) while ML-GP (973/1173 m) and DL (1169/1173 m) again undershoot by -200 m and just -4 m.

C. CNN-BLSTM Long-Term Dependencies

From the architecture of our model, it is obvious that we cannot only handle short-term dependencies but also handle large data streams, i.e., 20 s sequences of windows. This is when our model's (un)folding layers and the for/backward extensions of BLSTM shine. Our model can learn long-term dependencies across consecutive windows and hence achieves even lower error rates on the test set (RMSE 0.21 m/s). In contrast, ML-GP (RMSE 1.21 m/s) does not benefit from longer data streams as the features blur out.

D. Further Findings

Computational Effort. In total, the training time is high for both ML-GP ($\varnothing n \times 33h$) and CNN-BLSTM ($\varnothing n \times 9h$), with n = # parameter sets. Instead, the inference time is low for all models (note: the architecture of our DL model is simple).

Impact of Parameters. A higher f_s results in redundant information per window, slows down the training process for DL ($\varnothing \geq 31$ h at 200 Hz), lowers valuable information content for ML-GP (test set: RMSE 1.21 m/s), and raises the computational effort for Classic methods. Instead, a lower f_s=50 increases computational performance, but yields less accurate estimates on the test set (RMSE: Classic 1.9, ML-GP 0.74, DL 0.32 m/s). However, this may still be an option for smartphone-based localization, when lower energy consumption is important. Our experiments revealed that DL on both 1D- and 2D-SMVs performs similarly (RMSE SD $\leq$0.01 m/s) but 1D-SMV saves computational costs.

Although ML-GP shows good and robust generalization effects, a medium accuracy, effort, and cost due to handcrafted features, DL yields the highest accuracy, at lower design costs, but at high pre-computational costs due to the long-lasting grid-search. The results showed that CNN-BLSTM surpasses all other models and networks. This demonstrates that the CNN-BLSTM takes advantage of the CNN's extraction of optimal local features and the treatment of sequential long-term contextual dependencies of RNN. As we are taking the CNN-BLSTM end-to-end, our CNN part also extracts features that impact the recurrent units. However, even our ML- and DL-methods suffer from the accumulation of relative errors. Even our DL method accumulates a noticeable error of 145 m (28 m/km) on an unknown test dataset (5,329/5,184 m). Hence, there are still research challenges ahead of us.

Although, dynamic movements and pronounced walking habits of an unknown test subject do not affect the quality of our DL method much (RMSE SD $\leq$0.01 m/s) there still may be unpredictable real-world situations that we do not cover yet. Thus, more data to increase the boundaries of generality, or training on synthetic data and retraining on real data may help here. Nevertheless, while we can align and map measurement values from the past to future labels in the training phase it is unclear how to predict future velocities at run-time.

VII. Conclusion

The paper compares current methods of ML and DL with classical methods for implicit gravity, noise, and signal artifact compensation to provide a high and robust velocity estimate both on a short and long duration of human motion. We present data-driven approaches that learn to map a velocity vector of varying velocities to an accurate reference velocity in dynamic scenarios. Our methods are invariant to a sensor's orientation and placement. Our evaluation shows that our novel CNN-BLSTM performs well in your pocket: instantaneous velocity error $\leq$0.10 m/s) and traveled distance error $\leq$29 m/km.

Future work will explicitly explore other types of movement, such as jumping and climbing stairs.

Acknowledgments

This work was supported by the Bavarian Ministry for Economic Affairs, Infrastructure, Transport and Technology through the Center for Analytics Data Applications (ADA-Center) within the framework of "BAYERN DIGITAL II".

We thank Christian Nickel and Maximilian Stahlke for technical support, Peter Bauer for helping with the organization and supervision of the measurement campaign, Christian Schöner for help with the graphical evaluation, and all participants of our department for providing data.

References

[1] L. E. Díez, A. Bahillo, J. Otegui, and T. Otim, "Step length estimation methods based on inertial sensors: A review," *Sensors*, vol. 18, no. 17, pp. 6908–6926, 2018.

[2] B. Wagstaff and J. Kelly, "Lstm-based zero-velocity detection for robust inertial navigation," in *Proc. Intl. Conf. Indoor Positioning and Indoor Navigation*, (Nantes, France), pp. 22–30, 2018.

[3] D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, V. Handziski, and S. Sen, "A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned," in *Proc. Intl. Conf. Indoor Positioning and Indoor Nav.*, (Banff, Canada), p. 178–189, 2015.

