RTSNet: Learning to Smooth in Partially Known State-Space Models

Guy Revach, Xiaoyong Ni, Nir Shlezinger, Ruud J. G. van Sloun, and Yonina C. Eldar

Abstract—**The smoothing task, which considers recovery of a sequence of hidden state variables from a sequence of noisy observations, is core to many signal processing applications. A widely popular smoother is the Rauch-Tung-Striebel (RTS) algorithm, which achieves minimal mean-squared error recovery with low complexity in dynamic systems that are represented as linear Gaussian state space (SS) models. However, this model-based algorithm is limited in systems that are only partially known, as well as non-linear and non-Gaussian. In this work we propose RTSNet, a highly efficient model-based and data-driven smoothing algorithm suitable for partially known SS models. RTSNet integrates dedicated trainable models into the flow of the classical RTS smoother, while iteratively refining its sequence estimate via deep unfolding methodology. As a result, RTSNet learns from data to reliably smooth when operating under model mismatch and non-linearities while retaining the efficiency and interpretability of the model-based RTS algorithm. Our empirical study demonstrates that RTSNet overcomes non-linearities and model mismatch, outperforming classic smoothers operating with both mismatched and accurate domain knowledge. Moreover, while RTSNet is based on compact neural networks, which leads to faster training and inference times, it is shown to outperform previously proposed deep smoothers in non-linear settings.**

I. Introduction

A broad range of applications in signal processing and control require estimation of the hidden state of a dynamical system from noisy observations. Such tasks arise in localization, tracking, and navigation [2]. State estimation by filtering and smoothing date back to the work of Wiener from 1949 [3]. Filtering (also known as real-time tracking) is the task of estimating the current state from past and current observations, while smoothing deals with simultaneous state estimation across the entire time horizon using all available data.

Arguably the most common and celebrated filtering algorithm is the Kalman filter (KF) proposed in the early 1960s [4]. The KF is a low-complexity implementation of the minimum mean-squared error (MMSE) estimator for time-varying systems in discrete-time that are characterized by a linear state space (SS) model with additive white Gaussian noise (AWGN). The Rauch-Tung-Striebel (RTS) smoother [5], also referred to here as the Kalman smoother (KS), adapts the KF for smoothing in discrete-time. The KS implements MMSE estimation for linear Gaussian SS models by applying a recursive forward pass, i.e., from the past to the future by directly applying the KF, followed by a recursive update backward pass.

While the classical KF and KS both assume linear SS models, many problems encountered in practice are governed by non-linear dynamical equations. Therefore, shortly after the introduction of the original KF and KS, non-linear variations of them were proposed [2, Ch. 10]. Since the KS extends the KF, its non-linear variations are based on the corresponding non-linear variations of the KF. These include the extended KS (EKS), which is an adaptation of the extended KF (EKF) to smoothing [6], [7], and the unscented KS (UKS) that is based on the unscented KF (UKF) [8]. An alternative family of methods, for state estimation in non-linear, non-Gaussian SS models, are these based on sequential sampling, such as the family of particle smoothers (PSs) [9]–[11].

The KS and its variants are model-based (MB) algorithms; namely, they rely on accurate characterization of the underlying dynamics as a SS model and full knowledge of it. Real world systems in many practical use cases are complex, non-linear, and may be difficult to characterize faithfully with a fully known tractable SS model. Consequently, despite its low complexity and theoretical soundness, applying the KS in practical scenarios may be limited due to its critical dependence on accurate knowledge of the underlying SS model. A common approach to deal with partially known SS models is to impose a parametric model and then estimate its parameters. This can be achieved by jointly learning the parameters and state sequence using expectation maximization [12], [13] and Bayesian probabilistic algorithms [14], [15], or by selecting from a set of *a priori* known models [16]. When training data is available, it is commonly used to tune the missing parameters in advance [17], [18]. These strategies are restricted to an imposed parametric model on the underlying dynamics (e.g., linear models with Gaussian noises), and thus may still lead to mismatched operation. MB algorithms designed to cope with some degree of uncertainty in the SS models, e.g., [19], [20], are typically designed for the worst case deviation between the postulated model and the ground truth, rarely approaching the performance achievable with full domain knowledge. Furthermore, the non-linear variants of the KS do not share its MMSE optimality, and their performance degrades under strong non-linearities.

Data-driven (DD) approaches are an alternative to MB algorithms, relaxing the requirement for explicit and accurate knowledge of the underlying model. Many of these strategies are now based on deep neural networks (DNNs), which have shown remarkable success in capturing the subtleties of complex processes [21]. When there is no characterization of the dynamics, one can train deep learning systems designed

Parts of this work were presented at the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2022 [1]. G. Revach and X. Ni are with the Institute for Signal and Information Processing (ISI), D-ITET, ETH Zürich, 8006 Zürich, Switzerland, (e-mail: grevach@ethz.ch; xiaoni@student.ethz.ch). N. Shlezinger is with the School of ECE, Ben-Gurion University of the Negev, Beer Sheva, Israel (e-mail: nirshl@bgu.ac.il). R. J. G. van Sloun is with the EE Dpt., Eindhoven University of Technology, and with Phillips Research, Eindhoven, The Netherlands (e-mail: r.j.g.v.sloun@tue.nl). Y. C. Eldar is with the Faculty of Math and CS, Weizmann Institute of Science, Rehovot, Israel (e-mail: yonina.eldar@weizmann.ac.il). We thank Hans-Andrea Loeliger for helpful discussions.

for processing time sequences, e.g., recurrent neural networks (RNNs) [22] and attention mechanisms [23], for state estimation in intractable environments [24]. Yet, they do not incorporate domain knowledge such as structured SS models in a principled manner, while requiring many trainable parameters and large data sets even for simple sequence models [25] and lack the interpretability of MB methods. It is also possible to combine DNNs with variational inference in the context of state space models, as in [26]–[30]. This is done by casting the Bayesian inference task as the optimization of a parameterized posterior and maximizing an objective. However, the learning procedure tends to be complex and prone to approximation errors since these methods often rely on highly parameterized models. Furthermore, their applicability to use cases with a bounded delay on hardware-limited devices is limited.

An alternative DD approach for state estimation in SS models uses DNNs to encode the observations into some latent space that is assumed to obey a simple SS model, typically a linear Gaussian one. State estimation is then carried out based on the extracted features [31]–[35], and can be followed by another DNN decoder [30]. This form of DNN-aided state estimation is intended to cope with complex and intractable observations models, e.g., when processing visual observations, while one should still know (or estimate) the state evolution. When the SS model is known, DNNs can be applied to improve upon MB inference, as done in [36], where graph neural networks are used in parallel with MB smoothing. However, this approach requires full knowledge of the SS model, as in MB smoothers.

Real world smoothing applications often involve partially known dynamics, where one has access to an approximation of some parts of the SS model based on, e.g., understanding of the underlying physics or established motion models. In such scenarios, both MB smoothing and DD methods based on DNNs may be limited in their performance and suitability. In our previous work [37] we derived a hybrid MB/DD implementation of the KF following the emerging MB deep learning methodology [38], [39]. The augmentation of the KF with a dedicated DNN was shown to result in a filter that approaches MMSE performance in partially known dynamics, while being operable at high rates on limited hardware [40]. Further, the interpretable nature of the resulting architectures was leveraged to provide reliable measures of uncertainty [41] and support unsupervised training [42]. These findings, which all considered a filtering task, motivate deriving a hybrid MB/DD smoothing algorithm.

In this work we propose RTSNet, which combines trainable RNNs with the flow of the KS. We design RTSNet for partially known SS models, where estimates of the evolution and observation models of the underlying system are given, which can be obtained based on knowledge of the setup or approximated from data. Yet, the distribution of the process and observation noise signals, which capture the uncertainty of the SS model, are unknown. These model parameters are the main ingredients for deriving the filtering and smoothing operators of the KS, and are encapsulated in the computation of the forward and backward Kalman gains (KGs). Thus, in RTSNet we circumvent the need to learn the noise distribution by designing it to learn the filtering and smoothing operations directly from data. Furthermore, as the model-based KS, which is comprised of a single forward-backward iteration, is not MMSE optimal for non-linear models, we design RTSNet to carry out multiple iterations via deep unfolding [43]. In the resulting architecture, the given (or approximated) evolution and observation models are plugged into the RTSNet smoother architecture. The forward and backward KGs in each iteration are replaced by dedicated compact RNNs. RTSNet thus converts a fixed number of KS iterations into a discriminative model [44] that is trained end-to-end.

In addition to the ability of the RTSNet to learn the smoothing task from data, its hybrid MB/DD architecture also retains the interpretability and other desired properties of the KS. By preserving the KS flow, RTSNet is invariant to the length of the sequence and to the initial conditions. In particular, RTSNet is shown to achieve the MMSE for linear models in the same way as the KS with full information, while only having access to partial information, and notably outperforms it with model mismatch. For non-linear SS models, RTSNet is shown to outperform MB variants of the KS, that are no longer optimal even with full domain knowledge. We also show that RTSNet outperforms leading DD smoothers, while using less trainable parameters, and being more efficient in terms of training and inference times. This improved performance follows from the ability of RTSNet to follow the principled KS operation, while circumventing its dependency on knowledge of the underlying noise statistics. In particular, by training the RTSNet smoother to directly compute the posterior distribution using learned KGs, we overcome the need to approximate the propagation of the noise statistics through the non-linearity. Moreover, doing so also bypasses the need for numerically costly matrix inversions and linearizations required in the KS equations.

The rest of this paper is organized as follows: Section II reviews the SS model and its associated tasks, and discusses relevant preliminaries. Section III details the discriminative architecture of RTSNet. Section IV presents the empirical study. Section V provides concluding remarks.

Throughout the paper, we use boldface lower-case letters for vectors and boldface upper-case letters for matrices. The transpose, ℓ_2 norm, and stochastic expectation are denoted by $\{\cdot\}^\top$, $\|\cdot\|$, and $\mathbb{E}[\cdot]$, respectively. The Gaussian distribution with mean μ and covariance $\mathbf{\Sigma}$ is denoted by $\mathcal{N}(\mu, \mathbf{\Sigma})$. Finally, $\mathbb{R}$ and $\mathbb{Z}$ are the sets of real and integer numbers, respectively.

