KalmanNet: Neural Network Aided Kalman Filtering for Non-Linear Dynamics with Partial Domain Knowledge

Guy Revach 1 Adrià López Escoriza 1,2 Nir Shlezinger 3 Ruud J. G. van Sloun 4,5 Yonina C. Eldar 6

Abstract

Real-time state estimation of dynamical systems is a fundamental problem in signal processing and control. For systems that are well-represented by a fully known linear Gaussian state-space (SS) model, the celebrated Kalman filter (KF) is a low complexity optimal solution. However, both linearity of the underlying SS model and accurate knowledge of it are often not encountered in practice. Here, we present KalmanNet, a real-time state estimator that learns from data to carry out Kalman filtering under non-linear dynamics and partial information. By incorporating the structural SS model with a dedicated recurrent neural network module in the flow of the KF, we retain data efficiency and interpretability of the classic algorithm while implicitly learning complex dynamics from data. KalmanNet overcomes non-linearities and model mismatch, outperforming classic filtering methods operating with both mismatched and accurate domain knowledge.

1. Introduction

In his pioneering work (Kalman, 1960), Rudolf Kalman introduced the Kalman filter (KF). The KF solves a fundamental problem in signal processing: real-time state estimation of a dynamical system in discrete-time. Its recursive low-complexity implementation combined with its sound theoretical basis have made the KF a widely used online filtering algorithm in numerous real world applications that involve tracking and localization (Auger et al., 2013). The KF is a model-based (MB) algorithm; its performance critically depends on the validity of a key assumption that the underlying state-space (SS) model is linear and accurately known. When the noise is also Gaussian, the KF is the minimum mean-squared error (MMSE) estimator. In practice, these key assumptions often do not hold; the underlying dynamics are complex and non-linear, while domain knowledge often relies on a crude approximation. There are well-known MB variants of the KF that are designed for non-linear dynamics, such as the extended Kalman filter (EKF) and the unscented Kalman filter (Julier & Uhlmann, 1997). Unfortunately, they are not theoretically optimal, and suffer from severe degradation in the face of strong non-linearity and model mismatch (Wan & Van Der Merwe, 2000).

An alternative strategy to implement state estimation without relying on accurate knowledge of the underlying complex and possibly non-linear SS model is to learn the filter mapping from data. In particular, neural network (NN) architectures such as recurrent neural networks (RNNs) and attention-based transformers have demonstrated remarkable success in data-driven (DD) time series prediction and denoising under complex setups. While such DD architectures can learn to capture complex dynamics, they lack the interpretability of MB methods and tend to require many trainable parameters even for seemingly simple sequence models (Zaheer et al., 2017). These constraints limit the application of highly parameterized deep models for real-time state estimation in applications embedded on hardware-limited mobile devices, such as drones and vehicular systems.

The limitations of MB Kalman filtering and DD state estimation motivate a hybrid approach, which combines the soundness and low complexity of the classic KF while exploiting the model-agnostic nature of NNs to mitigate its dependence on accurate knowledge of the SS. In this work we study a hybrid MB/DD implementation of the KF for state estimation in non-linear dynamics with partial domain knowledge. The state estimator referred to as KalmanNet implements online recursive filtering by integrating a compact RNN of limited complexity to replace the computation
of the Kalman filtering gain (KFG), which is an intermediate step in the KF flow. In particular, the KFG is the critical component encapsulating the dependence of the KF on domain knowledge and noise statistics.

This contributions of this work are as follows:

1) We design KalmanNet by incorporating domain knowledge and by taking advantage of theoretical principles of the KF algorithm. This yields an interpretable model that requires small amounts of data for training. KalmanNet circumvents the dependency of the KF on knowledge of the underlying SS model and the noise statistics, overcoming the need for a tailored solution for non-linear systems.

2) We show how KalmanNet learns to carry out Kalman filtering from data, thus sharing the real-time operation and reduced complexity of the MB KF.

3) We evaluate KalmanNet in non-linear SS models. The experimental scenarios include a synthetic setup, tracking the chaotic Lorenz system, and localization using the Michigan NCLT dataset (Carlevaris-Bianco et al., 2016). KalmanNet is shown to converge much faster compared to purely DD architectures, while outperforming the MB EKF when facing model mismatch and dominant non-linearities.

The rest of this paper is organized as follows: Section 2 reviews the related works on DD state estimation. Section 3 presents the system model and recalls the basics of Kalman filtering. Section 4 details KalmanNet. Section 5 presents the numerical study. Section 6 provides concluding remarks.

2. Related Work

State estimation in SS models is a fundamental problem of interest to various scientific communities. It encapsulates three distinct types of problems: smoothing, in which the state vector is estimated offline in a one shot manner (specifically, given a block of observations, one must estimate the states corresponding to the entire block); prediction, which operates in a similar fashion to predict future states based on a block of past observations; and filtering, which is the focus of this paper, where one must provide an instantaneous estimate based on each incoming observation in an online manner. Filtering is at the core of real-time tracking.

The KF and its variants, such as the EKF, have been extensively studied in the signal processing literature, and are widely used in a multitude of practical applications. We thus focus our survey of the related works on the usage of DD tools, and particularly the utilization of NNs, for state estimation in a hybrid manner with structural assumptions of SS models. We first discuss works that deal with offline state estimation, and are thus tailored towards smoothing and/or prediction tasks rather than the real-time filtering problem considered here. The work (Satorras et al., 2019) focused on the smoothing task, and proposed to apply a graph neural network in parallel to the Kalman smoother to improve its accuracy via neural augmentation. Since the estimation was performed by an iterative message passing over the entire time horizon, it cannot be adopted with low complexity to the filtering task. In (Rangapuram et al., 2018) a deep RNN combined with an SS model was used for univariate time series prediction, mainly in the context of business decision processes. During the training phase the underlying SS model parameters were directly estimated, limiting its scalability for high dimensional systems. In (Krishnan et al., 2015) a deep KF was proposed; however, the task is fundamentally different from state estimation. Specifically, it is a variational auto-encoder used for counterfactual estimation that involves capturing how actions affect observations. The work (Karl et al., 2016) introduced deep variational Bayes filters to enable unsupervised learning and system identification of latent Markovian SS models. Given this trained model it is then possible to perform long term predictions.