[4] J. Bravo, E. P. Herrera, and D. A. Sierra, "Comparison of step length and heading estimation methods for indoor environments," in *Proc. 24nd Intl. Conf. Electronics, Electrical Eng. and Comp.*, (Cusco, Peru), 2017.

[5] T. Feigl, C. Mutschler, and M. Philippsen, "Supervised Learning for Yaw Orientation Estimation," in *Proc. Intl Conf. on Indoor Positioning and Indoor Navigation*, (Nantes, France), pp. 87–95, 2018.

[6] Y. Wang, A. Chernyshoff, and A. M. Shkel, "Error analysis of zupt-aided pedestrian inertial navigation," in *Proc. Intl Conf. on Indoor Positioning and Indoor Navigation*, (Nantes, France), pp. 206–212, 2018.

[7] J. Hong Lee, H. Ju, and C. Gook Park, "Error analysis of pdr system using dual foot-mounted imu," *Sensors*, vol. 94, no. 1, 2019.

[8] C. Gentner and M. Ulmschneider, "Simultaneous localization and mapping for pedestrians using low-cost ultra-wideband system and gyroscope," in *Proc. Intl Conf. Indoor Positioning and Indoor Navigation*, (Sapporo, Japan), 2017.

[9] M. Edel and E. Köppe, "An advanced method for pedestrian dead reckoning using blstm-rnns," in *Proc. Intl Conf. Indoor Positioning and Indoor Navigation*, (Banff, Canada), 2015.

[10] J. Windau and L. Itti, "Walking compass with head-mounted IMU sensor," in *Proc. Intl. Conf. Robotics and Automation*, (Stockholm, Sweden), pp. 5542–5547, 2016.

[11] J. Seitz, L. Patino-Studencki, B. Schindler, S. Haimerl, J. Gutierrez, S. Meyer, and J. Thielecke, "Sensor Data Fusion for Pedestrian Navigation Using WLAN and INS," in *Symp. Gyro Technology*, (Bonn, Germany), 2007.

[12] A. M. Shkel, "Inertial mems sensors are becoming 3d and atomically precise," in *Proc. Intl. Conf. Indoor Positioning and Indoor Navigation*, (Nantes, France), 2018.

[13] M. Rantanen, J.and Mäkelä, L. Ruotsalainen, and M. Kirkko-Jaakkola, "Motion context adaptive fusion of inertial and visual pedestrian navigation," in *Proc. Intl. Conf. Indoor Positioning and Indoor Navigation*, (Nantes, France), pp. 206–212, 2018.

[14] A. M. Sabatini, "Quaternion-based extended kalman filter for determining orientation by inertial and magnetic sensing," *Trans. Biomed. Eng.*, vol. 53, no. 7, pp. 1346–1356, 2006.

[15] C. Combettes and V. Renaudin, "Comparison of misalignment estimation techniques between handheld device and walking directions," in *Proc. Intl. Conf. Indoor Positioning and Indoor Nav.*, (Banff, Canada), 2015.

[16] J. Changhui, C. Shuai, C. Yuwei, Z. Boya, F. Ziyi, Z. Hui, and B. Yuming, "A mems imu de-noising method using long short term memory recurrent neural networks (lstm-rnn)," *Sensors*, vol. 1, no. 3, p. 1281–1293, 2018.

[17] H. Hellmers, A. Eichhorn, A. Norrdine, and J. Blankenbach, "Imu/magnetometer based 3d indoor positioning for wheeled platforms in nlos scenarios," in *Proc. Intl. Conf. Indoor Positioning and Indoor Navigation*, (Barcelona, Spain), 2016.

[18] J. L. Carrera, Z. Zhao, T. Braun, and Z. Li, "A real-time indoor tracking system by fusing inertial sensor, radio signal and floor plan," in *Proc. Intl. Conf. Indoor Positioning and Indoor Navigation*, (Barcelona, Spain), 2016.

[19] K. Nguyen-Huu, K. Lee, and S. Lee, "An indoor positioning system using pedestrian dead reckoning with wifi and map-matching aided," in *Proc. Intl. Conf. Indoor Positioning and Indoor Navigation*, (Sapporo, Japan), 2017.

[20] J. Hooman, W. Roever, P. Pandya, Q. Xu, P. Zhou, and H. Schepers, "A compositional object-based approah to learning physical dynamics," in *Proc. Intl. Conf. Learning Representations*, (Toulon, France), p. 155–173, 2017.