II. System Model and Preliminaries

A. Problem Formulation

SS Models: Dynamical systems in discrete-time describe the relationship between a sequence of observations $\mathbf{y}_t$ and a sequence of unknown latent state variables $\mathbf{x}_t$, where $t \in \mathbb{Z}$ is the time index. SS models are a common characterization of dynamic systems [45], which in the (possibly) non-linear and Gaussian case, take the form

$$\mathbf{x}_t = \mathbf{f}\left(\mathbf{x}_{t-1}\right) + \mathbf{e}_t, \qquad \mathbf{e}_t \sim \mathcal{N}\left(\mathbf{0}, \mathbf{Q}\right), \quad \mathbf{x}_t \in \mathbb{R}^m, \tag{1a}$$

$$\mathbf{y}_t = \mathbf{h}\left(\mathbf{x}_t\right) + \mathbf{v}_t, \qquad \mathbf{v}_t \sim \mathcal{N}\left(\mathbf{0}, \mathbf{R}\right), \quad \mathbf{y}_t \in \mathbb{R}^n. \tag{1b}$$

In (1a), the state vector $\mathbf{x}_t$ evolves from the previous state $\mathbf{x}_{t-1}$, by a (possibly) non-linear, state-evolution function $\mathbf{f}(\cdot)$ and by an AWGN $\mathbf{e}_t$ with covariance matrix $\mathbf{Q}$. The observations $\mathbf{y}_t$ in (1b) are related to the current latent state vector by a (possibly) non-linear observation mapping $\mathbf{h}(\cdot)$ corrupted by AWGN $\mathbf{v}_t$ with covariance $\mathbf{R}$. A common special case of (1) is that of linear Gaussian SS models, where there exist matrices $\mathbf{F}, \mathbf{H}$ such that

$$\mathbf{f}(\mathbf{x}_{t-1}) = \mathbf{F} \cdot \mathbf{x}_{t-1}, \quad \mathbf{h}(\mathbf{x}_t) = \mathbf{H} \cdot \mathbf{x}_t. \tag{2}$$

Smoothing Task: SS models as in (1) are studied in the context of several different tasks, which can be roughly classified into two main categories: observation recovery and hidden state estimation. The first category deals with recovering parts of the observed signal $\mathbf{y}_t$. This can correspond, for example, to prediction and imputation. The second category lies at the core of the family of tracking problems, considering the estimation of $\mathbf{x}_t$. These include online (real-time) recovery, typically referred to as *filtering*, which is the task considered in [37], and *offline* estimation, i.e., *smoothing*, which is the main focus of this paper. More specifically, smoothing involves the joint computation the state estimates $\hat{\mathbf{x}}_{t|T}$ in a given timer horizon T mode, i.e., jointly estimating $\{\mathbf{x}_t\}$ for each $t \in \{1, 2, \ldots, T\} \triangleq \mathcal{T}$, given the corresponding block of noisy observations $\{\mathbf{y}_1, \mathbf{y}_1, \ldots, \mathbf{y}_T\}$.

Data-Aided Smoothing for Partially Known SS Models: In practice, the SS model parameters may be partially known, and one is likely to only have access to an approximated characterization of the underlying dynamics. We thus focus on such scenarios where the state-evolution function $\mathbf{f}(\cdot)$ and the state-observation function $\mathbf{h}(\cdot)$ can be reasonably approximated (possibly with mismatch) from our understating of the system dynamics and its physical design, or learned from data (as discussed in Subsection III-D). Regardless of how these functions are obtained, they can be used for smoothing. As opposed to the classical assumptions of the KS algorithms, the statistics of noises $\mathbf{e}_t$ and $\mathbf{v}_t$ are completely unknown, and may be non-Gaussian.

To deal with the partial modelling of the dynamics, we assume access to a labeled data set containing a sequence of observations and their corresponding states. Such data can be acquired, e.g., from field experiments, or using computationally intensive physically-compliant simulations [39]. The data set is comprised of N time sequence pairs, i.e., $\mathcal{D} = \left\{\left(\mathbf{Y}^{(i)}, \mathbf{X}^{(i)}\right)\right\}_{i=1}^{N}$, each of length T_i, namely,

$$\mathbf{Y}^{(i)} = \left[\mathbf{y}_1^{(i)}, \ldots, \mathbf{y}_{T_i}^{(i)}\right] \in \mathbb{R}^{n \times T_i} \tag{3}$$

are the noisy observations, and the corresponding states are

$$\mathbf{X}^{(i)} = \left[\mathbf{x}_0^{(i)}, \mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_{T_i}^{(i)}\right] \in \mathbb{R}^{m \times T_i + 1}. \tag{4}$$

Given $\mathcal{D}$ and the (approximated) $\mathbf{f}(\cdot), \mathbf{h}(\cdot)$, our objective is to design a smoothing function which maps the observations $\{\mathbf{y}_t\}_{t \in \mathcal{T}}$ into a state estimate $\{\hat{\mathbf{x}}_t\}_{t \in \mathcal{T}}$, where the accuracy of the smoother is evaluated as the mean-squared error (MSE) with respect to the true state $\{\mathbf{x}_t\}_{t \in \mathcal{T}}$.

B. Model-Based Kalman Smoothing

We next recall the MB RTS smoother [5], which is the basis for our proposed RTSNet, detailed in Section III. We describe the original algorithm for linear SS models, as in (2), and then discuss how to extend it for non-linear SS models.

The RTS smoother recovers the latent state variables using two linear recursive steps, referred to as the *forward* and *backward* passes. The forward pass is a standard KF, while the backward pass recursively computes corrections to the forward estimate, based on future observations.

Forward Pass: The KF produces a new estimate $\hat{\mathbf{x}}_{t|t}$ using its previous estimate $\hat{\mathbf{x}}_{t-1|t-1}$ and the observation $\mathbf{y}_t$. For each $t \in \mathcal{T}$, the KF operates in two steps: *prediction* and *update*.

The first step predicts the current *a priori* statistical moments based on the previous *a posteriori* moments. The moments of $\mathbf{x}_t$ are computed using the knowledge of the evolution matrix $\mathbf{F}$ as

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{F} \cdot \hat{\mathbf{x}}_{t-1|t-1}, \tag{5a}$$
$$\mathbf{\Sigma}_{t|t-1} = \mathbf{F} \cdot \mathbf{\Sigma}_{t-1|t-1} \cdot \mathbf{F}^\top + \mathbf{Q}, \tag{5b}$$

and the moments of the observations $\mathbf{y}_t$ are computed based on the knowledge of the observation matrix $\mathbf{H}$ as

$$\hat{\mathbf{y}}_{t|t-1} = \mathbf{H} \cdot \hat{\mathbf{x}}_{t|t-1}, \tag{6a}$$
$$\mathbf{S}_{t|t-1} = \mathbf{H} \cdot \mathbf{\Sigma}_{t|t-1} \cdot \mathbf{H}^\top + \mathbf{R}. \tag{6b}$$

In the update step, the *a posteriori* state moments are computed based on the *a priori* moments as

$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathcal{K}_t \cdot \Delta\mathbf{y}_t, \tag{7a}$$
$$\mathbf{\Sigma}_{t|t} = \mathbf{\Sigma}_{t|t-1} - \mathcal{K}_t \cdot \mathbf{S}_{t|t-1} \cdot \mathcal{K}_t^\top. \tag{7b}$$

Here, $\mathcal{K}_t$ is the KG, and it is given by

$$\mathcal{K}_t = \mathbf{\Sigma}_{t|t-1} \cdot \mathbf{H}^\top \cdot \mathbf{S}_{t|t-1}^{-1}, \tag{8}$$

while $\Delta\mathbf{y}_t = \mathbf{y}_t - \hat{\mathbf{y}}_{t|t-1}$ is the innovation, and is the only term that depends on the observed data.

Backward pass: The backward pass is similar in its structure to the update step in the KF. For each $t \in \{T-1, \ldots, 1\}$, the forward belief is corrected with future estimates via

$$\hat{\mathbf{x}}_{t|T} = \hat{\mathbf{x}}_{t|t} + \mathcal{G}_t \cdot \overleftarrow{\Delta}\mathbf{x}_{t+1}, \tag{9a}$$
$$\mathbf{\Sigma}_{t|T} = \mathbf{\Sigma}_{t|t} - \mathcal{G}_t \cdot \Delta\mathbf{\Sigma}_{t+1|T} \cdot \mathcal{G}_t^\top. \tag{9b}$$

Here, $\mathcal{G}_t$ is the backward KG, computed based on second-order statistical moments from the forward pass as

$$\mathcal{G}_t = \mathbf{\Sigma}_{t|t} \cdot \mathbf{F}^\top \cdot \mathbf{\Sigma}_{t+1|t}^{-1}. \tag{10}$$

The difference terms are given by $\overleftarrow{\Delta}\mathbf{x}_{t+1} = \hat{\mathbf{x}}_{t+1|T} - \hat{\mathbf{x}}_{t+1|t}$ and $\Delta\mathbf{\Sigma}_{t+1} = \mathbf{\Sigma}_{t+1} - \mathbf{\Sigma}_{t+1|t}$. The KS is MMSE optimal for linear Gaussian SS models.

Extension to Non-Linear Dynamics: For non-linear SS models as in (1), the first-order statistical moments (5a) and (6a) are replaced with

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{f}\left(\hat{\mathbf{x}}_{t-1}\right), \tag{11a}$$
$$\hat{\mathbf{y}}_{t|t-1} = \mathbf{h}\left(\hat{\mathbf{x}}_{t|t-1}\right), \tag{11b}$$

respectively. Unfortunately, the second-order moments cannot be propagated directly through the non-linearity, and thus must be approximated, resulting in methods that no longer share the MSE optimilaity achieved in linear models.