The task of symbol detection in digital communication corresponds to state estimation in discrete-valued SS models, where the state is a categorical variable. In the offline setting, NN-aided implementations of the BCJR decoder and the sum-product method were proposed in (Shlezinger et al., 2020b) and (Shlezinger et al., 2020c) respectively, while in (Shlezinger et al., 2020a) an online NN-aided implementation of the Viterbi decoder was proposed. The focus of our work is continuous-value SS models for which the KF and its variants are designed. The most closely related work for online state estimation is (Coskun et al., 2017), which used an RNN to learn each of the SS model parameters and then applied the EKF for state recovery in a pose estimation application. In (Laufer-Goldshtein et al., 2018) manifold learning was used to uncover the SS parameters, which were then plugged into the EKF for speaker tracking. In this work we circumvent the need to learn model parameters directly and learn them only implicitly via the KFG, thus bypassing the inherent approximations carried out by the EKF to handle non-linearities. In (Zheng et al., 2017), long short-term memory (LSTM) was used to model non-linear SS dynamics, combined with sequential Monte Carlo sampling for parameter estimation. It is noted that this method may be computationally prohibitive for most real-time applications.

3. System Model and Preliminaries

3.1. State Space Model

We consider a time-invariant dynamical system, which is represented by a non-linear, Gaussian continuous SS evolution model in discrete time $t \in \mathbb{Z}$ (Haykin, 2005):

$$x_t = f(x_{t-1}) + e_t, \quad e_t \sim \mathcal{N}(0, Q), \quad x_t \in \mathbb{R}^m. \quad (1)$$

Here, $x_t$ is the latent state vector of the system at time $t$, which evolves by a (possibly) non-linear state evolution
We assume that the transformations respond to some approximation (accurately or inaccurately). How-
function $f(\cdot)$, and by an additive Gaussian noise $e_t$, without external control. The system is partially observable, such that the state observations vector in time $t$, denoted $y_t$, is generated from the latent $x_t$ via

$$y_t = h(x_t) + v_t, \quad v_t \sim N(0, R), \quad y_t \in \mathbb{R}^n. \quad (2)$$

Here, $g(\cdot)$ is a (possibly) non-linear function, and $v_t$ represents additive Gaussian observation noise. The SS model is graphically described in Fig. 1 by a factor graph.

In practice, the state evolution model (1) is determined by the complex dynamics of the underlying system, while the observation model (2) is dictated by the nature and quality of the observations. For instance, $x_t$ can determine the location, velocity, and acceleration of a vehicle, while $y_t$ are measurements obtained from several sensors. The parameters of these models; e.g., the covariance matrices $Q$ and $R$, may be difficult to obtain accurately, often leading to performance degradation due to model mismatch.

### 3.2. Filtering Problem Formulation

We consider the problem of state estimation in real-time, i.e., the filtering problem. Here, at each time instance $t$, we are given access to a new noisy observation $y_t$, and the estimation is carried out on each incoming observation. Given the initial state $x_0$, the goal is to track the current latent state $x_t$ using the observations available so far, $y_1, \ldots, y_t$. This is opposed to smoothing considered in, e.g., (Satorras et al., 2019), where the goal is to recover $\{x_t\}_{t=1}^T$ from observations $\{y_t\}_{t=1}^T$ over a time horizon $T$.

We assume that the transformations $f(\cdot)$ and $h(\cdot)$ are known to some approximation (accurately or inaccurately). However, as opposed to the classical KF, the noise statistics $Q$ and $R$ are not known. Instead, we have access to a labeled data set containing a sequence of observations and their corresponding states. Since our architecture is inspired by the structure of the classic KF, we review it in the next section.

### 3.3. Model-Based Kalman Filter

The KF is a linear recursive estimator. In every time step $t$, it produces a new estimate $x_t$ using only the previous estimate $x_{t-1}$ as a sufficient statistic and the new observation $y_t$ in a constant computational complexity. When the evolution and observation functions are linear, i.e., $f(x) = F \cdot x$ and $h(x) = H \cdot x$, the noise is Gaussian, and the SS model parameters are known, the KF is the MMSE estimator.

The KF can be described by a two-step procedure: prediction and update. In the prediction step, the $a$ priori first- and second-order statistical moments are computed via

$$\hat{x}_{t|t-1} = F \cdot x_{t-1}, \quad \Sigma_{t|t-1} = F \cdot \Sigma_{t-1} \cdot F^\top + Q,$$

$$\hat{y}_{t|t-1} = H \cdot \hat{x}_{t|t-1}, \quad S_{t|t-1} = H \cdot \Sigma_{t|t-1} \cdot H^\top + R.$$  

In the update step, the $a$ posteriori statistical moments are computed given the new observed data $y_t$, via

$$\hat{x}_t = \hat{x}_{t|t-1} + \mathcal{K}_t \cdot \Delta y_t, \quad (3a)$$

$$\Sigma_t = \Sigma_{t|t-1} - \mathcal{K}_t \cdot S_{t|t-1} \cdot \mathcal{K}_t^\top. \quad (3b)$$

Here, $\Delta y_t$ is the innovation process

$$\Delta y_t = y_t - \hat{y}_{t|t-1} \quad (4)$$

and $\mathcal{K}_t$ is the Kalman filtering gain

$$\mathcal{K}_t = \Sigma_{t|t-1} \cdot H^\top \cdot S_{t|t-1}^{-1}. \quad (5)$$

An illustration of the KF is depicted in Fig. 2.

