2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 24-27 September 2018, Nantes, France

Recurrent Neural Networks on Drifting Time-of-Flight Measurements

Tobias Feigl*†, Thorsten Nowak‡, Michael Philippsen*, Thorsten Edelhäußer†, Christopher Mutschler†§
{ tobias.feigl | thorsten.nowak | michael.philippsen } @ fau.de
{ thorsten.edelhaeusser | christopher.mutschler } @iis.fraunhofer.de

*Programming Systems Group (Informatics II)
‡Institute of Information Technology (Comm. Electronics)
§Machine Learning and Data Analytics Lab
Friedrich-Alexander University Erlangen-Nürnberg
Erlangen, Germany

†Machine Learning and Information Fusion Group
Precise Positioning and Analytics Department
Fraunhofer Institute for Integrated Circuits IIS
Nürnberg, Germany

***Abstract*—Kalman filters (KFs) are popular methods to estimate position information from a set of time-of-flight (ToF) values in radio frequency (RF)-based locating systems. Such filters are proven to be optimal under zero-mean Gaussian error distributions. In presence of multipath propagation ToF measurement errors drift due to small-scale motion. This results in changing phases of the multipath components (MPCs) which cause a drift on the ToF measurements. Thus, on a short-term scale the ToF measurements have a non-constant bias that changes while moving. KFs cannot distinguish between the drifting measurement errors and the true motion of the tracked object. Hence, very rigid motion models have to be used for the KF which commonly causes the filters to diverge. Therefore, the KF cannot resolve the short-term errors of consecutive measurements and the long-term motion of the tracked object.**

This paper presents a data-driven approach that uses training sequences to derive a near-optimal position estimator. A Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) learns to interpret drifting errors in ToF measurements of a tracked dynamic object directly from raw ToF data. Our evaluation shows that our approach outperforms state-of-the-art KFs on both synthetically generated and real-world dynamic motion trajectories that include drifting ToF measurement errors.

I. Introduction

Radio-based localization is a key component of many indoor applications and industrial environments. Commonly, a set of synchronized antennas receives radio signal bursts from mobile tags to exploit time-of-flight (ToF) values, like time-of-arrival (ToA) and time-difference-of-arrival (TDoA), to estimate a position.

The multipath propagation of RF signals is the most prominent error source that adds a dynamic bias (drift) to the ToF measurements. In dynamic motion scenarios the phases of the multipath components (MPCs) change. This results in a ToF drift as consecutive ToF measurements are impaired by biased delay errors due to multipath. Thus, the longer one tracks a quasi-linear motion of an object the better one can describe this drifting behavior, for example by means of Bayesian filters.

In practice, objects move dynamically and non-linearly and change their motion behavior both over a short- and a long-term. Hence, the main challenge is to distinguish between the true motion and the drift of the ToF measurements. Unfortunately, Bayesian filters cannot correctly describe these overlapping object dynamics and multipath-induced motion (i.e., the drifting error), as they only consider the short-term context. In Markovian motion models [1] the current state only depends on its previous state, independent of anything that happened before. Bayesian filters cannot take long-term dependencies into account and thus provide a poor performance.

Their performance is also limited by the empirical knowledge that has to be provided. This includes a good estimate of the motion model, process and measurement noise. However, even if perfectly known, Bayesian filters only have a limited ability to describe complex motion. To overcome long-term dependent signal drift, a Kalman filter (KF) tunes its motion model to be rigid. Hence, it cannot react to rapid motion changes or it may even diverge. Instead, tuning the KF's Q- and R-parameters so that they fit to the measurements, lets the filter follow the short-term delay error drifts. Thus, by design a KF provides inaccurate and imprecise trajectory estimates.

Instead of using empirical knowledge, to specify the system and the motion of an object we use raw ToF data and highly precise ground truth positions to train a filter model that uses a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM). While simple neural networks only interpret snapshots of the measurement, an LSTM learns both short- and long-term dependencies from noisy radio signals. The time- and context-sensitive LSTM interprets time-dependent multipath scenarios, and thus drifting measurements, and can return an absolute 3-dimensional position from raw ToF data. We train our LSTM end-to-end with both synthetic trajectory data (that contains long-term signal drift) and with real-world ToA measurements to fit sub-millimeter precise baseline positions. For the evaluation we compare our estimates with those from a KF.

The paper is structured as follows. Sec. II reviews related work before we cover the foundation in Sec. III. Afterwards we present the details of our LSTM model in Sec. IV. Sec. V lines out the experimental setup and Sec. VI evaluates the results. Sec. VII concludes.

II. Related Work

Radio-frequency-based positioning systems suffer from signal propagation effects such as attenuation, scattering, diffrac-

978-1-5386-5635-8/18/$31.00 ©2018 IEEE

tion, and reflection, which lead to a geometrical dilution of precision and to multipath effects [2], [3].

Naive standalone Bayesian Filters such as KFs and Particle Filters (PF) use a (linear) motion model and transition state. They (re-)calibrate and smooth the real physical position by continuously estimating measurement and process covariance matrices. To make this work in practice, this needs a lot of expertise and manual fine-tuning of filter parameters [4]. As the sensor characteristics (and the noise) change with time, an a-priori parameterized filter cannot correctly represent the real-world state and must return wrong position estimates. But what is more important, the covariance matrices only represent a snapshot of the sensor and motion characteristics and cannot describe a long-term dependency of a moving object. Even popular enhancements such as Extended and Unscented KFs cannot correctly describe highly non-linear motion models as they would need to define an infinite number of linear motion constraints [5]. While PFs can correctly approximate the position if there are enough particles, they also perform poorly in highly dynamic situations or if the tracking is lost [6].

Optimization techniques like Bancroft, Gradient Descent, and Levenberg Marquardt (LM), solve the non-convex optimization problem and estimate a position that minimizes the error term [7]. But they do not consider consecutive measurements that span up trajectories over time. This limits accuracy. Dong et al. [8] show that consecutive estimates from LM only represent a statistical optimum under laboratory conditions and hence are not applicable for real-world scenarios.