Among the methods proposed to approximate the second-order moments are the unscented RTS, that is based on unscented transformations [46], and PSs which use sequential sampling [9]. Arguably the most common non-linear smoother is the EKS, which uses straight forward linearization. Specifically, when $\mathbf{f}(\cdot)$ and $\mathbf{h}(\cdot)$ are differentiable, the EKS linearizes them in a time-dependent manner. This is done using their partial derivative matrices (Jacobians), evaluated at $\hat{\mathbf{x}}_{t-1|t-1}$ and $\hat{\mathbf{x}}_{t|t-1}$, namely,

$$\hat{\mathbf{F}}_t = \mathcal{J}_f\left(\hat{\mathbf{x}}_{t-1|t-1}\right), \tag{12a}$$

$$\hat{\mathbf{H}}_t = \mathcal{J}_h\left(\hat{\mathbf{x}}_{t|t-1}\right). \tag{12b}$$

The Jabobians in (12) are then substituted into equations (5b), (10), (6b), and (8) of the KS.

The forward and backward KGs are pivot terms, that are used as tuning factors for updating our current belief, and they depend on the second-order moments. For linear SS models, the covarince computation is purely MB, i.e., based solely on the noise statistics, while for non-linear systems the covariance depends on the specific trajectory. Furthermore, these covariance computations require full knowledge of the underlying model, and performance notably degrades in the presence of model mismatch. This motivates the derivation of a data-aided smoothing algorithm that estimates the KGs directly as a form of discriminative learning [44], and by that circumvents the need to estimate the second-order moments.

III. RTSNet

Here, we present RTSNet. We begin by describing our design rationale and high level architecture in Subsection III-A, after which we detail the micro architecture in Subsection III-B. We then describe the training procedure in Subsection III-C, and provide a discussion in Subsection III-D.

A. High Level Design

Rationale: The basic design idea behind the proposed RTSNet is to utilize the skeleton of the MB RTS smoother, hence the name RTSNet, and to replace modules depending on unavailable domain knowledge, with trainable DNNs. By doing so, we convert the KS into a discriminative algorithm, that can be trained in a supervised end-to-end manner.

Our design is based on two main guiding properties:

P1 The RTS operation, comprised of a single forward-backward pass, is not necessarily MMSE optimal when the SS model is not linear Gaussian.

P2 The RTS smoother requires the missing domain knowledge (i.e., the noise statistics) and linearization operations solely for computing the KGs in (8) and (10).

Unfolded Architecture (by *P1*): Property *P1* indicates that one can possibly improve performance by carrying out multiple forward-backward passes, iteratively refining the estimated sequence. Following the deep unfolding methodology [39], we design RTSNet to carry out K forward-backward passes, for some fixed $K \geq 1$.

Each pass of index $k \in \{1,\ldots,K\}$ involves a single forward-backward smoothing. The input and output of this pass are the sequences $\mathbf{Y}_k = [\mathbf{y}_{k,1}, \mathbf{y}_{k,2}, \ldots, \mathbf{y}_{k,T}]$ and $\hat{\mathbf{X}}_k = \left[\hat{\mathbf{x}}_{k,1|T}, \hat{\mathbf{x}}_{k,2|T}, \ldots \hat{\mathbf{x}}_{k,T|T}\right]$, respectively, and its operation is based on an SS model of the form

$$\mathbf{x}_{k,t} = \mathbf{f}\left(\mathbf{x}_{k,t-1}\right) + \mathbf{e}_{k,t}, \qquad \mathbf{x}_{k,t} \in \mathbb{R}^m, \tag{13a}$$

$$\mathbf{y}_{k,t} = \mathbf{h}_k\left(\mathbf{x}_{k,t}\right) + \mathbf{v}_{k,t}, \qquad \mathbf{y}_{k,t} \in \mathbb{R}^{n_k}. \tag{13b}$$

For the first pass, the inputs are $\mathbf{y}_{1,t} = \mathbf{y}_t$, i.e., the observations, and thus $\mathbf{h}_1(\cdot) \equiv \mathbf{h}(\cdot)$ and $n_1 = n$ in (13b). For the following passes where $k > 1$, the input is the estimate produced by the subsequent pass, i.e., $\mathbf{y}_{k,t} = \hat{\mathbf{x}}_{k-1,t|T}$. This input is treated as noisy state observations, and thus $\mathbf{h}_k(\cdot)$ is the identity mapping and $n_k = m$. The noise signals in (13) obey an unknown distribution, following the problem formulation in Subsection II-A.

Deep Augmenting RTS (by *P2*): We choose the RTS smoother as our MB backbone based on *P2*. Specifically, as opposed to other alternatives, e.g., MBF [47], [48] and BIFM [49], in RTS all the unknown domain knowledge is encapsulated in the forward and backward KGs, $\boldsymbol{\mathcal{K}}_t$, and $\boldsymbol{\mathcal{G}}_t$, respectively. Consequently, for each pass k, we employ an RTS smoother, while replacing the KGs computation with DNNs.

Since both KGs involve tracking time-evolving second-order moments, they are replaced by RNNs in each pass of RTSNet, with input features encapsulating the missing statistics. The resulting operation of the kth pass commences with a forward pass, that is based upon KalmanNet [37]: For each t from 1 to T a *prediction* and *update* steps are applied. In the *prediction* step, we the prior estimates for the current state and for the current observation using (11), namely,

$$\hat{\mathbf{x}}_{k,t|t-1} = \mathbf{g}(\hat{\mathbf{x}}_{k,t-1}), \quad \hat{\mathbf{y}}_{k,t|t-1} = \mathbf{h}_k(\hat{\mathbf{x}}_{k,t|t-1}).$$

In the *update* step, we compute $\hat{\mathbf{x}}_{k,t|k,t}$, the current forward posterior, using (7a), i.e.,

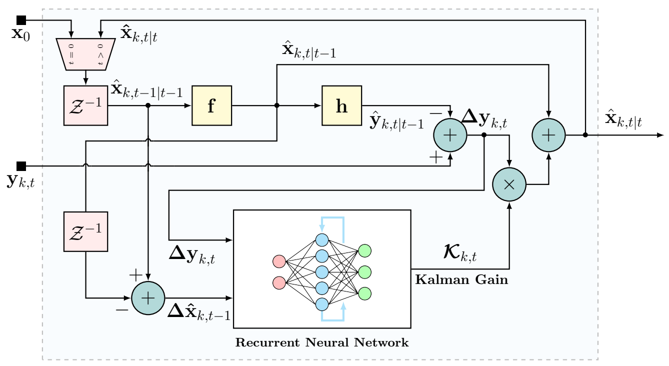
$$\hat{\mathbf{x}}_{k,t|t} = \hat{\mathbf{x}}_{k,t|t-1} + \boldsymbol{\mathcal{K}}_{k,t} \cdot \left(\mathbf{y}_{k,t} - \hat{\mathbf{y}}_{k,t|t-1}\right).$$

As opposed to the KS, here the filtering (forward) KG $\boldsymbol{\mathcal{K}}_{k,t}$ is computed using an RNN.

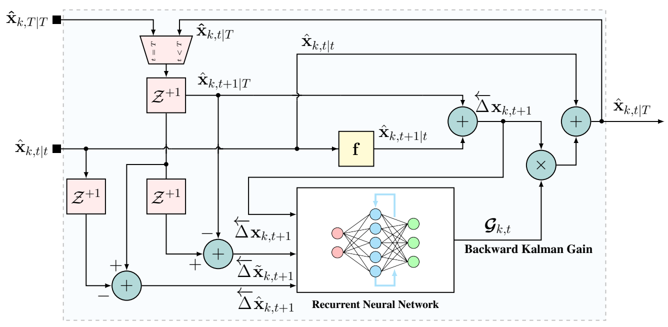
The forward pass if followed by a backward pass, which updates our state estimates using information from future estimates. As in (9a), this procedure is given by

$$\hat{\mathbf{x}}_{k,t|T} = \hat{\mathbf{x}}_{k,t|t} + \boldsymbol{\mathcal{G}}_{k,t} \cdot \left(\hat{\mathbf{x}}_{k,t+1|T} - \hat{\mathbf{x}}_{k,t+1|t}\right).$$

where the resulting estimate is $\hat{\mathbf{x}}_{k,t} = \hat{\mathbf{x}}_{k,t|T}$ for each t. As in the forward pass, the smoothing (backward) KG $\boldsymbol{\mathcal{G}}_t$ is computed using an RNN. The high-level architecture of RTSNet is depicted in Fig. 1.

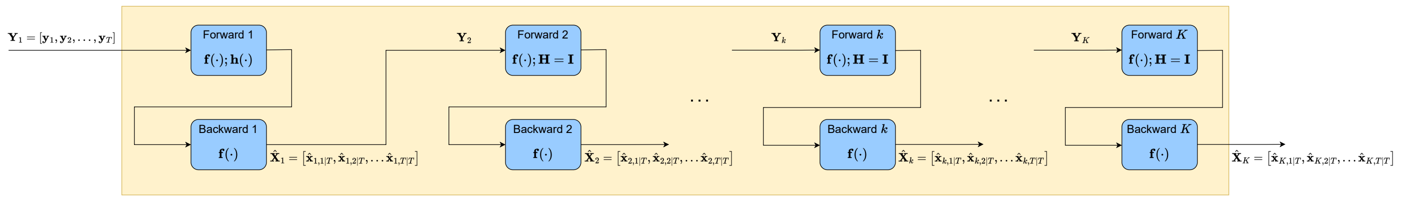


(a) k^{th} forward pass.



$\hat{\mathbf{x}}_{k,T|T}$
$\hat{\mathbf{x}}_{k,t|T}$
$\hat{\mathbf{x}}_{k,t|t}$
$\hat{\mathbf{x}}_{k,t+1|T}$
$\mathcal{Z}^{+1}$
f
$\hat{\mathbf{x}}_{k,t+1|t}$
$\overleftarrow{\Delta}\mathbf{x}_{k,t+1}$
$\hat{\mathbf{x}}_{k,t|T}$
$\overleftarrow{\Delta}\mathbf{x}_{k,t+1}$
$\overleftarrow{\Delta}\tilde{\mathbf{x}}_{k,t+1}$
$\overleftarrow{\Delta}\hat{\mathbf{x}}_{k,t+1}$
$\mathcal{G}_{k,t}$
Backward Kalman Gain
Recurrent Neural Network

(b) k^{th} backward pass.