Two widely used variants of KF were proposed for models in which $f(\cdot)$ and/or $h(\cdot)$ are non-linear: the unscented Kalman filter, which is a sampling based approach; and EKF, which is linearization based and is considered our baseline (Julier & Uhlmann, 1997). When $f(\cdot)$ and $h(\cdot)$ are differentiable, they can be linearized by using their partial derivative (Jacobian) matrices, evaluated at $x_{t|t-1}$; i.e.,

$$\hat{F}_t = J_f (\hat{x}_{t|t-1}), \quad \hat{H}_t = J_h (\hat{x}_{t|t-1}). \quad (6)$$

The first-order statistical moments are propagated through the non-linearity, and the second-order statistical moments are evaluated by the first-order Taylor series expansion:

$$x_{t|t-1} = f (\hat{x}_{t|t-1}), \quad \hat{\Sigma}_{t|t-1} = \hat{F}_t \cdot \hat{\Sigma}_{t-1} \cdot \hat{F}_t^\top + Q,$$

$$\hat{y}_{t|t-1} = h (\hat{x}_{t|t-1}), \quad \hat{S}_{t|t-1} = \hat{H}_t \cdot \hat{\Sigma}_{t|t-1} \cdot \hat{H}_t^\top + R.$$  

Then, the EKF estimates $x_t$ in a similar manner as the KF. When $f(\cdot)$ and/or $h(\cdot)$ are highly non-linear or the noise is relatively high, the local linearity approximation may not hold, and the EKF can result in degraded performance.

### 4. KalmanNet

Here, we present KalmanNet; a hybrid, interpretable, data efficient architecture for online state estimation in non-linear dynamical systems with partial domain knowledge, combining NNs with MB Kalman filtering to cope with model mismatch and non-linearities (Shlezinger et al., 2020d).
KalmanNet: Neural Network Aided Kalman Filtering for Non-Linear Dynamics with Partial Domain Knowledge

4.1. Overview of the Design Process

Before presenting KalmanNet, we review two alternative candidate architectures for NN-aided filtering. To limit ourselves to low complexity recursive filters, we consider only those approaches in which \( x_{t-1} \) is a sufficient statistic of the observations \( \{ y_t \}_{t=1}^T \). (i) The vanilla NN architecture for filtering a time series uses an RNN in an end-to-end manner to recover \( x_t \) directly from \( y_t \) and (the internal state) \( x_{t-1} \), without relying on domain knowledge. We numerically show in Section 5 that this intuitive architecture does not learn to carry out Kalman filtering; i.e., recover a trajectory with arbitrary initial conditions and time horizon. (ii) A direct approach to adapt an RNN to account for the KF operation is to estimate \( x_{t|t-1} \), and then use the RNN to estimate an increment \( \Delta \hat{x}_t \) from the prior to posterior. As we show in Section 5, this architecture is not data efficient.

The drawbacks in directly using RNNs for the filtering tasks motivate preserving the flow of the KF in designing KalmanNet. Consequently, we first identify the specific computations that are based on unavailable knowledge. As detailed in Subsection 3.2, the functions \( f \) and \( h \) are known (though perhaps inaccurately); yet the covariance matrices \( Q \) and \( R \) are unavailable. As illustrated in Fig. 2, these missing statistical moments are used only for computing the KFG. Therefore, KalmanNet learns the KFG from data, rather than explicitly estimating the missing moments, as done in (Coskun et al., 2017), or alternatively, learning the complete state estimation task as in (Rangapuram et al., 2018).

Figure 2. Kalman filter block diagram. Here \( \{ \cdot \}^\top \) denotes transpose and \( Z^{-1} \) is the unit delay.

4.2. Architecture

KalmanNet combines the learned KFG in the overall KF flow, as illustrated in Fig. 3. The previous posterior estimate \( \hat{x}_{t-1} \) is used as a sufficient statistic to compute the next prior estimate \( \hat{x}_{t|t-1} \) by a forward pass. Similarly, the next observation \( y_t \) is predicted and the innovation value \( \Delta y_t \) is computed. State estimation is then carried out via (3), where \( \mathcal{K}_t \) is estimated using a dedicated NN.

It follows from (5) that computing \( \mathcal{K}_t \) involves tracking the second-order statistical moment \( \Sigma_t \), merely for producing the KFG. When the covariance matrices \( Q \) and \( R \) are fully known, the KFG does not depend on the current observation \( y_t \), and is not a function of the previous estimate \( \hat{x}_{t-1} \). Nonetheless, in the partial model information case or the non-linear case, the unknown statistical relationship of the SS model is encapsulated in these quantities. For this reason we provide the observation \( y_t \) and the previous estimate \( \hat{x}_{t-1} \) as inputs to the NN that computes the KFG. The recursive nature of the KFG computation indicates that its learned module should involve an internal memory element as an RNN; and to maintain low complexity we use a gated recurrent unit (GRU) (Chung et al., 2014).
We next address (ii); i.e., the inputs to the network. Theoretical intuition and empirical evidence suggest that to estimate the KFG in time $t$, it is better to provide the difference between the previous posterior and the previous prior,

$$\Delta \hat{x}_{t-1} = \hat{x}_{t-1} - \hat{x}_{t-1|t-2},$$

and the innovation sequence $\Delta \hat{y}_t$ (4), which is the difference between the observation at time $t$ and its prediction based on information available prior to time $t$. These difference sequences encapsulate the statistics of the noise signals. By removing the predictable components, the successive differences are a temporally uncorrelated time series that is induced by the noise statistics that we wish to learn.

The resulting architecture, illustrated in Fig. 4, is composed of a GRU with dedicated input and output layers. The input layer is fully connected, expanding the input dimensionality by a factor of 80. The GRU then provides the capacity for implicitly tracking the unknown second-order statistical moments used in computing the KFG. The output layer is fully connected, reshaping the output to the dimensions of $K_i$. Specifically, the input layer dimensions are $80 \cdot (m + n)$, and the dimensions of the output layer are $4 \cdot (m \cdot n)$. We use a GRU cell with hidden state of size $10 \cdot (m^2 + n^2)$; i.e., proportional to the size of the covariance matrices.