Decision tree, k-Nearest Neighbor, Support Vector Machine, Neural Network, and other classification methods estimate accurate positions. While they are popular for signal strength data such as RSS [9] there is also work that builds on ToF data [10]–[14]. By design neural networks automatically extract relevant features that describe the absolute position. They can also learn the effects of multipath propagation in static scenarios, e.g., when some radio signals are reflected by objects such as walls [15]. However, all these models assume static environments, thus they cannot compensate for dynamic changes of the environment, and cannot handle dynamic situations that affect the ToF fingerprints. In dynamic situations, track-ables move around and can create unknown (unseen or unlearned) reflections that lead to wrong position estimates.

Hybrid schemes combine the coarse position estimate of a neural network with a post-processing in a KF [16], LM [7], or Hidden Markov Model (HMM) [17]. However, both pre- and post-processing suffer from complex parameterization needs as well as from linear motion models that only respect short-time causations (time between two filter updates) and thus are not aware of long-term signal drift. Al-Mayyahi et al. [18] apply LM to a neural network on simulated trajectories and show that their simulation successfully optimizes vehicle control parameters and reduces the absolute position error. While their approach works in a lab, they do not account for motion.

The movement context is important as it describes the relationship between consecutive positions over a (very) long time. LSTMs [19] can model such relationships [20]–[22]. Yousefi et al. [23] investigate the performance of RNNs and LSTMs on RSS fingerprints. They combine the coarse position estimation from an LSTM with the fine positioning from an HMM. While their accuracy outperforms previous approaches, their HMM is constrained in its motion model (due to its a-priori design). RNNs in combination with RF-based features are investigated in [24] but only to determine the signal-to-noise ratio of the tracking system. Thus, while it was not our idea to use LSTMs to identify, learn, and reconstruct time dependencies of information, we are the first to investigate LSTMs on ToF from radio signals.

Hybrid KFs [4], [16], [25], [26] address the problem of insufficient estimation of the process and measurement noise with knowledge of the underlying sensor characteristics. Odelson et al. [27] can estimate the covariance matrices with auto-covariance least-squares methods, but they need a large sample size to converge and they only either estimate optimal measurement and process noise covariances [5] or state transition functions [27], but not both. Unfortunately, the position estimate of a KF depends on both. Coskun et al. [4] learn these parameters from data samples in image-based pose estimation. They use three stacked LSTMs layers to learn the non-linear transition function. Two additional single LSTM layers learn the process and measurement covariance matrices. This approach outperforms all previous image-based positioning methods. But although conceptually, their approach could also be adapted to ToF-based positioning, learning every (comparably) short-term motion type would need lots of computation-intensive image data. Since they only use RGB data, their approach does not work on small and limited information sequences. On large sequences it only provides a short fraction of repetitive movement types. Although Perez-Ortiz et al. [28] and Iter et al. [22] argue that their KF-LSTM better generalizes than standalone LSTMs (the implicit fusion of the past motion states and the measurement update is said to be too complex to be learned by a standalone LSTM), our LSTM does learn contextual motion information to optimally model long-term signal drift. As we derive accurate and precise positions our motion model implicitly proves to be adequately modeled.

To create and design physically correct models of radio channels we need to break down the radio-based communication pipeline into smaller segments. We show that we can learn to cope with signal drift and outperform state-of-the-art KFs. We use ideas from previous research and apply them to a baseline problem: phase shift that leads to signal drift in the long-term. Instead of applying end-to-end learning to the complete pipeline we understand all pieces that are necessary to describe the communication channel model.

III. Foundations

In this paper we train a model end-to-end to estimate a transmitters trajectory from a series of ToF values. We formally describe the problem of trajectory estimation in Sec. III-A and drifting ToF errors in Sec. III-B. We also briefly introduce LSTMs in Sec. III-C.

A. Problem Description and Terminology

Consider a transmitter's trajectory $t = [p_0, p_1, \cdots, p_i, ...]$, with i baseline positions p_i that represent the ground truth of the trajectory in Euclidean space ($p_i \in \mathbb{R}^3$). i denotes

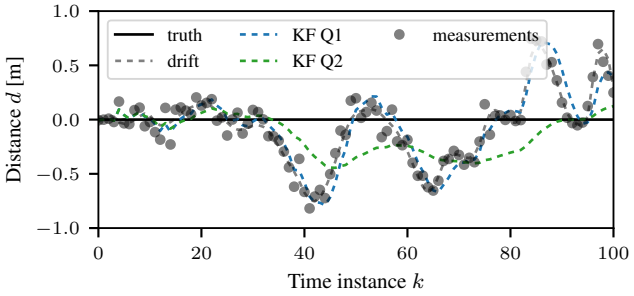


Fig. 1. KFs with different Q-matrices applied to a drifting error. Agile filter settings cause the KF to follow the drift (Q1). Rigid settings result in biased estimates (Q2). More rigid settings end up in filter divergence.

the time step of a position. Baseline positions are obtained from highly accurate reference systems. Each position p_i has a corresponding set of n ToF values $d_i = [d_{i0}, d_{i1}, \cdots, d_{in}]$, with $d_{ij} \in \mathbb{N}$ that represent the position p_i and that come from n receivers with low accuracy for time step i. Hence, for each time step i there is a baseline position p_i and a set of ToF values d_i. The notation $p_{i-s:i}$ stands for the slice $[p_{i-s}, p_{i-s+1}, \cdots, p_{i-1}, p_i]$ of the trajectory t, analogously for $d_{i-s:i}$, that spans a slice of time steps $i-s$ to i, of the full trajectory t.

Our goal is to fit a model M for position estimation $\hat{p_i} = M(d_{i-s:i})$ at time step i that works on s most recently observed sets of ToF values provided by the n receiver units. For the training of M we also use the exact ground truth positions $p_{i-s:i}$ to label the corresponding set of ToF values.

B. Drifting ToF Errors

In theory, trilateration can determine the correct position p_i from a set of exact ToF measurements d_i=$\{d_{i0}, d_{i1}, d_{i2}\}$. In practice, ToF data is inexact, their radii do not meet in a single point, and the estimated position $\hat{p_i}$ diverges significantly from the true position. The problem is that the locating system induces errors and artifacts to the signal processing pipeline. One such artifact in every radio-based signal processing system is the multipath propagation and signal drift.

Commonly, the multipath channel is characterized by a discrete number of multipath components (MPC). In general, the impulse response h of a wireless channel can be described as a tapped delay line with L MPCs $h = \sum_{l=1}^{L} a_l \cdot \delta(t - \tau_l)$ with a_l being the complex coefficient of the l-th MPC and τ_l the delay of the MPCs resulting from scattering objects [29].