$\mathbf{Y}_1 = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T]$
Forward 1
$\mathbf{f}(\cdot); \mathbf{h}(\cdot)$
Backward 1
$\mathbf{f}(\cdot)$
$\hat{\mathbf{X}}_1 = [\hat{\mathbf{x}}_{1,1|T}, \hat{\mathbf{x}}_{1,2|T}, \ldots \hat{\mathbf{x}}_{1,T|T}]$
$\mathbf{Y}_2$
Forward 2
$\mathbf{f}(\cdot); \mathbf{H} = \mathbf{I}$
Backward 2
$\mathbf{f}(\cdot)$
$\hat{\mathbf{X}}_2 = [\hat{\mathbf{x}}_{2,1|T}, \hat{\mathbf{x}}_{2,2|T}, \ldots \hat{\mathbf{x}}_{2,T|T}]$
$\mathbf{Y}_k$
Forward k
$\mathbf{f}(\cdot); \mathbf{H} = \mathbf{I}$
Backward k
$\mathbf{f}(\cdot)$
$\hat{\mathbf{X}}_k = [\hat{\mathbf{x}}_{k,1|T}, \hat{\mathbf{x}}_{k,2|T}, \ldots \hat{\mathbf{x}}_{k,T|T}]$
$\mathbf{Y}_K$
Forward K
$\mathbf{f}(\cdot); \mathbf{H} = \mathbf{I}$
Backward K
$\mathbf{f}(\cdot)$
$\hat{\mathbf{X}}_K = [\hat{\mathbf{x}}_{K,1|T}, \hat{\mathbf{x}}_{K,2|T}, \ldots \hat{\mathbf{x}}_{K,T|T}]$

Fig. 1: RTSNet high level architecture block diagram.

B. Micro Architecture

RTSNet includes $2K$ RNNs, as each kth pass utilizes one RNN to compute the forward KG and another RNN to compute the backward KG. In this subsection, we focus on a single pass of index k and formulate the micro architecture its forward and backward RNNs, as well as which features the RNNs use to compute the KGs.

Forward Gain: The forward pass is built on KalmanNet, where architecture 2 of [37] is particularly utilized to compute the forward KG, i.e., $\mathcal{K}_{k,t}$, using separate gated recurrent unit (GRU) cells for each of the tracked second-order statistical moments. The division of the architecture into separate GRU cells and fully connected (FC) layers and their interconnection is illustrated in Fig. 2. As shown in the figure, the network composes three GRU layers, connected in a cascade with dedicated input and output FC layers. This architecture, which is composed of a non-standard interconnection between GRUs and FC layers, is directly tailored towards the formulation of the SS model and the operation of the MB KF, as detailed in [37].

The input features are designed to capture differences in the state and the observation model, as these differences are mostly affected by unknown noise statistics. As in [37], the following features are used to compute $\mathcal{K}_{k,t}$ (see Fig. 2):

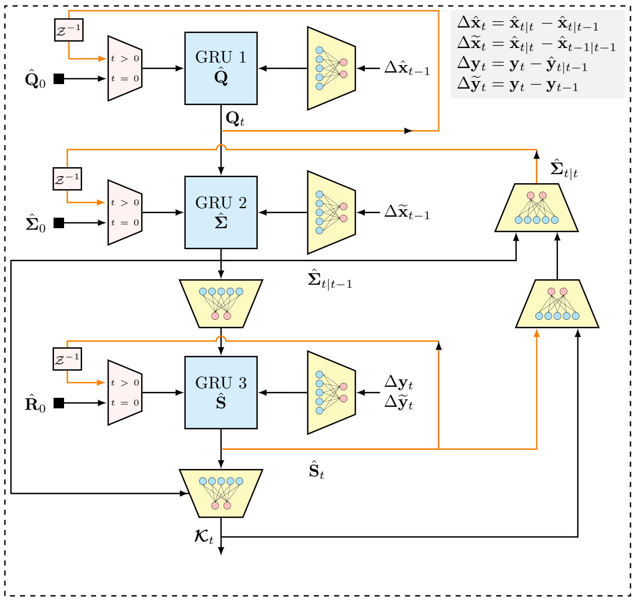
F1 Observation difference $\Delta\tilde{\mathbf{y}}_{k,t} = \mathbf{y}_{k,t} - \mathbf{y}_{k,t-1}$.

F2 Innovation difference $\Delta\mathbf{y}_{k,t} = \mathbf{y}_{k,t} - \hat{\mathbf{y}}_{k,t|t-1}$.

F3 Forward evolution difference $\Delta\tilde{\mathbf{x}}_{k,t} = \hat{\mathbf{x}}_{k,t|t} - \hat{\mathbf{x}}_{k,t-1|t-1}$.

F4 Forward update difference $\Delta\hat{\mathbf{x}}_{k,t} = \hat{\mathbf{x}}_{k,t|t} - \hat{\mathbf{x}}_{k,t|t-1}$.

Backward Gain: To compute $\mathcal{G}_{k,t}$ in a learned manner, we again design an architecture based on how the KS computes the backward gain. To that aim, we again use separate GRU cells for each of the tracked second-order statistical moments, as illustrated in Fig. 3. The first GRU layer tracks the unknown state noise covariance $\mathbf{Q}$, thus tracking m^2 variables. Simi-



$\Delta\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t|t} - \hat{\mathbf{x}}_{t|t-1}$
$\Delta\tilde{\mathbf{x}}_t = \hat{\mathbf{x}}_{t|t} - \hat{\mathbf{x}}_{t-1|t-1}$
$\Delta\mathbf{y}_t = \mathbf{y}_t - \hat{\mathbf{y}}_{t|t-1}$
$\Delta\tilde{\mathbf{y}}_t = \mathbf{y}_t - \mathbf{y}_{t-1}$
$\mathcal{Z}^{-1}$
$\hat{\mathbf{Q}}_0$
$t>0$
$t=0$
GRU 1 $\hat{\mathbf{Q}}$
$\Delta\hat{\mathbf{x}}_{t-1}$
$\mathbf{Q}_t$
$\hat{\mathbf{\Sigma}}_{t|t}$
$\hat{\mathbf{\Sigma}}_0$
GRU 2 $\hat{\mathbf{\Sigma}}$
$\Delta\tilde{\mathbf{x}}_{t-1}$
$\hat{\mathbf{\Sigma}}_{t|t-1}$
$\hat{\mathbf{R}}_0$
GRU 3 $\hat{\mathbf{S}}$
$\Delta\mathbf{y}_t$
$\Delta\tilde{\mathbf{y}}_t$
$\hat{\mathbf{S}}_t$
$\mathcal{K}_t$

Fig. 2: Forward gain RNN block diagram. The input features are used to update three GRUs with dedicated FC layers, and the overall interconnection between the blocks is based on the flow of the Forward KG $\mathcal{K}_t$ computation in the MB KF.

larly, the second GRUs tracks the predicted moment $\hat{\mathbf{\Sigma}}_{t|T}$ (9b) and $\hat{\mathbf{S}}_t$ (6b), thus having m^2 hidden state variables. The GRUs are interconnected such that the learned $\mathbf{Q}$ is used to compute $\hat{\mathbf{\Sigma}}_{t|T}$, while FC layers are utilized for input and output shaping.

We utilize the following features, which are related to the unknown underlying statistics:

B1 Update difference $\overleftarrow{\Delta}\mathbf{x}_{k,t+1} = \hat{\mathbf{x}}_{k,t+1|T} - \hat{\mathbf{x}}_{k,t+1|t}$.

B2 Backward forward difference $\overleftarrow{\Delta}\hat{\mathbf{x}}_{k,t+1} = \hat{\mathbf{x}}_{k,t+1|T} - \hat{\mathbf{x}}_{k,t+1|t+1}$.

B3 Evolution difference $\overleftarrow{\Delta}\tilde{\mathbf{x}}_{k,t+1} = \hat{\mathbf{x}}_{k,t+2|T} - \hat{\mathbf{x}}_{k,t+1|T}$.

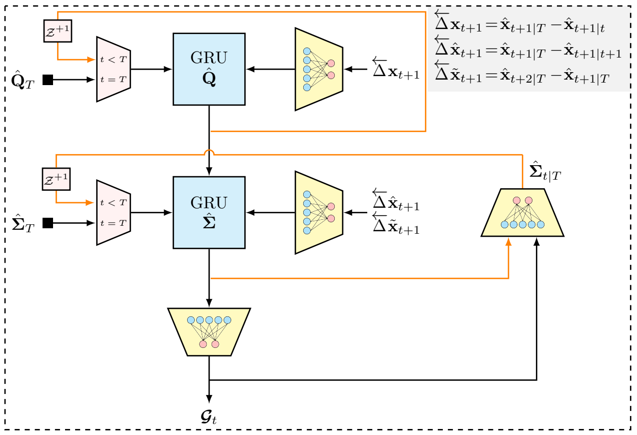

$\overleftarrow{\Delta}\mathbf{x}_{t+1} = \hat{\mathbf{x}}_{t+1|T} - \hat{\mathbf{x}}_{t+1|t}$
$\overleftarrow{\Delta}\hat{\mathbf{x}}_{t+1} = \hat{\mathbf{x}}_{t+1|T} - \hat{\mathbf{x}}_{t+1|t+1}$
$\overleftarrow{\Delta}\tilde{\mathbf{x}}_{t+1} = \hat{\mathbf{x}}_{t+2|T} - \hat{\mathbf{x}}_{t+1|T}$
$\mathcal{Z}^{+1}$
$t < T$
$t = T$
$\hat{\mathbf{Q}}_T$
GRU $\hat{\mathbf{Q}}$
$\overleftarrow{\Delta}\mathbf{x}_{t+1}$
$\mathcal{Z}^{+1}$
$t < T$
$t = T$
$\hat{\mathbf{\Sigma}}_T$
GRU $\hat{\mathbf{\Sigma}}$
$\overleftarrow{\Delta}\hat{\mathbf{x}}_{t+1}$
$\overleftarrow{\Delta}\tilde{\mathbf{x}}_{t+1}$
$\hat{\mathbf{\Sigma}}_{t|T}$
$\boldsymbol{\mathcal{G}}_t$

Fig. 3: Backward gain RNN block diagram. The input features are used to update two GRUs with dedicated FC layers, and the overall interconnection between the blocks is based on the flow of the Backward KG $\boldsymbol{\mathcal{G}}_t$ computation in the KS.