[21] D. Kim, H. Ju, and C. Gook Park, "Comparison of step length estimation models using inertial sensor on pelvis," in *Proc. Intl. Conf. Indoor Positioning and Indoor Navigation*, (Nantes, France), pp. 222–225, 2018.

[22] R. Stewart and S. Ermon, "Label-free supervision of neural networks with physics and domain knowledge," in *Proc. 31st AAAI Conf. Artificial Intelligence*, (San Francisco, CA), pp. 2576–2582, 2017.

[23] F. Gu, K. Khoshelham, C. Yu, and J. Shang, "Accurate step length estimation for pedestrian dead reckoning localization using stacked autoencoders," *Trans. Instrum. and Meas.*, vol. 2, no. 1, pp. 1–9, 2018.

[24] 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. 1, p. 840, 2019.

[25] C. Chen, P. Zhao, C. Xiaoxuan Lu, W. Wang, A. Markham, and A. Trigoni, "Oxiod: The dataset for deep inertial odometry," *CoRR*, vol. 13, no. 1, p. 1281–1293, 2018.

[26] C. Chen, C. Xiaoxuan Lu, A. Markham, and A. Trigoni, "Ionet: Learning to cure the curse of drift in inertial odometry," in *Proc. 32nd AAAI Conf. Artificial Intelligence*, (New Orleans, LA), p. 12–17, 2018.

[27] Y. Hang, S. Qi, and F. Yasutaka, "Ridi: Robust imu double integration," in *Proc. European Conf. Computer Vision*, (Munich, Germany), p. 111–119, 2018.

[28] J. Wahlström, I. Skog, F. Gustafsson, A. Markham, and N. Trigoni, "Zero-velocity detection – a bayesian approach to adaptive thresholding," *Sensors*, vol. 2017, no. 1, 2019.

[29] S. Zihajehzadeh and E. J. Park, "Regression model-based walking speed estimation using wrist-worn inertial sensor," *PLOS ONE*, vol. 11, no. 1, pp. 22–31, 2016.

[30] H. Vathsangam, A. Emken, D. Spruijt-Metz, and G. S. Sukhatme, "Toward free-living walking speed estimation using gaussian process-based regression with on-body accelerometers and gyroscopes," in *Intl. Conf. Perv. Computing Technologies for Healthcare*, (Munich, Germany), 2010.

[31] V. Renaudin, M. Ortiz, and J. Le Scornec, "Foot-mounted pedestrian navigation reference with tightly coupled gnss carrier phases, inertial and magnetic data," in *Proc. Intl. Conf. Indoor Positioning and Indoor Navigation*, (Sapporo, Japan), pp. 14–19, 2017.

[32] T. Feigl, T. Nowak, M. Philippsen, T. Edelhäußer, and C. Mutschler, "Recurrent Neural Networks on Drifting Time-of-Flight Measurements," in *Proc. Intl Conf. Indoor Positioning and Indoor Navigation*, (Nantes, France), pp. 112–120, 2018.

[33] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[34] W. Shao, H. Luo, F. Zhao, C. Wang, A. Crivello, and M. Z. Tunio, "Depedo: Anti periodic negative-step movement pedometer with deep convolutional neural networks," in *Proc. IEEE Intl. Conf. Communications*, (Kansas City, MO), 2018.

[35] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," in *Proc. 52nd Ann. Meeting Assoc. for Comp. Linguistics*, (Baltimore, Maryland), pp. 655–664, 2014.

[36] L. Li, Z. Wu, M. Xu, H. Meng, and L. Cai, "Combining cnn and blstm to extract textual and acoustic features for recognizing stances in mandarin ideological debate competition," in *Intl. Conf. INTERSPEECH, San Francisco, CA*, pp. 1392–1396, 2016.

[37] M. Startsev, I. Agtzidis, and M. Dorr, "1d cnn with blstm for automated classification of fixations, saccades, and smooth pursuits," *Behavior Research Methods*, vol. 51, no. 2, pp. 556–572, 2019.

[38] A. Niitsoo, T. Edelhäußer, E. Eberlein, N. Hadaschik, and C. Mutschler, "A Deep Learning Approach to Position Estimation from Channel Impulse Responses," *Sensors*, vol. 19, no. 5, pp. 1064–1087, 2019.

[39] C. E. Rasmussen and E. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.