### 4.3. Training Scheme

KalmanNet is trained in a supervised manner using a data set of $N$ length $T$ trajectories, denoted $\{Y_i, X_i\}_{i=1}^N$. Here, $Y_i = [y^{(i)}_1, \ldots, y^{(i)}_T]$ and $X_i = [x^{(i)}_1, \ldots, x^{(i)}_T]$ are sequences of observation and true hidden state vectors, respectively, from trajectory $i$. The initial state for each trajectory, $x_0$, is also given as part of the data set. The learning procedure unrolls the recurrence to $T$ time steps and uses sequence-to-sequence training. For each trajectory $i$ we use the regularized mean-squared error (MSE) loss

$$\ell_i(\Theta) = \frac{1}{T} \sum_{t=1}^T \left\| \Psi(\hat{x}_{i,t-1|t-1} - x^{(i)}_t) \right\|^2 + \gamma \cdot \| \Theta \|^2,$$

where $\Psi(\cdot)$ is the output of KalmanNet, $\Theta$ is its trainable parameters, and $\gamma$ is a regularization coefficient. To optimize $\Theta$, for every batch indexed by $k$, we randomly choose $M < N$ trajectories indexed by $i_{k1}, \ldots, i_{kM}$, computing the mini-batch loss as $L_k(\Theta) = \frac{1}{M} \sum_{j=1}^M \ell_{ikj}(\Theta)$. 

### 5. Experiments and Results

In this section we present an extensive numerical evaluations of KalmanNet. We consider three experimental setups: a synthetic toy example for evaluating the gains of KalmanNet over the aforementioned RNN state estimators, as well as the MB KF and EKF, in controlled linear and non-linear dynamic SS models; real-time tracking of the Lorenz attractor, which is a highly non-linear system with chaotic behavior; and localization in real world dynamics based on the Michigan NCLT data set (Carlevaris-Bianco et al., 2016). In the synthetic linear setup we show that KalmanNet outperforms KF in the presence of model mismatch, whereas in the non-linear setup it outperforms it with full domain knowledge. Then in the Lorenz attractor setup we show how these gains of KalmanNet viewed for the synthetic problems are translated into concrete benefits for tracking a challenging chaotic system in real-time, while providing the ability to overcome crude sampling based decimation. Finally, we demonstrate that all these gains directly result in improved capability of real-time tracking with real-life data.

We train KalmanNet using ADAM (Kingma & Ba, 2014). In the experiments involving synthetic data, the SS model is generated using diagonal covariance matrices; i.e., $Q = q^2 \cdot I$, $R = r^2 \cdot I$, and we define $\nu \triangleq \frac{q^2}{r^2}$. In those setups, we depict the MSE vs. $1/r^2$; i.e., the observation noise decreases along the x-axis.

### 5.1. Synthetic Toy Model

In our first experimental study we generate training and test data from a controlled SS model. Our aim here is twofold: 1) evaluate the design of KalmanNet by comparing it to the DD benchmarks discussed in Subsection 4.1; 2) compare KalmanNet to its MB counterparts for different forms of system dynamics. We evaluate both a linear SS model, where the KF is MSE optimal, as well as a non-linear setup, where the common KF variant is the EKF.

**Linear Setup** Our first set of experiments uses training and test data that were generated from linear Gaussian SS models of dimensions $2 \times 2$, $5 \times 5$, and $10 \times 1$; e.g., the state vector $x_t$ has 10 entries, while the observation $y_t$ is a scalar quantity. We set $F$ and $H$ to take the controllable canonical and inverse canonical forms, respectively.

We first evaluate the design of KalmanNet by comparing it
to alternative NN architectures of similar capacity. To that aim, we consider the $2 \times 2$ SS model with trajectory length $T = 20$, and use three NN-based benchmarks: Vanilla RNN, described as alternative $(i)$ in Subsection 4.1; MB RNN, which is alternative $(ii)$ in Subsection 4.1; and MB RNN with differences inputs, that is the same as alternative $(ii)$ only with its inputs being \( \Delta x_{t-1} \) and \( \Delta y_{t} \), as used by KalmanNet. All NN-based estimators share the same RNN architecture and are trained with the same learning parameters. The learning curves are depicted in Fig. 5, where we observe how each of the key design considerations of KalmanNet affect the learning curve: the incorporation of the known \( F \) and \( H \) allows the MB RNN to outperform the vanilla RNN, although both converge slowly and fail to achieve the MMSE; using the difference sequences as input notably improves the convergence rate of the MB RNN; and learning is further improved by using the RNN for recovering the KFG as part of the KF flow, as done by KalmanNet, rather than for directly estimating \( x_{t} \).

To further evaluate the gains of the hybrid architecture of KalmanNet, we checked if its learning is transferable. Here, we tested the pretrained model of vanilla RNN, MB RNN, and KalmanNet using trajectories with different initial conditions and a longer time horizon ($T = 200$) than those on which it was trained. The results, summarized in Table 1, show that KalmanNet maintains achieving the MMSE, implying that it learned to carry out Kalman filtering. The MB RNN and vanilla RNN are more than 50 [dB] from the MMSE, implying that learning is not transferable and that they do not learn to implement Kalman filtering.

We next compare KalmanNet to its MB counterparts. Fig. 6a considers the case where the KF operates with accurate SS knowledge, and thus achieves the minimal MSE. Observing Fig. 6a, we note that KalmanNet learns to achieve the MMSE, coinciding with the MSE of KF. To show that KalmanNet is invariant to $T$ and to the training initial conditions $x_{0}$, we tested KalmanNet on trajectories with different $T$ and $x_{0}$ than those on which it was trained. The results demonstrate again that KalmanNet is not tailored to the trajectories presented during training, and it learns to implement the KF, whose dependency is only on the SS model. Fig. 6b studies the robustness of KalmanNet to model uncertainty. Here, we plug in \( F_{0} \) as a design parameter for both filters, while the data is generated from an SS model based on a rotated matrix \( F_{\alpha^*} = \left( \begin{array}{cc} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{array} \right) \cdot F_{0} \), with $\alpha \in \{10^\circ, 20^\circ\}$. Such scenarios represent a setup in which the analytical approximation of the SS model used as a plugged-in parameter in the KF flow differs from the true model. Here, KalmanNet achieves a 3 [dB] gain over KF. In particular, despite the fact that KalmanNet implements the KF with inaccurate \( F \), it learns to apply an alternative KFG, resulting in MSE within a minor gap from the MMSE; i.e., from the KF with the true \( F_{\alpha^*} \) plugged in.