The first two features capture the uncertainty in the state estimate, where the differences remove predictable components such that they are mostly affected by the unknown noise statistics. The third feature is related to the evolution of the predicted state, and thus reflects on its statistics that are tracked by the KS. The features are utilized by the proposed architecture, as illustrated in Fig. 3.

C. Training Algorithm

Loss Function: We train RTSNet in a supervised manner. To formulate the loss function used for training, let $\mathbf{\Theta}_k$ denote the trainable parameters of the RNNs of the kth pass. The loss measure used to tune these parameters is the regularized ℓ_2 loss; for a labeled pair $(\mathbf{Y}_k(i), \mathbf{X}^{(i)})$ of length T_i, this loss is given by

$$\mathcal{L}_k^{(i)}(\mathbf{\Theta}_k) = \frac{1}{T_i} \cdot \sum_{t=1}^{T_i} \left\| \hat{\mathbf{x}}_{k,t|T}\left(\mathbf{Y}_k^{(i)}, \mathbf{\Theta}_k\right) - \mathbf{x}_t^{(i)} \right\|^2 + \gamma_k \cdot \left\|\mathbf{\Theta}_k\right\|^2. \tag{14}$$

In (14), $\hat{\mathbf{x}}_{k,t|T}(\mathbf{Y}_k, \mathbf{\Theta}_k)$ denotes the tth output of the kth pass with input $\mathbf{Y}_k$ and RNN parameters $\mathbf{\Theta}_k$. While the loss in (14) refers to a single trajectory i, we use it to optimize RTSNet using variants of mini-batch stochastic gradient descent. Here for every batch indexed by j, we choose $B < N$ trajectories indexed by $i_1^j, \ldots, i_B^j$, and compute the mini-batch loss of the kth pass as

$$\bar{\mathcal{L}}_{k,j}(\mathbf{\Theta}_k) = \frac{1}{B} \sum_{b=1}^{B} \mathcal{L}_k^{(i_b^j)}(\mathbf{\Theta}_k). \tag{15}$$

Gradient Computation: RTSNet uses the RNNs for computing the KGs rather than for directly producing the estimate $\hat{\mathbf{x}}_{k,t|T}$. The loss function (14) enables to evaluate the overall system without having to externally provide ground truth values of the KGs for training purposes. Training RTSNet in end-to-end manner thus builds upon the ability to backpropagate the loss to the computation of the KGs. One can obtain the loss gradient with respect to the KGs from the output of RTSNet since by combing (7a) and (9a) we get that

$$\hat{\mathbf{x}}_{k,t|T} = \hat{\mathbf{x}}_{k,t|t-1} + \boldsymbol{\mathcal{K}}_{k,t} \cdot \Delta\mathbf{y}_{k,t} + \boldsymbol{\mathcal{G}}_{k,t} \cdot \overleftarrow{\Delta}\mathbf{x}_{k,t+1}. \tag{16}$$

Consequently, the gradients of the ℓ_2 loss terms with respect to the KGs obey

$$\frac{\partial}{\partial \boldsymbol{\mathcal{K}}_{k,t}} \left\|\hat{\mathbf{x}}_{k,t|T} - \mathbf{x}_t\right\|^2 \propto \left(\mathbf{x}_{k,t|T} - \mathbf{x}_t\right) \cdot \Delta\mathbf{y}_{k,t}^\top \tag{17a}$$

$$\frac{\partial}{\partial \boldsymbol{\mathcal{G}}_{k,t}} \left\|\hat{\mathbf{x}}_{k,t|T} - \mathbf{x}_t\right\|^2 \propto \left(\mathbf{x}_{k,t|T} - \mathbf{x}_t\right) \cdot \overleftarrow{\Delta}\mathbf{x}_{k,t+1}^\top, \tag{17b}$$

which in turn allows to compute the gradient of the ℓ_2 loss with respect to $\mathbf{\Theta}_k$ via the chain rule. The gradient computation indicates that one can learn the computation of the KGs by training RTSNet end-to-end.

End-to-End Training: The differentiable loss function in (14) allows end-to-end training of a single forward-backward pass of index k. To train the overall unfolded RTSNet, we consider the following loss measures:

Joint learning, where the RNNs of all the passes are simultaneously using the labeled dataset $\mathcal{D}$. Here, we stack the trainable parameters as $\mathbf{\Theta} = \{\mathbf{\Theta}_k\}$, and set the loss function for the ith trajectory to

$$\mathcal{L}^{(i)}(\mathbf{\Theta}) = \sum_{k=1}^{K} \alpha_k \mathcal{L}_k^{(i)}(\mathbf{\Theta}_k), \tag{18}$$

where each $\mathcal{L}_k^{(i)}(\cdot)$ is evaluated via (14) with $\mathbf{Y}_1^{(i)} = \mathbf{Y}^{(i)}$ and $\mathbf{Y}_k^{(i)} = \hat{X}_{k-1}^{(i)}$ for $k > 1$. The coefficients $\{\alpha_k\}_{k=1}^K$ in (18) balance the contribution of each pass to the loss – setting $\alpha_k = 0$ for $k < K$ evaluates RTSNet based solely on its output, while setting $\alpha_k \neq 0$ for $k < K$ encourages also the intermediate passes to provide accurate estimates. The latter exploits the fact that the interpretable architecture of RTSNet allows to associate its internal features with operations meanings in order to facilitate training; a candidate setting is $\alpha_k = \log(1 + k)$, see, e.g., [50].

Sequential learning repeats the training procedure K times, training each pass of index k after its preceding passes have been trained using the same dataset. Here, the dataset $\mathcal{D}$ is first used to train only $\mathbf{\Theta}_1$ using (14) with $k = 1$; then, $\mathbf{\Theta}_1$ is frozen and $\mathbf{\Theta}_2$ is trained using (14) with $k = 2$ and $\mathbf{Y}_2^{(i)} = \hat{X}_1^{(i)}$, and the procedure repeats until $\mathbf{\Theta}_K$ is trained. This form of training, proposed in [51], exploits the modular structure of the unfolded architecture and tends to be data efficient and simpler to train compared with joint learning, and is also the form of training used in our empirical study presented in Section IV.

D. Discussion

RTSNet is designed to operate in a hybrid DD/MB manner, combining deep learning with the classical KS. By identifying the specific noise-model-dependent computations of the KS and replacing them with a dedicated RNNs integrated in the MB flow, RTSNet benefits from the individual strengths of both DD and MB approaches. We particularly note several core differences between RTSNet and its MB counterpart. Unlike the KS, RTSNet does not attempt to linearize the SS model, and does not impose a statistical model on the noise signals, while avoiding the need to compute a Jacobian and invert a matrix at each iteration. In addition, RTSNet filters in a non-linear manner, as, e.g., its forward KG matrix depends on the input $\mathbf{y}_t$. Moreover, RTSNet supports multiple learned forward-backward passes in order to improve accuracy in non-linear settings where the single pass KS is not optimal. Due to these differences, RTSNet is more robust to model mismatch and can infer more efficiently compared with the KS, as shown in Section IV.

Compared to purely DD state estimation, RTSNet benefits from its model awareness, as it supports systematic inclusion of the available state evolution and observation functions, and does not have to learn its complete operation from data. As empirically observed in Section IV, RTSNet achieves improved MSE compared to utilizing RNNs for end-to-end state estimation, and also approaches the MMSE performance achieved by the KS in linear Gaussian SS models.

Furthermore, the operation of KalmanNet follows the flow of KS rather than being utilized as a black-box. This implies that the intermediate features exchanged between its modules have a specific operation meaning, providing interpretability that is often scarce in end-to-end, deep learning systems. Finally, the fact that RTSNet learns to compute the KGs indicates the possibility of providing not only estimates of the state $\mathbf{x}_t$, but also a measure of confidence in this estimate, as the KGs can be related to the covariance of the estimate, as initially explored for KalmanNet in [41].

While we train RTSNet in a supervised manner using labeled data, the fact that it preserves the operation of the KS indicates the possibility of training it in an *unsupervised* manner using a loss function measuring consistency between its estimates and observations, e.g., $\left\|\mathbf{h}\left(\hat{\mathbf{x}}_{t|T}\right)-\mathbf{y}_t\right\|^2$. One can thus envision RTSNet being trained offline in a supervised manner, while tracking variations in the underlying SS model at run-time by online self supervision, as was initially explored for KalmanNet in [42] and we leave its extension to future work. The ability to train in an unsupervised manner opens the door to use the backbone of RTSNet in more SS related tasks other than smoothing, e.g., signal de-noising, imputation, and prediction. Nonetheless, we leave the exploration of these extensions of RTSNet for future work.

IV. Experiments and Results

In this section we present an extensive empirical study of RTSNet[1], evaluating its performance in multiple setups and comparing it with both MB and DD benchmark algorithms. We consider both linear Gaussian SS models, where we identify the ability of RTSNet to coincide with the KS (which is MSE optimal in such settings), as well as challenging non-linear models.

A. Experimental Setup

Smoothers: Our empirical study compares RTSNet with MB and DD counterparts. We use the KS (RTS smoother) as the MB benchmark for linear models. For non-linear models, we use the EKS and the PS [52], where the latter is based on forward-filter backward simulator of [53] with 100 particles and 10 backward trajectories. The benchmark algorithms were optimized for performance by manually tuning the covariance matrices. This tuning is often essential to avoid divergence under model uncertainty as well as under dominant non-linearities.