Recent statistical channel models [30] for a scattering environment are commonly characterized by large-scale parameters (LSPs) like initial delays, complex coefficients, and arrival angles for the individual MPCs. In order to ensure spatial consistency, these values have to be updated along the track of the moving object. Otherwise, if one draws a new channel realization for every position on the track, the estimates of the channel coefficients do not fit the real ones. To avoid such effects, a trajectory is divided into segments (of up to several meters). Within a segment the channel is assumed to be wide sense stationary (WSS), i.e., the LSPs do not change significantly and leave the scattering environment unchanged. However, due to small-scale motion the path delays change. This causes a change in the complex channel coefficients. Changes in the phase of the MPCs inherently lead to drifting ToF errors [30].

Moving along a segment causes ToF estimates to drift. This phenomenon is commonly observed in ToF-based systems [31]. In consequence, we cannot describe the ToF errors by a zero-mean probability density function (PDF). On short-term scale, the ToF errors are biased and drift, i.e., ToF errors follow a random walk-like motion which is overlaid with the true motion of the transmitter. The challenge is to separate these motions. Fig. 1 illustrates the application of KFs with different parameters on measurements with drifting ToF errors.

C. Long-Short Term Memory (LSTM) Networks

Neural networks such as MultiLayer Perceptrons (MLPs) are composed of an input layer X, one or more hidden layers H, and an output layer Y. Input signals propagate in forward direction through the network: a loss-function compares the output to the ground truth label to compute the error; error signals propagate backwards through the network; adjusted weights reduce the error. Positioning applications often use variants such as Radial Basis Function Networks (RBFN) [32]. However, all MLP-variants can only take a fixed input sequence length of fingerprints of ToF values d_i at time step i to predict $\hat{p_i}$ and lack the ability to describe the relationship of time- and context-sensitive consecutive position(s) of a trajectory, as they miss the recurrent feature (i.e., ability to consider input and previous state). There is no way to include latent motion characteristics that encode a sequence of measurements.

Motion is a temporal-dynamic process that depends on different factors like acceleration, velocity, and direction which we can only observe indirectly from the sets of ToF values d_i and their corresponding positions p_i. Moreover, these values are also noisy and have drifting signal artifacts. Therefore, the model M that we aim to fit has to capture the latent dynamics of these factors from the stream of ToF and position data.

While we could design an MLP that uses a stack of past measurements to predict $\hat{p_i}$, Recurrent Neural Networks (RNNs) are a better choice [33]. RNNs are a class of neural networks that process sequences of data (here: consecutive sets of ToF measurements) and have successfully been applied to various time- and context-sensitive tasks. RNNs have persistent internal states c_i (the cell's memory state) that get updated from the input sequence X_i and that are carried across transitions, e.g., from H_i to H_{i+1}. That enables the model to capture dynamic behavior from the input data, and to use it to predict the output Y_i. The following equations describe the updates of a RNNs cell state whenever new input data X_i arrives:

$$H_i = g(W_h \cdot H_{i-1} + W_{in} \cdot X_i + B_h),$$
$$Y_i = a(W_o \cdot H_i + B_o).$$

Here, $g(\cdot)$ and $a(\cdot)$ are (non-linear) activation functions, $W\_$ are shared weight matrices, and $B\_$ are bias terms. But since RNNs suffer from vanishing or exploding gradients [34] in practice, they hardly recognize long-range dependencies.

LSTM networks [19] avoid this problem as they replace the computation of H_i with a cell that includes three *gates*: a

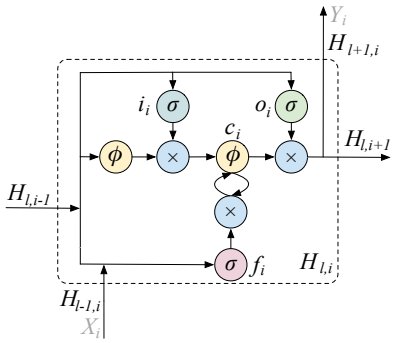


Fig. 2. Basic LSTM memory cell.

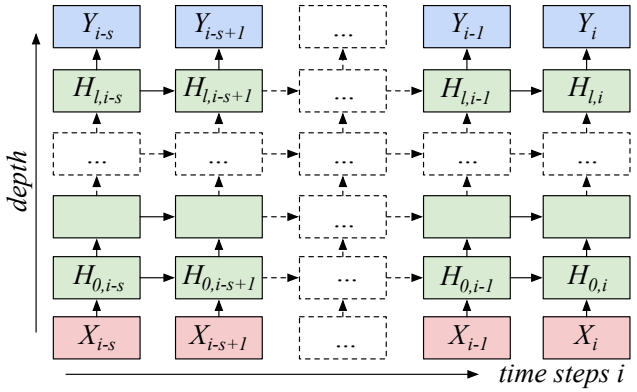


Fig. 3. Proposed unrolled many-to-many LSTM model architecture with l hidden LSTM layers H for a slice of s that covers i time steps. The inputs X feed the hidden layers with sets of ToF measurements. The predicted positions are in Y. (Fully connected layers and dropout layers are not outlined explicitly).

forget gate f_i, an input gate i_i, and an output gate o_i at time step i. These cells explicitly model the flow of information between consecutive time steps. Fig. 2 depicts the internal structure of a basic LSTM cell. Here σ represents the sigmoid gate activation function (forget f_i, input i_i, output o_i), ϕ is the $\tanh$ input/output activation function, and $\times$ is a multiplication operator. The cell's memory state c_i gets updated in every step with information from both the forget and the input gate. H_i is the new state, extracted through the output gate from the memory cell c_i.

Information flows through the LSTM from left to right and from bottom to top. Initial input X_i goes into the lowest LSTM layer l=0 ($H_{l=0,i}$). Each cell $H_{l,i}$ feeds also the adjacent cell on the same layer $H_{l,i+1}$ from the side and the adjacent cell on the next higher layer $H_{l+1,i}$ from the bottom.

During the training phase, the model M learns to leverage latent motion characteristics and trajectories from a consecutive sequence of positions p_i and their corresponding sets of ToF values d_i to provide an accurate and precise estimation of a transmitter's current position and trajectory on sets of raw ToF measurements during the live phase.