**Non-Linear Setup** In this setup we consider a non-linear SS model, where the state evolution and observation functions are computed in a component-wise manner via

\[
\begin{align}
\mathbf{f}(x) &= \alpha \cdot \sin(\beta \cdot x + \phi) + \delta, & x \in \mathbb{R}^2, \quad (9a) \\
\mathbf{h}(x) &= a \cdot (b \cdot x + c)^2, & y \in \mathbb{R}^2. \quad (9b)
\end{align}
\]

We simulate two settings: full domain knowledge, where the design parameters plugged into KF and KalmanNet are those of the true generative model; and partial domain knowledge, where the state evolution parameters used by the filters slightly differ from the true model. The parameters are summarized in Table 2, and we set $\nu = 0$ [dB] and $T = 100$.

In Fig. 6c we see that for full domain knowledge and in the low noise regime, there is no MSE degradation due to
which is also achieved by KalmanNet. For higher noise
differential equation:

\[ \dot{x}_t = A(x_t) \cdot x_t, \quad A(x_t) = \begin{pmatrix}
-10 & 10 & 0 \\
28 & -1 & -x_3 \\
0 & x_1 & -\frac{8}{3}
\end{pmatrix}. \tag{10} \]

To generate a data set for this attractor, we use a 5th order
Taylor series approximation; i.e., \( J = 5 \), resulting in

\[ F(x_t) = I + \sum_{j=1}^{J} \left( A(x_t) \cdot \Delta t \right)^{j} \cdot \frac{1}{j!}. \tag{11} \]

Unless stated otherwise, we use \( \Delta t = 10^{-2} \), and both
training and testing trajectories are generated with \( T = 30 \).

**5.2. Lorenz Attractor**

The Lorenz system is a system of ordinary differential equations. The Lorenz attractor is a set of three-dimensional
chaotic solutions of the Lorenz system. This example is
chosen because it represents challenging trajectories for
online tracking that can be generated synthetically. The
noiseless state evolution is given by the continuous-time
differential equation:

non-linearity and EKF demonstrates optimal performance,
which is also achieved by KalmanNet. For higher noise
levels, EKF fails to overcome the non-linearity, and a signif-
icant MSE degradation is observed. Nonetheless, by learn-
ing to compute the KFG from data, KalmanNet manages to
overcome this pitfall and achieves superior MSE. For the
partial information case, EKF notably degrades due to the
model mismatch. In all experiments, KalmanNet overcomes
such mismatches, and its performance is within a minor gap
of the MMSE for such setups. We thus conclude that in the
presence of harsh non-linearities as well as model uncer-
tainty due to inaccurate approximation of the underlying
dynamics, where MB variations of the KF fail, Kalman-
Net learns to approach the MMSE while maintaining the
real-time operation and low complexity of the KF.

**Partial Information**

In the second set of experiments we demonstrate that KalmanNet outperforms EKF in the case of partial information. We again set \( h \) to be the identity mapping, and consider three forms of mismatch: inaccurate modeling of the noise, a crude approximation of the state
evolution dynamics (11), and imperfect knowledge of the
observations model \( h \) due to rotation.

In Fig. 8a we observe the sensitivity of EKF to inaccurate
noise modeling. As KalmanNet does not rely on explicit
modeling of the noise, it outperforms the EKF in this case. Furthermore, in practice, one often has access only to a crude approximation of the true state evolution. In Fig. 8b we demonstrate that KalmanNet outperforms in this case, in which the crude approximation is represented by a 2nd order Taylor series expansion. In many real world problems, sensors can be misaligned; therefore, coping with the case where the observation is slightly rotated is of practical importance. In Fig. 8c we show that KalmanNet also achieves superior MSE in the presence of such inaccurate modeling of the observations process, which is rotated by 1°.

**Sampling from Continuous Time to Discrete Time**  
We evaluate KalmanNet in handling mismatches due to sampling a process described by a continuous-time model into discrete-time. We generate a high resolution Lorenz attractor with $\Delta t = 10^{-5}$. We then sample observations from the (approximate) continuous-time evolution process by a ratio of $\frac{1}{2000}$ and get a decimated process with $\Delta t = 0.02$. From the decimated process, we generate an observation sequence with observation noise $\frac{1}{\sqrt{2}} = 0 [\text{dB}]$. To guarantee a fair comparison, we also give EKF access to labeled data to optimize the value of $q^2$. In Fig. 9 we present a trajectory with length $T = 30$, noting that KalmanNet outperforms EKF with 5.1488 [dB] gain on average. In Fig. 10 we demonstrate that using KalmanNet learning is transferable from short trajectories to long trajectories; when trained with $T = 30$ and tested on $T = 3000$, KalmanNet still outperforms the EKF with 4.411 [dB] gain on average. We also depict the trajectory produced by the vanilla RNN, which achieves similar results as KalmanNet when tested on $T = 30$, yet is unable to transfer from short trajectories to long ones.

**5.3. Real World Dynamics: Michigan NCLT Dataset**

Here, we evaluate KalmanNet in real world localization applications. We consider real-time tracking using the Michigan NCLT data set (Carlevaris-Bianco et al., 2016). It comprises different trajectories, providing various (possibly) noisy sensor readings (e.g., GPS and odometer) and the ground truth location ($p$) of a moving Segway robot. Given these noisy readings, the goal is to localize more accurately than the raw measurements at any given time. For this setup, we use constant velocity ($v$) motion equations, and get a linearized model for the Segway kinematics on each axis:

$$
\begin{pmatrix}
\dot{p} \\
\dot{v}
\end{pmatrix} =
\begin{pmatrix}
v \\
0
\end{pmatrix},
\tilde{F} =
\begin{pmatrix}
1 & \Delta t \\
0 & 1
\end{pmatrix}.
$$

The full model used by KalmanNet and EKF is given by

$$
F =
\begin{pmatrix}
\tilde{F} & 0 \\
0 & \tilde{F}
\end{pmatrix},
Q = q^2 \Delta t I_{4 \times 4},
H =
\begin{pmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}.
$$

We arbitrarily use the session with date 2012-01-22 that consists of a single trajectory of 0.1[Km]. Sampling at 1[Hz] results in 5,850 time steps. We removed unstable readings and were left with 5,556 time steps. The trajectory was split into three sections: 85% for training (23 sequences of length $T = 200$), 10% for validation (2 sequences, $T = 200$), and 5% for testing (1 sequence, $T = 277$).