[1]The source code used in our empirical study along with the complete set of hyperparameters can be found at https://github.com/KalmanNet/RTSNet_TSP.

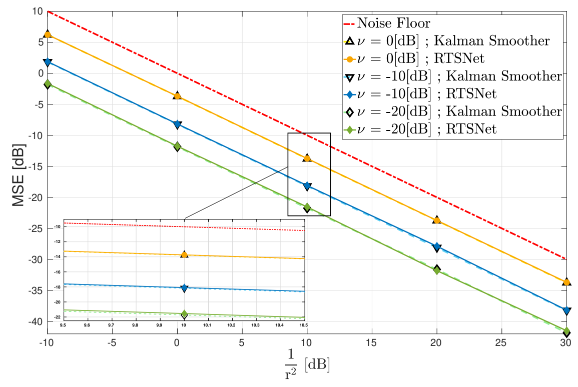


Fig. 4: Linear SS model - full information.

Our main DD benchmark is the hybrid graph neural network-aided belief propagation smoother of [36] (referred to as GNN-BP), which incorporates knowledge of the SS model. We also compare with a black-box architecture using a Bi-directional RNN (BRNN), which is comprised of bi-directional GRU with input and output FC layers designed to have a similar number of trainable parameters as RTSNet. Both DNN-aided benchmarks are empirically optimized and cross validated to achieve their best training performance.

We use the term *full information* to describe cases where $\mathbf{f}(\cdot)$ and $\mathbf{h}(\cdot)$ are accurately known. The term *partial information* refers to the case where RTSNet and benchmark algorithms operate with some level of model mismatch in their available knowledge of the SS model parameters. RTSNet and the DD BRNN operate without access to the noise covariance matrices (i.e., $\mathbf{Q}$ and $\mathbf{R}$), while their MB counterparts operate with an accurate knowledge of the noise covariance matrices from which the data was generated.

Evaluation: The metric used to evaluate the performance is the empirical mean and standard deviation of squared error, denoted,$\hat{\mu}$ and $\hat{\sigma}$, respectively. Unless stated otherwise, we evaluate using a $N_{\text{test}} = 200$ test trajectories.

Throughout the empirical study and unless stated otherwise, in the experiments involving synthetic data, the SS model is generated using diagonal noise covariance matrices; i.e.,

$$\mathbf{Q} = \mathrm{q}^2 \cdot \mathbf{I}, \quad \mathbf{R} = \mathrm{r}^2 \cdot \mathbf{I}, \quad \nu \triangleq \frac{\mathrm{q}^2}{\mathrm{r}^2}. \tag{19}$$

By (19), setting ν to be 0 dB implies that both the state noise and the observation noise have the same variance.

B. Linear State Space Models with Full Information

We first focus on comparing RTSNet with the KS for synthetically generated linear Gaussian dynamics with full information. Since the KS is MSE optimal here, we show that the

TABLE I: Linear SS model - Full Information - MSE [dB]

(a) $\nu = 0\,[\text{dB}]$

r^2 [dB]	10	0	-10	-20	-30
Noise	10.023 $\pm$ 0.424	0.054 $\pm$ 0.448	-10.003 $\pm$ 0.427	-19.947 $\pm$ 0.411	-29.962 $\pm$ 0.430
KF	8.085 $\pm$ 0.502	-1.827 $\pm$ 0.525	-11.880 $\pm$ 0.464	-21.903 $\pm$ 0.419	-31.886 $\pm$ 0.514
KS	6.215 $\pm$ 0.487	-3.710 $\pm$ 0.535	-13.776 $\pm$ 0.466	-23.751 $\pm$ 0.519	-33.749 $\pm$ 0.508
RTSNet	6.225 $\pm$ 0.487	-3.695 $\pm$ 0.537	-13.738 $\pm$ 0.463	-23.732 $\pm$ 0.512	-33.698 $\pm$ 0.506

(b) $\nu = -10\,[\text{dB}]$

r^2 [dB]	10	0	-10	-20	-30
Noise	10.004 $\pm$ 0.419	0.000 $\pm$ 0.392	-10.029 $\pm$ 0.431	-19.982 $\pm$ 0.404	-29.969 $\pm$ 0.435
KF	5.299 $\pm$ 0.710	-4.703 $\pm$ 0.663	-14.756 $\pm$ 0.675	-24.680 $\pm$ 0.596	-34.731 $\pm$ 0.696
KS	1.834 $\pm$ 0.794	-8.220 $\pm$ 0.721	-18.179 $\pm$ 0.778	-28.098 $\pm$ 0.726	-38.236 $\pm$ 0.837
RTSNet	1.881 $\pm$ 0.796	-8.169 $\pm$ 0.720	-18.092 $\pm$ 0.797	-27.875 $\pm$ 0.746	-38.183 $\pm$ 0.836

(c) $\nu = -20\,[\text{dB}]$

r^2 [dB]	10	0	-10	-20	-30
Noise	10.012 $\pm$ 0.398	-0.008 $\pm$ 0.434	-10.004 $\pm$ 0.448	-19.977 $\pm$ 0.416	-30.003 $\pm$ 0.429
KF	2.756 $\pm$ 1.086	-7.254 $\pm$ 1.012	-17.339 $\pm$ 0.967	-27.194 $\pm$ 1.011	-37.324 $\pm$ 0.963
KS	-1.790 $\pm$ 1.242	-11.847 $\pm$ 1.190	-21.738 $\pm$ 1.281	-31.620 $\pm$ 1.107	-41.831 $\pm$ 1.289
RTSNet	-1.640 $\pm$ 1.199	-11.712 $\pm$ 1.173	-21.543 $\pm$ 1.279	-31.817 $\pm$ 1.152	-41.505 $\pm$ 1.229

(a) Scaling SS Model Dimensions

Dimensions	2×2	5×5	10×10	20×20
KF	-7.791 $\pm$ 2.204	-10.931 $\pm$ 1.411	-11.062 $\pm$ 0.951	-11.540 $\pm$ 0.771
KS	-11.732 $\pm$ 2.489	-12.350 $\pm$ 1.561	-12.436 $\pm$ 1.058	-12.762 $\pm$ 0.808
BRNN	3.289 $\pm$ 4.495	4.261 $\pm$ 4.724	5.581 $\pm$ 4.553	3.742 $\pm$ 4.416
GNN-BP	-10.016 $\pm$ 2.331	-9.028 $\pm$ 1.312	-8.674 $\pm$ 0.803	-8.557 $\pm$ 0.603
RTSNet	-11.208 $\pm$ 2.438	-11.9725 $\pm$ 1.597	-12.0231 $\pm$ 1.055	-12.2755 $\pm$ 0.828

(b) Scalability for Trajectory Length

$T_{\text{training}}, T_{\text{testing}}$	$20, 20$	$100, 100$	$100, 1000$	$100, \mathcal{U}\,[100, 1000]$
Noise	0.025 $\pm$0.919	-0.008 $\pm$ 0.434	0.011 $\pm$ 0.132	0.015 $\pm$ 0.227
KF	-7.791 $\pm$ 2.204	-7.254 $\pm$ 1.012	-7.162 $\pm$ 0.335	-7.241 $\pm$ 0.543
KS	-11.732 $\pm$ 2.489	-11.847 $\pm$ 1.190	-11.810 $\pm$ 0.450	-11.853 $\pm$ 0.687
BRNN	3.289 $\pm$ 4.495	22.277 $\pm$ 5.190	54.955 4.419	48.390 5.436
GNN-BP	-10.016 $\pm$ 2.331	-11.433 $\pm$ 1.166	-11.662 $\pm$ 0.448	-11.687 $\pm$ 1.740
RTSNet	-11.208 $\pm$ 2.438	-11.753 $\pm$ 1.182	-11.753 $\pm$ 0.449	-11.773 $\pm$ 0.685

(c) Initial Conditions

Training, Testing	Fixed, Fixed	Fixed, Random	Random, Random
Noise	-0.008 $\pm$ 0.434	NA NA	-0.019 $\pm$ 0.360
KF	-7.254 $\pm$ 1.012	NA NA	-7.426 $\pm$ 0.963
KS	-11.847 $\pm$ 1.190	NA NA	-12.025 $\pm$ 1.238
BRNN	22.277 $\pm$ 5.190	37.281 $\pm$ 2.003	26.606 $\pm$ 3.547
GNN-BP	-11.433 $\pm$ 1.166	-10.655 $\pm$ 1.219	-11.382 $\pm$ 1.164
RTSNet	-11.753 $\pm$ 1.182	-11.757 $\pm$ 1.187	-11.701 $\pm$ 1.214

performance of both algorithms coincides, demonstrating that RTSNet with $K = 1$ can learn to be optimal. Then, we show that the learning capabilities of RTSNet are scalable, namely, that they hold for different SS dimensions, and transferable, i.e., that it can be trained and evaluated with different trajectory lengths and initial conditions.

Approaching Optimality: We consider a 2×2 SS model (1)-(2), where $\mathbf{F}$ takes a *canonical* form and $\mathbf{H}$ is set to be the identity matrix, namely,

$$\mathbf{F} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{H} = \mathbf{I}. \tag{20}$$

We use multiple noise levels, in [dB] scale, of $\text{r}^2 \in \{10, 0, -10, -20, -30\}$, and $\nu \in \{-20, -10, 0\}$. The results provided in Table I, and in Fig. 4, shows that RTSNet converges to the MMSE estimate produced by the KS in the first two moments. This indicates that RTSNet successfully learns to implement the KS when it is MMSE optimal.

Scaling up Model Size: Next, we empirically show that RTSNet is a scalable smoothing architecture that is not limited to a small dimensions SS models. In this experiment $\mathbf{F}$ and $\mathbf{H}$ in their canonical form were considered, $\text{q}^2 = -20\,[\text{dB}]$, $\text{r}^2 = 0\,[\text{dB}]$, and $T = 20$. It is clearly observed in Table IIa that RTSNet retains its optimality also for high dimensional models, outperforming its DD benchmarks: BRNN is far from optimal, and the performance of GNN-BP degrades when the model dimensions increase.