The following equations define the forward-pass of an LSTM cell for the hidden state H_i at step i. $U\_$ are more shared weight matrices and $\times$ is the element-wise multiplication operator from Fig. 2.

$$
\begin{aligned}
i_i &= \sigma\left(W_i X_i + U_i H_{i-1} + B_i\right) \\
o_i &= \sigma\left(W_o X_i + U_o H_{i-1} + B_o\right) \\
f_i &= \sigma\left(W_f X_i + U_f H_{i-1} + B_f\right) \\
c_i &= f_i \times c_{i-1} + i_i \times \tanh\left(W_c X_i + U_c H_{i-1} + B_c\right) \\
H_i &= o_i \times \tanh\left(c_i\right)
\end{aligned}
$$

We define the width of a model by the size of an LSTM cell. The size of a single cell is defined by the number of possible weights and biases that each cell can handle. In Fig. 2 every blob (function, operator, gate, or cell) represents a fixed denoted functionality but also embodies a stack of weights and biases according to the cell size, i.e., a cell size of 128 denotes that each blob in the cell holds 128 weights and 128 biases.

IV. Proposed LSTM Model

Fig. 3 shows our proposed LSTM model. For simplification for every time step i (column) we left out both the drop-out layers between each of the hidden layers H_0 to H_l and also left the dense, fully-connected classification layer just before the output layer Y.

During the training phase, the model takes slices of consecutive trajectory positions $p_{i-s:i}$ and their corresponding sets of ToF values $d_{i-s:i}$ as input. The sequence length s defines how many of such consecutive bundles the model processes in each iteration. The longer the sequence the more motion dependencies the model learns to interpret correctly but the more complex the model gets. The model not only learns how to obtain a single position from a single set of ToF values but also learns to predict consecutive positions from sets of consecutive ToF values. While X_i represents an input set at times step i, the resulting position estimation is Y_i. With a randomly initialized initial state H_0, the model has no prior knowledge about the motion and consequently learns to estimate the position from local information. As output Y the model estimates the transmitter's current positions $\hat{p}_i$.

Data Preprocessing. Machine learning models suffer from data with high variance and deviation. A numerical tiny (local) gradient or weight explodes if a huge value is fed to the neuron. On the other hand, a large weight vanishes with very small update values. As such models perform best when the input data has zero *mean* and unit *variance* we normalize the dataset to $[-1;+1]$ by subtracting the *mean* and scaling with the *standard deviation* over the training sets for each individual feature (ToF value) [35]. Once trained, the model can be applied to unseen ToF data.

For the training of M, we slice the complete trajectory into consecutive sequences of length s and create sub-trajectories starting from random offsets in the trajectory (like a sliding window). (Note, in the live phase the model predicts consecutive positions $p_{i-s:i}$ only on consecutive sets of ToF measurements $d_{i-s:i}$).

Data Batching. Fig. 4 provides a detailed insight of our input data batching method. In total, we have batches (independent groups of data) G for each time step in the sequence. A batch consists of j groups $g_{j,i}$ at each time step from $i-s$ to i. Each group embodies the position p_i (only during the training phase) and the corresponding ToF values d_i (during the training and live phases). Each group g in a batch G

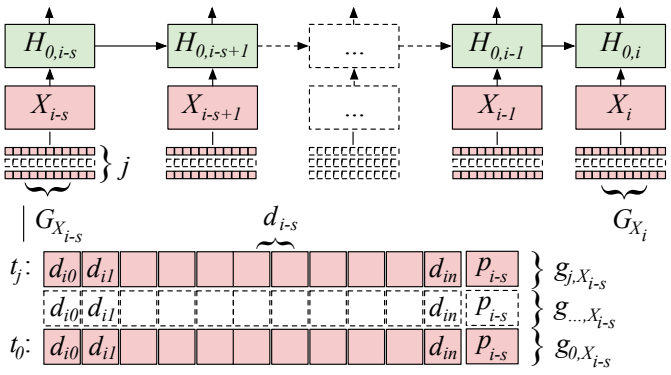


Fig. 4. Data preparation process that prepares batches G of ToF values d at each time step i and feeds them as input X_i to the first hidden layer H_0 of our LSTM network.

represents a position of a different training trajectory t_j, i.e., our batch contains data at time step i from all our recorded training trajectories. (Note during the live phase we only feed $G_{X_{i-s:i}}$ each with a single $g_{j=0,Xi-s:i}$ to M). The three rows of small boxes in Fig. 4 show a simple example of how our batches G look like for each single time step from $i-s$ to i.

We split a sequence in $i-s:i$ batches G_{X_i} that hold j groups $g_{j,X_{i-s}}$ (during training j denotes the number of different training trajectories; during the live phase j is 1). Each group $g_{i,X}$ (group g at time step i with input X; X contains either p_i and d_i during the training, or only d_i during the live phase) holds a set of ToF values d_i that describe a position p_i from a trajectory t_j. The number of different training trajectories j defines the size of the batch G, with $G_{X_{i-s}}$=$\{g_{0,X_{i-s}}, g_{1,X_{i-s}}, ..., g_{j,X_{i-s}}\}$. The sequence length s defines the number of time steps that our model takes into account to predict new positions (i.e., how many ToF and position sets are fed to the network) per iteration. Thus, each sequence holds s batches G and each batch G holds j groups g of n ToF values d. Each g_j in G represents an independent part of the motion trajectory so that the model learns different time-based motion relations. Such time-dependent relations include changes in e.g. acceleration, velocity, displacement, trajectory, and other motion types. An increase in depth, i.e., the number of layers, yields an increase of variety of different time-based

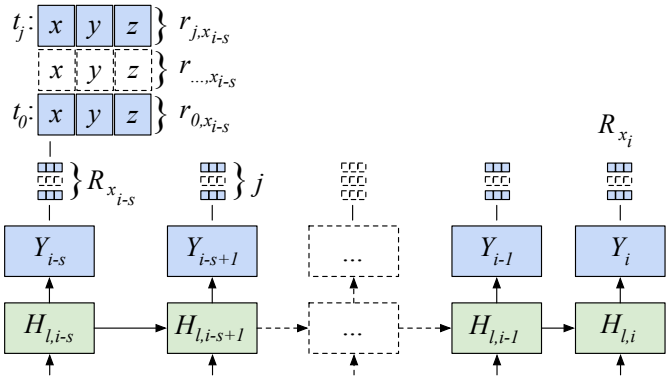


Fig. 5. Fully connected layer that condensates the output of the last hidden layer H_l of our LSTM model to Y and returns 3-dimensional absolute positions $p_{i-s:i}$.

characteristics, i.e., the deeper the network the more time-dependent relations can be modeled by the network [36].