We use the odometer to extract noisy velocity readings. Due to not having a direct measurement for positioning, the odometry-based position (which is directly integrated from its velocity readings) starts drifting away. This scenario is typical for many applications where the GPS position is not available indoors and one relies on noisy odometer readings for self-localization. Inevitably, the estimate drifts from the true position. Fig. 11 and Table 3 demonstrate the superiority of KalmanNet for such scenarios. EKF, for which the matrices $Q$ and $R$ were optimized through a grid search, blindly follows the odometry trajectory and is incapable of accounting for the drift, producing a very similar or even worse estimation than the integrated velocity. The vanilla RNN, which is agnostic of the motion model (12), fails to localize. KalmanNet overcomes the errors induced by the noisy odometry observations, and provides the most accurate real-time locations, demonstrating the gains of combining MB KF-based inference with integrated DD modules for real world applications.

Table 3. Numerical MSE in [dB] for the NCLT experiment.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>EKF</th>
<th>KalmanNet</th>
<th>Vanilla RNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE [dB]</td>
<td>25.47</td>
<td>25.385</td>
<td>22.2</td>
<td>40.21</td>
</tr>
</tbody>
</table>

Figure 9. Lorenz attractor sampling: $T_{\text{training}} = 30$, $T_{\text{testing}} = 30$. KalmanNet gain over EKF is 5.1 [dB].

Figure 10. Lorenz attractor - sampling: $T_{\text{training}} = 30$, $T_{\text{testing}} = 3000$. KalmanNet gain over EKF is 0.4 [dB].
KalmanNet: Neural Network Aided Kalman Filtering for Non-Linear Dynamics with Partial Domain Knowledge

6. Conclusions

In this work we studied KalmanNet, which combines deep learning with the classic KF. Our design identifies the SS-model-dependent computations of the KF, replacing them with a dedicated NN. Doing so enables KalmanNet to carry out real-time state estimation in the same manner as the KF, while learning to overcome model mismatches and non-linearities. KalmanNet uses a relatively compact NN, that can be trained with a relatively small data set, and infers a reduced complexity, making it applicable for high dimensional SS models and computationally limited devices.

References


KalmanNet: Neural Network Aided Kalman Filtering for Non-Linear Dynamics with Partial Domain Knowledge
Supplementary Material

Guy Revach 1  Adrià López Escoriza 1,2  Nir Shlezinger 3  Ruud J. G. van Sloun 4,5  Yonina C. Eldar 6

In these supplementary notes we provide a detailed description of the experimental setup used in our numerical evaluation of the proposed KalmanNet architecture. In particular, Section 1 detailed the alternate deep neural network (DNN)-aided state estimators to which KalammNet is compared, while Sections 2-3 provide details and notations used in the description of the experiments in the sequel. Then, we elaborate on the exact parameters used in the three experimental studies presented in the work: the synthetic toy model; the Lorentzian attractor; and the NCLT localization scenario, in Sections 4-6, respectively. Finally, we detail the simulation environment in terms of software and hardware, with which the experiments were conducted, in Section 7.

1. Comparing Alternative Architectures

1. The vanilla neural network (NN) architecture for filtering a time series, uses a recurrent neural network (RNN) in an end-to-end manner to recover \( \hat{x}_t \) directly from \( y_t \) and (the internal state) \( x_{t-1} \), without relying on domain knowledge.

2. A direct approach to adapt an RNN to account for the Kalman filter (KF) operation given in Fig. 1 is to estimate \( \hat{x}_{t|t-1} \) and then use the RNN to estimate an increment \( \Delta \hat{x}_t \) from the prior to posterior. This approach only accounts for \( f \) and not \( h \).

3. The third approach given in Fig. 2 is to use \( \Delta \hat{x}_{t-1} \) and \( \Delta \hat{y}_t \) as input to the network.

2. KalmanNet Structure

Unless stated otherwise, the number of gated recurrent unit (GRU) cells used in KalmanNet is \( N_{GRU} = 1 \).

3. Notation for Training Scheme

We denote \( N \) as the number of training trajectories, \( N_V \) as the number of validation trajectories, and \( N_T \) as the number of testing trajectories. We denote \( M \) as the number of training trajectories in a mini-batch and \( N_O \) as a predefined number of optimization iterations. In each optimization iteration we use \( M \) examples at random from the training set and compute their average loss. We denote \( \eta \) as the learning rate and \( \gamma \) as the value of the \( \ell^2 \) regularization. We
use the notation $[K]$ for $10^3$ and $[m]$ for $10^{-3}$.

4. Synthetic Toy Model

4.1. Linear Setup

Our first set of experiments uses training and test data that were generated from linear Gaussian state-space (SS) models of dimensions $2 \times 2$, $5 \times 5$, and $10 \times 1$; i.e., the observation vector $y_t$ has 10 entries, while the state $x_t$ is a scalar quantity. We set $F$ to take the controllable canonical form

$$F_{5 \times 5} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad (1)$$

and $H$ to take the inverse canonical form

$$H_{5 \times 5} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}. \quad (2)$$

The parameters for the $2 \times 2$ experiment are given in Table 1, setting $T = 50$ for training and $\nu = 0$ [dB]. The parameters for the $5 \times 5$ experiment are given in Table 2, setting $T = 20$ for training and $\nu = 0$ [dB]. The parameters for the $10 \times 1$ experiment are given in Table 3, setting $T = 10$ for training and $\nu = 0$ [dB].