Trajectory Length: To show generalization in T, the SS model in (20) is again considered, where $\text{q}^2 = -20\,[\text{dB}]$, and $\text{r}^2 = 0\,[\text{dB}]$. Here, we first train RTSNet and its DD benchmarks on one trajectory length and then testing it on a longer one. The results reported in Table IIb show that while RTSNet retains optimally for various trajectory lengths, BRNN completely diverges. We can also see the superiority of RTSNet over GNN-BP which demonstrates slightly degraded performance.

Initial Conditions: The operation of all smoothing algorithms depends on their initial state. We next train the DD benchmarks on trajectories with a different initial state compared to that used in test for the the SS model in (20) with $\text{q}^2 = -20\,[\text{dB}]$, $\text{r}^2 = 0\,[\text{dB}]$, and $T = 100$. The results provided in Table IIc demonstrates that while BRNN completely diverges, and GNN-BP is with slightly degraded performance when trained and then tested on different initial conditions, RTSNet still retains its optimally, which again demonstrates that it learns the smoothing task, rather than to overfit to trajectories presented during training.

C. *Linear SS Models with Partial Information*

Next, we demonstrate the merits of using RTSNet in linear settings with *partial information* where the KS is degraded due to the missing information. We consider mismatches in both the observation model as well as in the state evolution model. In particular, a mismatch in a model, e.g., the observation model, refers to a case in which a wrong setting of $\mathbf{h}(\cdot)$ is used, while the MB smoothers have access to the noise distribution. We also consider the case in which the corresponding model is unknown, e.g., both $\mathbf{h}(\cdot)$ and the noise distribution are

TABLE III: Linear SS - Observation mismatch: $\nu = -20\,[\mathrm{dB}]$

r^2 [dB]		10	0	-10	-20	-30
KF	Full	2.702 ± 0.885	-7.394 ± 0.901	-17.367 ± 0.957	-27.293 ± 0.966	-37.273 ± 1.029
KS	Full	-1.875 ± 1.285	-11.880 ± 1.272	-21.812 ± 1.149	-31.961 ± 1.265	-41.810 ± 1.156
BRNN	Opt	33.490 ± 4.490	22.120 ± 4.808	13.523 ± 5.334	4.058 ± 4.543	-5.876 ± 4.421
GNN-BP	Full	-1.417 ± 1.244	-11.397 ± 1.248	-21.383 ± 1.148	-31.545 ± 1.252	-41.395 ± 1.200
RTSNet	Full	-1.790 ± 1.242	-11.847 ± 1.190	-21.738 ± 1.281	-31.620 ± 1.107	-41.831 ± 1.289
KF	Partial	11.154 ± 3.023	0.926 ± 3.064	-9.160 ± 3.651	-18.428 ± 3.253	-28.786 ± 3.301
KS	Partial	5.502 ± 2.942	-4.825 ± 3.205	-15.062 ± 3.440	-24.198 ± 3.119	-34.592 ± 3.133
GNN-BP	Partial	-0.989 ± 1.223	-11.123 ± 1.243	-20.865 ± 1.587	-30.080 ± 1.354	-38.174 ± 2.624
RTSNet	Partial	-0.774 ± 1.243	-10.852 ± 1.158	-21.104 ± 1.216	-29.667 ± 1.215	-38.066 ± 1.136
RTSNet	$\hat{\mathbf{H}}$	-1.743 ± 1.269	-11.697 ± 1.241	-21.721 ± 1.153	-31.186 ± 1.220	-40.301 ± 1.169

unknown for the observation model; In such cases, since unlike GNN-BP and the MB benchmarks, RTSNet does not require prior knowledge of the noise distribution, we also evaluate it when it uses its data also to estimate the missing design parameter, e.g., $\mathbf{h}(\cdot)$, using least squares (LS).

Observation Model Mismatch: We first consider the case where the *design* observation model is mismatched. We again use the canonical model in (20) with the observation matrix being either *unknown*, or assumed to be $\mathbf{H}_0 = \mathbf{I}$, while the observation model is

$$\mathbf{H}_{\alpha^\circ} = \mathbf{R}_{x,y}(\alpha) \cdot \mathbf{H}_0, \quad \mathbf{R}_{x,y}(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

with $\mathbf{H}_0 = \mathbf{I}$. The data is generated with $\alpha^\circ = 10$. Such scenarios represent a setup where the true observed values are rotated by α°, e.g., a slight misalignment of the sensors exists.

We compare the performance of RTSNet and the KS when their design observation model is either $\mathbf{H}_{\alpha=10}$ (full information) or $\mathbf{H}_{\alpha=0}$ (mismatch). We also consider the case where the matrix is estimated from the data set via LS, denoting $\hat{\mathbf{H}}_{\alpha=10}$. The empirical results reported in Table III and in Fig. 5 demonstrate that while KS experienced a severe performance degradation, RTSNet is able to compensate for mismatches using the learned KG. When assuming a mismatched model $\mathbf{H}_0$, RTSNet converges to within a minor gap from the MMSE, which is further reduced when the data is also used to estimate the observation model. The latter indicates that even when the SS model is completely unknown, yet can be postulated as being linear, RTSNet can reliably smooth by using its data to both estimate the state evolution matrix as well as learn to smooth, while bypassing the need to impose a model on the noise.

State Evolution Mismatch: We next consider a similar, but more challenging use case. Here, the *design* evolution model is either *unknown*, or assumed to be $\mathbf{F}_0 = \mathbf{I}$, while the *true* evolution model is

$$\mathbf{F}_{\alpha^\circ} = \mathbf{R}_{x,y}(\alpha) \cdot \mathbf{F}_0, \quad \alpha^\circ = 10. \tag{21}$$

The empirical results reported in Table IV demonstrate that

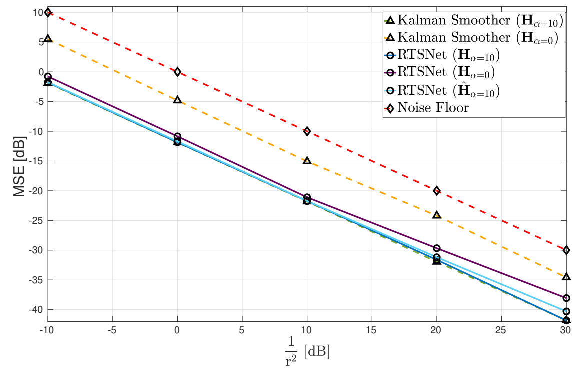


Fig. 5: Linear SS model - Observation mismatch.

TABLE IV: Linear SS - Evolution mismatch: $\nu = -20\,[\mathrm{dB}]$

r^2 [dB]		10	0	-10	-20	-30
KF	Full	3.450 ± 1.846	-6.594 ± 1.883	-16.562 ± 1.876	-26.601 ± 1.907	-36.565 ± 1.831
KS	Full	-3.843 ± 2.655	-13.913 ± 2.709	-23.592 ± 2.751	-33.861 ± 2.753	-43.593 ± 2.746
BRNN	Opt	40.915 ± 5.033	30.714 ± 5.317	20.796 ± 4.955	12.593 ± 4.281	2.411 ± 4.069
GNN-BP	Full	-1.975 ± 2.549	-11.850 ± 3.369	-21.403 ± 2.564	-30.579 ± 2.548	-41.016 ± 2.682
RTSNet	Full	-3.351 ± 2.699	-13.585 ± 2.673	-23.333 ± 2.671	-33.126 ± 2.628	-43.160 ± 2.649
KF	Partial	33.961 ± 3.933	23.833 ± 3.857	13.848 ± 3.683	4.199 ± 3.850	-6.434 ± 3.812
KS	Partial	32.963 ± 3.933	22.838 ± 3.859	12.853 ± 3.685	3.201 ± 3.851	-7.431 ± 3.809
GNN-BP	Partial	12.150 ± 4.215	-1.152 ± 6.240	-5.042 ± 4.272	-10.950 ± 2.790	-26.154 ± 3.968
RTSNet	Partial	10.553 ± 3.151	-2.011 ± 1.945	-10.689 ± 1.934	-21.683 ± 1.643	-31.887 ± 1.244
RTSNet	$\hat{\mathbf{F}}$	-3.433 ± 2.633	-12.945 ± 2.682	-23.013 ± 2.798	-32.932 ± 2.471	-41.864 ± 2.657

while KS experienced a severe performance degradation, RTSNet is able to compensate for unknown model information, by pre-estimating the evolution model via LS, and achieve the lower-bound. We can again clearly notice the performance superiority of our RTSNet over its DD counterparts, both for full and unknown information.

D. Kinematic Linear Differential Equations

As a concluding experiment in a setting of linear SS models, we consider smoothing in dynamics obtained from a stochastic differential equation (SDE) with a model mismatch. The state here represents a moving object obeying the constant acceleration (CA) model [45] for one dimensional kinematics. Here, $\mathbf{x}_t = (p_t, v_t, a_t)^\top \in \mathbb{R}^3$, where p_t, v_t, and a_t are the position, velocity, and acceleration, respectively, at time t. We observe noisy position measurements sampled at time intervals $\Delta t = 10^{-2}$, yielding a linear Gaussian SS model with $\mathbf{H} = (1, 0, 0)$ and

$$\mathbf{F} = \begin{pmatrix} 1 & \Delta\tau & \frac{1}{2}\Delta\tau^2 \\ 0 & 1 & \Delta\tau \\ 0 & 0 & 1 \end{pmatrix}; \mathbf{Q} = \mathrm{q}^2 \cdot \begin{pmatrix} \frac{1}{20}\Delta\tau^5 & \frac{1}{8}\Delta\tau^4 & \frac{1}{6}\Delta\tau^3 \\ \frac{1}{8}\Delta\tau^4 & \frac{1}{3}\Delta\tau^3 & \frac{1}{2}\Delta\tau^2 \\ \frac{1}{6}\Delta\tau^3 & \frac{1}{2}\Delta\tau^2 & \Delta\tau \end{pmatrix}.$$

While for the synthetic linear models considered in the previous subsections we used RTSNet with a single forward-backward pass, here we evaluate it with $K = 2$ unfolded pass, comparing it to both the KS and to GNN-BP when recovering

TABLE V: Linear kinematic SS model

Model	Error	KF	KS	GNN-BP	RTSNet-2
CA	Full State	-7.631 $\pm$ 2.891	-8.791 $\pm$ 3.054	14.351 $\pm$ 2.011	-8.432 $\pm$ 2.974
CA	Position	-22.074 $\pm$ 3.694	-23.221 $\pm$ 4.081	-11.456 $\pm$ 2.037	-22.241 $\pm$ 3.676
CV	Position	-7.657 $\pm$ 3.145	-14.752 $\pm$ 3.308	-10.732 $\pm$ 1.661	-15.900 $\pm$ 2.542

the entire state vector, as well as when recovering only the position (which is often the case in positioning applications). For the latter, we also consider the case where the smoothers assume a more simplified constant velocity (CV) model [45] state evolution for state evolution. The CV model captures in its state vector the position and velocity (without the acceleration), and is a popular model for kinematics due to its simplicity. Yet, for the current setting, it induces an inherent model mismatch.