Feed. We feed X_i (training: sets of ToF values and their corresponding reference positions; live: sets of ToF values) sequentially to the LSTM-cells, see Fig. 4. E.g., with a sequence length of s = 200 we feed 200 LSTM-cells of the first hidden layer $H_{0,i-s:i}$ with inputs $X_{i-s:i}$. With each step i, the state $H_{i-s:i}$ is updated and accumulates information about the changes in the positions. We use a larger cell size (e.g. 256 neurons), to handle more complex problems [37]. After the complete sequence has been processed the final LSTM state H_i contains a representation of the observed motion in LSTM state space.

Dense. In a final step, the model uses a fully connected layer to dense the final states $H_{i-s:i}$ to the output $Y_{i-s:i}$ that holds the transmitter's positions $\hat{p}_{i-s:i}$, see Fig. 5. According to the number j of blocks g in the batch G the dense layer returns the same number j of results r (here positions with x,y,z coordinates) in the resulting batch R, with $R_{X_{i-s}} = \{r_{0,X_{i-s}}, r_{1,X_{i-s}}, ..., r_{j,X_{i-s}}\}$. We take the latest element $r_{j,X_{i-s}}$ as a position estimate for $\hat{p}_i$.

V. Setup of the Experiments

Our experiments compare a KF model with our LSTM approach on both synthetic (Sec. V-A) and real-world (Sec. V-B) datasets with drifting ToF measurements. The datasets comprise multiple trajectories t, each of which is represented by a stream of ToF measurements d from n receivers and the corresponding positions p. We split both datasets into a training set (90%) and a test set (10%). Sec. V-C specifies the configuration of our KF and LSTM models.

A. Simulation Environment

To adequately model the signal propagation effects of the wireless channel we used QuaDRiGa [30] as it offers characteristics that apply for typical indoor propagation scenarios, i.e., multipath effects. We briefly outline the procedure of channel simulation here.

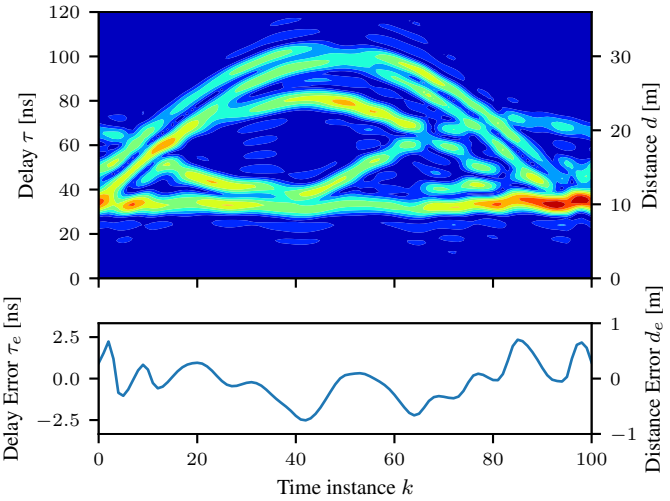


Fig. 6. Drifting in a multipath environment. The channel impulse response is evolving over time. Gradual change in delay and phases leads to biased ToF estimates. ToF errors in short-term context are biased. Hence, ToF errors are not to be characterized by a zero-mean PDF as demanded by KFs.

From LSPs with L dominant paths, the delay spread, and the angular spread, we construct a bandwidth-limited channel impulse response (CIR) to estimate the ToF values. To do this we (1) draw delays τ_l, (2) compute their powers P_l, (3) draw arrival angles ϕ_l, and (4) combine these values to channel coefficients a_l. The most essential channel parameters used in our simulations are: number of MPCs L = 6, RMS delay spread σ_τ = 10 *ns*, and the Rician K-factor K = 0.5. For ToF estimation we used a bandwidth of B = 80 MHz which is close to our real-world setup [38].

Fig. 6 depicts how a typical indoor channel evolves over time, see also Fig. 1, when a mobile tag moves along a circle with a radius of 10 *m* (line-of-sight) and a receiver placed in the center of the circle. While there are parts of the signal that arrive with 30 *ns* delay there is also a significant part that has a higher delay. This delay also changes while the transmitter moves through the multipath setting. In the lower part of Fig. 6 we see the resulting ToF errors. Obviously, drifting causes a slowly changing bias to the ToF values. (Fig. 6 shows a random walk-like motion).

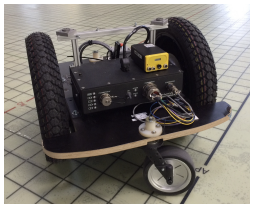
Our simulation framework generates the synthetic datasets D_{sd} and D_{rd} with a noisy Gaussian distribution (*mean* μ = 0.0 and a *standard deviation* σ = 0.1). Our dataset D_{sd} lets a transmitter follow along a sine wave walk ($A \cdot \sin(2\pi\frac{n}{w} + \Phi) + D$, with A = 1, a data length of n = 10,000 samples, a wave length of w = 1,000 samples, Φ = 0, and D = 3), see 'Truth' in Fig. 8(a), and generate a stream of ToF values as the transmitter moves along the sinusoid with an (ideal) static ToF of 3 *ns*, i.e., 1 *m* distance. Instead, D_{rd} lets the transmitter follow along a random walk (cumulative sum of n = 10,000 randomly variates with scale σ = 0.1, that we limit to $[2; 4]$ *m*). Besides a Gaussian noise we add a long-term signal drift, see 'Meas. Drift' in Fig. 8(a), in the form of D_{sd} a sinusoidal (same parameters, but A = 2) or D_{rd} a random walk (same parameters, but limit $[1; 5]$ *m*), see 'Meas. Drift' in Fig. 8(b).

We generated 100 synthetic trajectories per drifting parameter: with sinusoidal drift (D_{sd}), with random walk drift (D_{rd}). For each of the configurations we generate 100 trajectories with 10,000 positions each. These input data contains 1-dimensional ToF measurements and their corresponding position, i.e., the transmitter's distance to the antenna over time.