4.2. State-Space Rotation

For this experiment we used a $2 \times 2$ system, with $T = 20$ for training and $\nu = 0$ [dB]. We plug in $F_0$ as a design parameter for both filters, while the data is generated from an SS model based on a rotated matrix $F_{\alpha^o}$ with $\alpha \in \{0^o, 20^o\}$. The parameters for $\alpha = 10^o$ are given in Table 4 and the parameters for $\alpha = 200^o$ are given in Table 5.

$$F_{\alpha^o} = \begin{pmatrix} \cos \alpha \\ \sin \alpha \\ \sin \alpha \\ \cos \alpha \end{pmatrix} \cdot F_0. \quad (3)$$

4.3. Non-Linear Setup

In this setup we consider a non-linear SS model, where the state evolution and observation functions are computed in a component-wise manner via

$$f(x) = \alpha \cdot \sin(\beta \cdot x + \phi) + \delta, \quad x \in \mathbb{R}^2, \quad (4a)$$

$$h(x) = a \cdot (b \cdot x + c)^2, \quad y \in \mathbb{R}^2. \quad (4b)$$

The model parameters are summarized in Table 6, and we set $\nu = 0$ [dB] and $T = 100$. The learning parameters are summarized in Table 7.

<table>
<thead>
<tr>
<th>Table 1. Linear toy problem with $2 \times 2$ parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$ [K]</td>
</tr>
<tr>
<td>$N_T$ [K]</td>
</tr>
<tr>
<td>$M$ [K]</td>
</tr>
<tr>
<td>$\eta$ [m]</td>
</tr>
<tr>
<td>$\gamma$ [m]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2. Linear toy problem with $5 \times 5$ parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$ [K]</td>
</tr>
<tr>
<td>$N_T$ [K]</td>
</tr>
<tr>
<td>$M$ [K]</td>
</tr>
<tr>
<td>$\eta$ [m]</td>
</tr>
<tr>
<td>$\gamma$ [m]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3. Linear toy problem with $10 \times 1$ parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\eta$ [m]</td>
</tr>
<tr>
<td>$\gamma$ [m]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4. $\alpha = 10^o$ parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$ [K]</td>
</tr>
<tr>
<td>$N_T$ [K]</td>
</tr>
<tr>
<td>$M$</td>
</tr>
<tr>
<td>$\eta$ [m]</td>
</tr>
<tr>
<td>$\gamma$ [m]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5. $\alpha = 20^o$ parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$ [K]</td>
</tr>
<tr>
<td>$N_T$ [K]</td>
</tr>
<tr>
<td>$M$</td>
</tr>
<tr>
<td>$\eta$ [m]</td>
</tr>
<tr>
<td>$\gamma$ [m]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 6. Non-linear toy problem model parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\beta$</td>
</tr>
<tr>
<td>$\phi$</td>
</tr>
<tr>
<td>$\delta$</td>
</tr>
<tr>
<td>$\alpha$</td>
</tr>
<tr>
<td>$\beta$</td>
</tr>
<tr>
<td>$\phi$</td>
</tr>
</tbody>
</table>
Table 7. Non-linear toy problem learning parameters.

<table>
<thead>
<tr>
<th>h [dB]</th>
<th>12.04</th>
<th>6.02</th>
<th>0</th>
<th>20</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nₜ</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>Nₒ</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>η</td>
<td>0.1 [m]</td>
<td>0.1 [m]</td>
<td>0.1 [m]</td>
<td>0.1 [m]</td>
<td>0.1 [m]</td>
</tr>
<tr>
<td>γ</td>
<td>0.06 [m]</td>
<td>0.06 [m]</td>
<td>0.06 [m]</td>
<td>0.06 [m]</td>
<td>0.06 [m]</td>
</tr>
</tbody>
</table>

5. Lorenz Attractor

Unless stated otherwise, we use $\Delta t = 10^{-2}$, and both training and testing trajectories are generated with $T = 30$.

5.1. Full Information with h Identity

The learning parameters for the case where $h$ is identity and $\nu = 0$ [dB] are given in Table 8. The learning parameters for the case where $h$ is identity and $\nu = -20$ [dB] are given in Table 9.

Table 8. Lorenz Attractor h Identity $\nu = 0$ [dB].

<table>
<thead>
<tr>
<th>h [dB]</th>
<th>0</th>
<th>20</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nₜ</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>M</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>ν</td>
<td>8 [m]</td>
<td>8 [m]</td>
<td>8 [m]</td>
</tr>
<tr>
<td>γ</td>
<td>1.0 $\times$ 10⁻⁷</td>
<td>1.0 $\times$ 10⁻⁷</td>
<td>1.0 $\times$ 10⁻⁷</td>
</tr>
<tr>
<td>Nₒ</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 9. Lorenz Attractor h Identity $\nu = -20$ [dB].

<table>
<thead>
<tr>
<th>h [dB]</th>
<th>0</th>
<th>20</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nₜ</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>M</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>ν</td>
<td>8 [m]</td>
<td>8 [m]</td>
<td>8 [m]</td>
</tr>
<tr>
<td>γ</td>
<td>1.0 $\times$ 10⁻⁷</td>
<td>1.0 $\times$ 10⁻⁷</td>
<td>1.0 $\times$ 10⁻⁷</td>
</tr>
<tr>
<td>Nₒ</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

5.2. Full Information with h Non-Linear

For the case where $h$ is non-linear we use a transformation to spherical coordinates.

From spherical to Cartesian

$$x = r \sin \theta \cos \phi$$

$$y = r \sin \theta \sin \phi$$

$$z = r \cos \theta$$

From Cartesian to spherical

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\phi = \tan^{-1} \left( \frac{y}{x} \right)$$

$$\theta = \cos^{-1} \left( \frac{z}{\sqrt{x^2 + y^2 + z^2}} \right)$$

The learning parameters for this case are given in Table 10.

Table 10. Lorenz Attractor h Non-Linear $\nu = 0$ [dB].