B. Real-World Environment

Fig. 7(a) shows our real-world indoor tracking area. We capture ToF (ToA) data with a remotely controlled Segway RMP-210 mobile robot, see Fig. 7(b). For 220 times the robot follows a predefined zig-zag trajectory (see Fig. 8(c)) at a maximal speed of 12 *km/h* and a maximal acceleration of 2 *m/s*^{2}. The resulting dataset D_{rw} holds about 504 minutes of motion data, consisting of 220 trajectories with 1375 ToA values each taken from a radio-based locating system RedFIR [38], [39] with 12 time-synchronized antenna units at fixed locations. The locating system operates in the 2.4 *GHz* ISM band and uses around 80 *MHz* bandwidth. The miniaturized transmitters generate short broadband signal bursts at 10 positions per second. We also record the ground truth trajectory with a sub-millimeter precise optical laser-based Nikon iGPS system. In a data pre-processing step we linearly interpolate intermediate positions (10 *Hz* RedFIR vs. 30 *Hz* iGPS) where necessary.

(a) Top-view onto our 40 *m* × 40 *m* indoor RF-tracking area.



(b) Autonomous wheeled Segway mobile robot.

Fig. 7. Setup of our experiment.

C. Model Configuration and Training

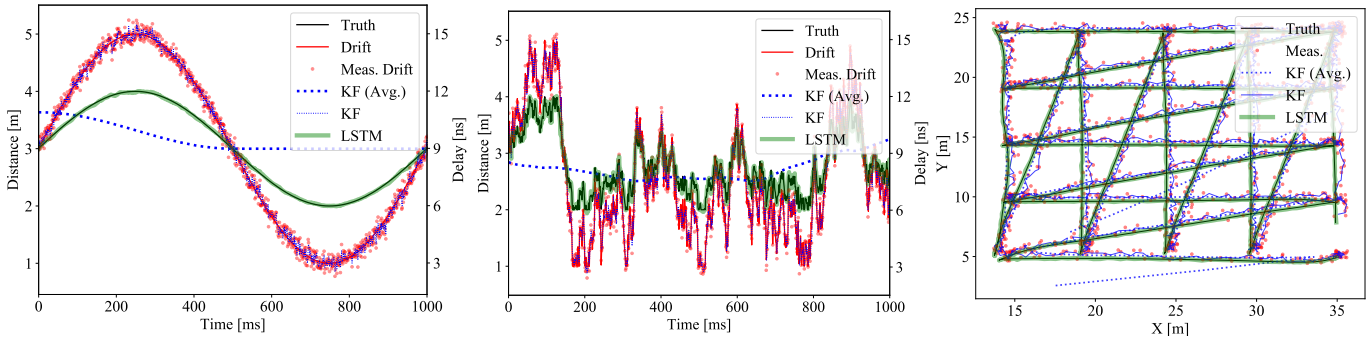
Kalman filter model. We use a classic linear KF model as an optimal state estimator under the assumption of drift-free signals D_{wo}. This Bayesian model describes the latent state evolution, i.e., linear motion transition function $l(t)$, the emission distribution, i.e., measurement noise, and internal action effects, i.e., a process noise, as linear functions that are perturbed by Gaussian noise. We initially parameterize the KF with a start state x_0 = 0, a covariance P = 1, process noise Q = 0.1, measurement noise $R = \sigma$ = 0.1, and a transition function $l(t)$ that models a constant velocity. By clearance if we have empirical knowledge we can configure an optimal KF (with x_0 = 0 and R=σ = 0.1) that perfectly fits the input data and then this model is a perfect optimizer for the drift-free dataset D_{wo}.

To evaluate the KF model on the drift-free datasets we sequentially apply a set of ToF/ToA data to the model to estimate a delay (for the synthetic data) or an absolute position (for the real world data). We then compare this position with the baseline position that represents the ToF/ToA inputs and derive the CEP_{50} and CEP_{95} metric errors, i.e., the median error and the 95% percentile in the horizontal plane.

LSTM model. As our LSTM model has non-linear activation functions by clearance this model may be a candidate that perfectly models all of our datasets, i.e., D_{sd}, D_{rd}, D_{wo}. With a grid search [40] we find a proper model architecture, i.e., we train different architectures and vary the number of layers, the width, the batch size, sequence length, i.e., number of time steps, optimizers, amount of drop-out per layer, and the effect of peepholes. See Table. I for details and the searched configuration that is a good trade-off between computational effort and performance: the batch size is as large as the number of various trajectories $=$100, epochs $\geq$1000, layers

TABLE I
RESULTS OF OUR GRID SEARCH FOR LSTM PARAMETERS.

Test Types	Settings	Selection
Batch size	[1, 2, 5, 10, 20, 50, 100, 200]	100
Epoches	[10, 50, 100, 1000, 2000]	1000
Depth/Layers	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20]	4
Sequence length	[1, 2, 5, 10, 20, 50, 100, 200]	20
Optimizer	[SGD, RMSprop, Adagrad, Adam]	Adam
Drop-out	[0.0, 0.0001, 0.001, 0.01, 0.1]	0.0
Width/Cell size	[1, 8, 16, 32, 64, 128, 256, 512]	256
Peepholes	[On, Off]	Off
Learning rate	[0.00001, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]	0.0001
Activation functions	[linear, softmax, tanh, relu, sigmoid]	tanh
Weight Initialization	[uniform, normal, zero]	normal



(a) Synthetic sinusoidal drifting dataset D_{sd}. (b) Synthetic random-walk dataset D_{rd}. (c) Real-world dataset D_{rw}.

Fig. 8. ToF distance and delay errors of our datasets (black: ground truth, red: measurements with drift, green: our LSTM model, blue: our KF).

$\geq$4, sequence length $\geq$20, optimizer is Adam, no drop-out, width $\geq$256, no peepholes, learning rate $\leq$0.001, sigmoid and tanh as activation and uniform weights.

We train our LSTM model on samples generated from our trajectories using a quasi-sliding window approach over the input dataset. For every input step X_i we create a sample dataset by extracting a slice of size s from the trajectory. The slice is split into batches of independent trajectory groups. Hence, every batch G and their respective groups g_j represent a set of consecutive ToF values and their corresponding position. Note, every sequence that is extracted from a trajectory starts at varying start offsets. Hence, we guarantee that our model learns completely independent trajectories. To update the weights of our LSTM we calculate the loss, i.e., the convergence error between the predicted position on the ToF values and the baseline position. We use the mean-squared error (MSE) loss function $L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$.

VI. Results

To compare the results of our KF and LSTM models, on both the drift-affected synthetic D_{sd}, D_{rd} and real-world D_{rw} datasets we calculate error metrics against the ground truth. We use the mean average error (MAE) of the estimated delay for the synthetic data and the $\text{CEP}_{50,95}$, i.e., the median positional error and its 95th percentile on the horizontal plane, of the 2-dimensional real-world dataset.

A. Synthetic Datasets

Both of our drifting datasets are 1-dimensional and thus the MAE metrics only show the accuracy of the models in 1-dimension (position translates to distance over time).

Sinusoidal Drift. Fig. 8(a) shows 1000 *ms* of the results for the synthetic dataset with the sinusoidal drift D_{sd}. The black line shows the ground truth (sinusoid with zero delay errors). The red dots are the drifting ToF measurements. They follow long-term (sinusoidal) drift, i.e., they suffer from the sinusoidal drift that comes on top of the sinusoidal signal, and are additionally impaired by additive white Gaussian noise. The blue dotted line(s) show(s) the estimations of the KF. While the KF smooths out the Gaussian noise it closely follows the drift, as this yields the optimal estimation state. Instead, KF (Avg.) follows its internal state, smooths over 50 samples, and performs even worse. KF results in an MAE of 2.13 *ns*, i.e., a distance measurement error of 64 *cm*. Since our LSTM fits the line of ground truth values, it not only clearly outperforms the state-of-the-art KF model but also follows the real ToF values very closely, which in total results in an MAE of only 0.029 *ns* or 0.009 *cm*.

Random Walk Drift. Fig. 8(b) shows the results for the synthetic random walk dataset with the random walk drift D_{rd}. Again, the KF follows the drift in the measurements while it smooths out the random noise or ignores the measurements completely (KF (Avg.)). The KF estimates have an MAE of 1.72 *ns* or 51 *cm*. Again our LSTM approximates the real ToF values closely and yields a much smaller MAE of 0.12 *ns* or 3.6 *cm*.

The results on the synthetic datasets show that in both scenarios our LSTM clearly outperforms the KF and yields close-to-optimal estimations.

B. Real-World Dataset

We run both the KF and the LSTM on our real-world dataset D_{rw}. On the position estimates, i.e., 'Meas.' in Fig. 8(c), we calculate the CEP_{50} and the CEP_{95} positional error of both models in the 2-dimensional horizontal plane. We used the Bancroft algorithm [41] to estimate a position from a set of corresponding ToFs. Fig. 8(c) shows the results.

Overall, the LSTM (with a CEP_{50} of 4.08 *cm* and a CEP_{95} of 12.93 *cm*) approximates the real position considerably better than the KF (with a CEP_{50} of 30.56 *cm* and a CEP_{95} of 35.34 *cm*). Again, the KF follows the drifting ToF values as it wriggles around the baseline trajectory. Moreover, the KF also under-/overshoots whenever the movement changes in the corners. Here the KFs takes a long time to recalibrate. In contrast, our LSTM not only closely approximates the real trajectory on the straight lines but also performs well in the corners.

C. Discussion

It is likely that our LSTM model can handle time-dependencies because of depth (number of hidden layers) and the size of the slice of time steps that it takes into account. The deeper a network the more time-dependencies, and thus motion derivatives such as acceleration, velocity, displacement, orientation, and intention it can handle. Throughout our experiments the more shallow models do not learn and generalize motion behavior as well. Instead they mostly map the inputs to

the outputs. We successfully applied and validated our method on rectangular, sinusoidal, elliptical, and arbitrary trajectories.

VII. Conclusion

The presented RNN with a stacked LSTM model can (after being trained) predict optimal consecutive positions from raw and drift-affected ToF measurements provided by a low accuracy locating systems.

While we show that our model learns to cope with time dependent errors such as multipath and drift, we are the first to also present novel steps towards understanding, interpreting, and context learning in a single position estimation model. The presented LSTM outperforms state-of-the-art KFs on both synthetically generated and real-world trajectories that include drifting ToF measurement errors.

Acknowledgments

This work was supported by the Bavarian Ministry for Economic Affairs, Infrastructure, Transport and Technology and the Embedded Systems Initiative (ESI).

References

[1] S. Särkkä, *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

[2] D. I. Tapia, R. S. Alonso, S. Rodriguez, F. de la Prieta, J. M. Corchado, and J. Bajo, "Implementing a real-time locating system based on wireless sensor networks and artificial neural networks to mitigate the multipath effect," in *Proc. Intl. Conf. Information Fusion*, (Chicago, IL), pp. 1–8, 2011.

[3] P. Torteeka, X. Chundi, and Y. Dongkai, "Hybrid technique for indoor positioning system based on wi-fi received signal strength indication," in *Proc. Intl. Conf. Indoor Positioning and Indoor Navigation*, (Busan, Korea), pp. 48–57, 2014.

[4] H. Coskun, F. Achilles, R. DiPietro, N. Navab, and F. Tombari, "Long short-term memory kalman filters: Recurrent neural estimators for pose regularization," *arXiv preprint arXiv:1708.01885*, 2017.

[5] "A generalized autocovariance least-squares method for kalman filter tuning," *J. Process Control*, vol. 18, no. 7, pp. 769 – 779, 2008.

[6] S. Särkkä, A. Vehtari, and J. Lampinen, "Rao-blackwellized particle filter for multiple target tracking," *Information Fusion*, vol. 8, no. 1, pp. 2–15, 2007.

[7] C.-S. Chen, "Artificial neural network for location estimation in wireless communication systems," *Sensors*, vol. 12, no. 3, pp. 2798–2817, 2012.

[8] L. Dong and F. L. Severance, "Position estimation with moving beacons in wireless sensor networks," in *Proc. Wireless Communications and Networking Conf.*, (Hong Kong, China), pp. 2317–2321, 2007.

[9] H. Dai, H.-B. Liu, X.-S. Xing, and Y. Jin, "Indoor positioning algorithm based on parallel multilayer neural network," in *Proc. Intl. Conf. Information System and Artificial Intelligence*, (Hong Kong, China), pp. 356–360, 2016.