<table>
<thead>
<tr>
<th>h [dB]</th>
<th>10</th>
<th>0</th>
<th>20</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nₜ</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>Nₒ</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>ν</td>
<td>8 [m]</td>
<td>8 [m]</td>
<td>8 [m]</td>
<td>8 [m]</td>
</tr>
<tr>
<td>γ</td>
<td>1.0 $\times$ 10⁻⁷</td>
<td>1.0 $\times$ 10⁻⁷</td>
<td>1.0 $\times$ 10⁻⁷</td>
<td>1.0 $\times$ 10⁻⁷</td>
</tr>
<tr>
<td>Nₒ</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

5.3. Partial Information

Lorenz Attractor State Evolution Model Mismatch In this experiment we consider a crude approximation of the state-evolution model. Table 11 describes the learning parameters for the case where the order of the Taylor approximation is $J = 2$ and $\nu = -20$ [dB].

Table 11. Lorenz Attractor state-evolution model mismatch.

<table>
<thead>
<tr>
<th>h [dB]</th>
<th>0</th>
<th>20</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nₜ</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>M</td>
<td>2 [K]</td>
<td>2 [K]</td>
<td>2 [K]</td>
</tr>
<tr>
<td>Nₒ</td>
<td>800</td>
<td>800</td>
<td>800</td>
</tr>
<tr>
<td>ν</td>
<td>8 [m]</td>
<td>8 [m]</td>
<td>8 [m]</td>
</tr>
<tr>
<td>γ</td>
<td>1.0 $\times$ 10⁻⁷</td>
<td>1.0 $\times$ 10⁻⁷</td>
<td>1.0 $\times$ 10⁻⁷</td>
</tr>
<tr>
<td>Nₒ</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Rotation Observation Model Mismatch We consider imperfect knowledge of the observations model $h$ due to rotation of 1°. We rotate the observation using the following equations:

Denoting $\alpha$ as the yaw (rotation angle in the $xy$ plane), $\beta$ as the pitch (rotation angle in the $xz$ plane), and $\gamma$ as the roll
(rotation angle in the $yz$ plane), we define the matrices:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{pmatrix}$$

(7a)

$$R_y = \begin{pmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{pmatrix}$$

(7b)

$$R_z = \begin{pmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

(7c)

And we denote the general matrix rotation:

$$R(\alpha, \beta, \gamma) = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha)$$

(8)

So the true observation model is formed by

$$H_{gen} = H_{model} \cdot R(\alpha, \beta, \gamma) = I \cdot R(\alpha, \beta, \gamma)$$

(9)

Table 12 describes the learning parameters where $\nu = -20$ [dB].

<table>
<thead>
<tr>
<th>$\nu$ [dB]</th>
<th>0</th>
<th>20</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_T$</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>$N_O$</td>
<td>800</td>
<td>800</td>
<td>800</td>
</tr>
<tr>
<td>$M$</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>$\eta$</td>
<td>1 [m]</td>
<td>2 [m]</td>
<td>2 [m]</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$1 \cdot 10^{-3}$</td>
<td>$1 \cdot 10^{-3}$</td>
<td>$1 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>$\gamma_{GRU}$</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 12. Lorenz Attractor observation model mismatch.**

### Sampling from Continuous Time to Discrete Time

We evaluate KalmanNet when handling inaccurate approximations due to sampling a process described by a continuous-time model into discrete time. We generate a high resolution Lorenz attractor with $(\Delta t = 10^{-5})$ and without process noise. We then sample observations from the (approximate) continuous-time evolution process by a ratio of $\frac{1}{2000}$ and get a decimated process with $\Delta t = 0.02$. From the decimated process, we generate an observation sequence with observation noise $\frac{1}{277} = 0$ [dB]. Table 13 describes the learning parameters.

### 6. Real World Dynamics: Michigan NCLT Dataset

To conclude our experimental study, we evaluate KalmanNet on the Michigan NCLT dataset.

For the NCLT dataset we use the uniform velocity motion equations. Following are the differential equations that describe the dynamics and the the transition matrix for one component:

$$\begin{cases} \frac{d}{dt} r = v \\ \frac{d}{dt} v = F - r^2 \cdot I_{4 \times 4} \end{cases}$$

(10)

The transition matrix and noise distribution are

$$F = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad Q = r^2 \cdot \Delta t \cdot I_{4 \times 4}$$

(11)

And the observation model is then defined by

$$H = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad R = \sigma^2 \cdot I_{2 \times 2}$$

(12)

where $\sigma^2$ and $r^2$ are tunable parameters of the extended Kalman filter (EKF).

In our experiment we arbitrarily use the session with date 2012-01-22 that consists of a single trajectory of 0.1[Km]. Sampling at 1[Hz] results in 5,850 time steps. We remove outliers; i.e., unstable readings, and are left with 5,556 time steps. The trajectory is split sequentially into three sections: 85% for training, 10% for validation (2 sequences of length $T = 200$), and 5% for testing (1 sequence, $T = 277$). The learning parameters are described in Table 14.

<table>
<thead>
<tr>
<th>$N$</th>
<th>23x200</th>
<th>31x150</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_T$</td>
<td>2x200</td>
<td>3x150</td>
</tr>
<tr>
<td>$N_O$</td>
<td>1x277</td>
<td>1x277</td>
</tr>
<tr>
<td>$M$</td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>$\eta$</td>
<td>2 [m]</td>
<td>2 [m]</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>1 [m]</td>
<td>10 [m]</td>
</tr>
<tr>
<td>$\gamma_{GRU}$</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 14. NCLT dataset.**

The detailed architecture of the vanilla RNN was selected based on experimental trials, in which it achieved the best mean-squared error accuracy among all considered architectures of varying capacities and different number of GRU layers. Nonetheless, as detailed in the manuscript, this configuration also achieved relatively poor performance compared to the hybrid model-based data-driven KalmanNet.
7. Tools and Simulation Environment

7.1. Software

The entire project was coded using Python 3.8. The PyTorch 1.7.1 package was used for the RNN development and all other tensor based operations such as the EKF implementation. Other libraries such as NumPy, pandas, and matplotlib were used for secondary tasks such as dataset management or data representation.

7.2. Hardware

The experiments were run on a normal laptop computer (MacBook Pro 15-inch, 2017) with a 2.9 GHz quad-core Intel Core i7 and 16GB of 2133MHz LPDDR3 onboard RAM. The training time usually took between 1 to 5 hours depending on the optimization iterations, number of GRU cells, and other parameters of the training session.