[10] P. Singh and S. Agrawal, "Tdoa based node localization in wsn using neural networks," in *Proc. Intl. Conf. Communication Systems and Network Technologies*, (Bangalore, India), pp. 400–404, 2013.

[11] H. Zhu, B. Huang, Y. Tanabe, and T. Baba, "Local positioning with artificial neural network and time of arrival technique," (Dalian, China), pp. 509–509, 2008.

[12] H. Dai, H. Liu, X.-S. Xing, and Y. Jin, "Indoor positioning algorithm based on parallel multilayer neural network," (Hong Kong, China), pp. 356–360, 2016.

[13] D. I. Tapia, R. S. Alonso, S. Rodríguez, F. de la Prieta, J. M. Corchado, and J. Bajo, "Implementing a real-time locating system based on wireless sensor networks and artificial neural networks to mitigate the multipath effect," in *Proc. 14th Intl. Conf. Information Fusion*, (Chicago, IL), pp. 1–8, 2011.

[14] W. Zhang, K. Liu, W. Zhang, Y. Zhang, and J. Gu, "Deep neural networks for wireless localization in indoor and outdoor environments," *Neurocomputing*, vol. 194, pp. 279 – 287, 2016.

[15] A. Niitsoo, T. Edelhäußer, and C. Mutschler, "Convolutional neural networks for position estimation in tdoa-based locating systems," in *Proc. 9th Intl. Conf. Indoor Positioning and Indoor Navigation*, (Nantes, France), pp. 1–8, 2018.

[16] S. D.-C. Shashua and S. Mannor, "Deep robust kalman filter," *arXiv preprint arXiv:1703.02310*, 2017.

[17] W. Zhang, K. Liu, W. Zhang, Y. Zhang, and J. Gu, "Deep neural networks for wireless localization in indoor and outdoor environments," *Neurocomputing*, vol. 194, pp. 279–287, 2016.

[18] A. Al-Mayyahi, W. Wang, and P. Birch, "Levenberg-marquardt optimised neural networks for trajectory tracking of autonomous ground vehicles," *Intl. J. Mechatronics and Automation*, vol. 5, no. 2, pp. 140–153, 2015.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–80, 1997.

[20] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, (Las Vegas, NV), pp. 961–971, 2016.

[21] A. Milan, S. H. Rezatofighi, A. R. Dick, I. D. Reid, and K. Schindler, "Online multi-target tracking using recurrent neural networks.," *AAAI*, pp. 4225–4232, 2017.

[22] D. Iter, J. Kuck, P. Zhuang, and C. M. Learning, "Target tracking with kalman filtering, knn and lstms," 2016.

[23] S. Yousefi, H. Narui, S. Dayal, S. Ermon, and S. Valaee, "A survey on behavior recognition using wifi channel state information," *IEEE Communications Magazine*, vol. 55, no. 10, pp. 98–104, 2017.

[24] L. Bo, Q.-z. LIU, Z.-d. YIN, and Z.-l. WU, "A novel snr estimator for ds-uwb wireless sensor network," *Trans. Computer Science and Engineering*, no. cmee, 2017.

[25] M. Wei-Lung, "Gps interference mitigation using derivative-free kalman filter-based rnn," *Radioengineering*, vol. 25, no. 3, p. 519, 2016.

[26] R. G. Krishnan, U. Shalit, and D. Sontag, "Deep kalman filters," *arXiv preprint arXiv:1511.05121*, 2015.

[27] B. J. Odelson, M. R. Rajamani, and J. B. Rawlings, "A new autocovariance least-squares method for estimating noise covariances," *Automatica*, vol. 42, no. 2, pp. 303–308, 2006.

[28] J. A. Pérez-Ortiz, F. A. Gers, D. Eck, and J. Schmidhuber, "Kalman filters improve lstm network performance in problems unsolvable by traditional recurrent nets," *Neural Networks*, vol. 16, no. 2, pp. 241–250, 2003.

[29] J. Proakis, *Digital communications*. McGraw-Hill, Boston, 2008.

[30] S. Jaeckel, L. Raschkowski, K. Börner, and L. Thiele, "Quadriga: A 3-d multi-cell channel model with time evolution for enabling virtual field trials," *IEEE Trans. Antennas and Propagation*, vol. 62, no. 6, pp. 3242–3256, 2014.

[31] T. Nowak and A. Eidloth, "Dynamic multipath mitigation applying unscented kalman filters in local positioning systems," in *Proc. European Wireless Technology Conf.*, (Paris, France), pp. 9–12, 2010.

[32] P. Singh and S. Agrawal, "Tdoa based node localization in wsn using neural networks," in *Proc. Intl. Conf. Communication Systems and Network Technologies*, (Gwalior, India), pp. 400–404, 2013.

[33] J. Dai, P. Zhang, J. Mazumdar, R. G. Harley, and G. Venayagamoorthy, "A comparison of mlp, rnn and esn in determining harmonic contributions from nonlinear loads," in *Proc. Conf. Industrial Electronics Society*, (Orlando, FL), pp. 3025–3032, 2008.

[34] Y. Bengio, P. Y. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–66, 1994.

[35] K. Cho, B. van Merrienboer, aglar Gülehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proc. Conf. Empirical Methods in Natural Language Processing*, (Doha, Qatar), pp. 1724–1734, 2014.

[36] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in *Advances in neural information processing systems*, pp. 190–198, MIT Press, 2013.

[37] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[38] T. v. d. Grün, N. Franke, D. Wolf, N. Witt, and A. Eidloth, "A real-time tracking system for football match and training analysis," in *Microelectronic Systems*, pp. 199–212, Springer Berlin, 2011.

[39] C. Mutschler, H. Ziekow, and Z. Jerzak, "The debs 2013 grand challenge," *Proc. 7th ACM Intl. Conf. Distributed event-based systems*, pp. 54–57, 2013.

[40] N. Reimers and I. Gurevych, "Optimal hyperparameters for deep lstm-networks for sequence labeling tasks," *arXiv preprint arXiv:1707.06799*, 2017.

[41] S. Bancroft, "An algebraic solution of the gps equations," *IEEE Trans. Aerospace and Electronic Systems*, no. 1, pp. 56–59, 1985.