The results are reported in Table V. For the GNN-BP smoother of [36], which is DD yet also requires knowledge of the SS model, we optimized $\mathbf{Q}$ via grid search to achieve the best performance, as it was shown to be unstable when substituting the true $\mathbf{Q}$. We observe in Table V that RTSNet comes within am minor gap of the KS in estimating both the full state as well as only the positions when it is known that the state obeys the CA model; When it is postulated that the state obeys the CV model, RTSNet outperforms all benchmarks.

E. Non-Linear Lorenz Attractor

We proceed to evaluating RTSNet in a non-linear SS model following the Lorenz attractor, which is a three-dimensional chaotic solution to the Lorenz system of ordinary differential equations. This synthetically generated chaotic system exemplifies dynamics formulated with SDEs, that demonstrates the task of smoothing a highly non-linear trajectory and a real world practical challenge of handling mismatches due to sampling a continuous-time signal into discrete-time [54]. As the dynamics are non-linear, here we use RTSNet with both $K = 1$ and $K = 2$ forward-backward passes, denoted RTSNet-1 and RTSNet-2, respectively.

The Lorenz attractor models the movement of a particle in 3D space, i.e., $m = 3$, which, when sampled at interval $\Delta\tau$, obeys a state evolution model with $\mathbf{f}(\mathbf{x}_t) = \mathbf{F}(\mathbf{x}_t) \cdot \mathbf{x}_t$, where

$$\mathbf{F}(\mathbf{x}_\tau) = \mathbf{I} + \sum_{j=1}^{J} \frac{(\mathbf{A}(\mathbf{x}_\tau) \cdot \Delta\tau)^j}{j!}, \qquad (22)$$

with J denoting the order of the Taylor series approximation used to obtain the model (where we use $J = 5$ when generating the data), and

$$\mathbf{A}(\mathbf{x}_\tau) = \begin{pmatrix} -10 & 10 & 0 \\ 28 & -1 & -x_{1,\tau} \\ 0 & x_{1,\tau} & -\frac{8}{3} \end{pmatrix}.$$

We first evaluate RTSNet under noisy rotated state observations, with and without observation model mismatch as well as with sampling mismatches, after which we evaluate it with non-linear observations.

Rotated State Observations: Here, we consider the discrete-time state evolution with $\Delta\tau = 0.02$. The observations model $\mathbf{h}(\cdot)$ is set to a rotation matrix with $\alpha = 1^\circ$, whereas $T = 100$ and $\nu = -20\,[\text{dB}]$. As in Subsection IV-C, we consider the cases where the smoothers are aware of the rotation (Full information); when the assumed state evolution is the identity matrix instead of the slightly rotated one (Partial information), as well as when it is estimated from the data set via LS ($\hat{\mathbf{H}}$).

TABLE VI: Lorenz attractor - Observation Mismatch:

r^2 [dB]		10	0	-10	-20	-30
Noise		10.017 $\pm$ 0.334	0.005 $\pm$ 0.376	-10.011 $\pm$ 0.368	-19.942 $\pm$ 0.347	-29.986 $\pm$ 0.354
EKF	Full	-0.299 $\pm$ 1.084	-10.533 $\pm$ 1.016	-20.493 $\pm$ 0.969	-30.348 $\pm$ 1.016	-40.483 $\pm$ 0.992
EKS	Full	-3.892 $\pm$ 0.996	-13.752 $\pm$ 1.161	-23.868 $\pm$ 1.025	-33.743 $\pm$ 1.013	-43.755 $\pm$ 1.145
GNN-BP	Full	-2.263 $\pm$ 1.113	-12.398 $\pm$ 1.182	-22.413 $\pm$ 1.076	-31.040 $\pm$ 1.046	-42.368 $\pm$ 1.256
RTSNet-2	Full	-3.138 $\pm$ 0.983	-13.330 $\pm$ 1.195	-23.304 $\pm$ 1.036	-33.311 $\pm$ 0.999	-43.235 $\pm$ 1.112
EKF	Partial	-0.258 $\pm$ 1.073	-9.747 $\pm$ 0.988	-15.945 $\pm$ 0.769	-17.549 $\pm$ 0.325	-17.752 $\pm$ 0.109
EKS	Partial	-3.824 $\pm$ 0.976	-12.932 $\pm$ 1.122	-18.957 $\pm$ 0.853	-20.363 $\pm$ 0.366	-20.563 $\pm$ 0.126
GNN-BP	Partial	-1.921 $\pm$ 0.962	-11.959 $\pm$ 1.308	-18.724 $\pm$ 0.871	-23.076 $\pm$ 0.743	-23.351 $\pm$ 0.472
RTSNet-2	Partial	-3.010 $\pm$ 1.067	-13.290 $\pm$ 1.175	-22.620 $\pm$ 1.077	-31.789 $\pm$ 1.207	-41.874 $\pm$ 1.216
RTSNet-2	$\hat{\mathbf{H}}$	-3.127 $\pm$ 1.057	-13.315 $\pm$ 1.194	-23.158 $\pm$ 1.043	-32.581 $\pm$ 1.119	-42.928 $\pm$ 1.145

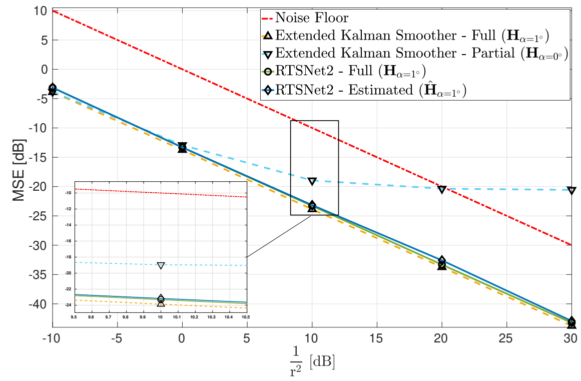


Fig. 6: $T = 2000$, $\nu = -20\,[\text{dB}]$, $\mathbf{h}(\cdot) = \mathbf{I}$.

The results reported in Table VI and in Fig. 6 demonstrate that although RTSNet does not have access to the true statistics of the noise, for the case of *full* observation model information, it still achieves the MSE lower-bound. It is also observed that a mismatched state observation model obtained from a seemingly minor rotation causes a severe performance degradation for the KS, which is sensitive to model uncertainty, while RTSNet is able to learn from data to overcome such mismatches. Finally, it is observed that RTSNet consistently and notably outperform the DD benchmark of [36], and that its unfolding with $K = 2$ forward-backward passes indeed achieves improved performance compared with a single pass.

Sampling Mismatch: Here, we demonstrate a practical case where a physical process evolves in continuous-time, but the smoother only has access to noisy observations in discrete-time, which then results in an inherent mismatch in the SS model. We generate data from an approximate continuous-time noise-less state evolution $\mathbf{F}(\mathbf{x}_\tau)$, with high resolution time interval $\Delta\tau = 10^{-5}$. We then sub-sampled the process by a ratio of $\frac{1}{2000}$ and get a decimated process with $\Delta t = 0.02$. Finally we generated noisy observations of the true state, by using an identity observation matrix $\mathbf{h}(\cdot) = \mathbf{I}$ and non-correlated observation noise with $\mathrm{r}^2 = 0\,[\mathrm{dB}]$. See an example in Fig. 7: ground truth, and noisy observations, respectively.

The MSE values for smoothing sequences with length $T = 3000$, reported in Table VII, demonstrate that RTSNet overcomes the mismatch induced by representing a continuous-time SS model in discrete-time, achieving a substantial processing gain over it MB and DD counterparts due to its learning capabilities. In Fig. 7, we visualize how this gain is translated into clearly improved smoothing of a single trajectory.

Noisy Non-Linear Observations: Finally, we consider the case of the discrete-time Lorenz attractor, with non-linear observations, which take the form of a transformation from a cartesian coordinate system to spherical coordinates. In such settings, the observations function is given by

$$\mathbf{h}([x,y,z]^T) \triangleq \begin{bmatrix} \sqrt{x^2+y^2+z^2} \\ \tan^{-1}\left(\frac{y}{x}\right) \\ \cos^{-1}\left(\frac{z}{\sqrt{x^2+y^2+z^2}}\right) \end{bmatrix}.$$

We further set $T = 20$ and $\nu = 0\,[\mathrm{dB}]$.

The MSE achieved by RTSNet with $K = 2$ forward-backward passes is compared with that of the KS, PS and RTSNet with $K = 1$, reported in Table VIII and depicted in Fig. 8. It is clearly observed here that in such non-linear setups, RTSNet outperforms its MB counterparts which operate with full knowledge of the underlying SS model, indicating on the ability of its DNN augmentation and unfolded architecture to improve performance in the presence of non-linearities.

F. Complexity Analysis

So far, we showed that RTSNet provides improved MSE performance, outperfoming both its DD counterparts, and also its MB counterparts operating with partial information or under non-linear dynamics. We conclude our empirical study by showing that these gains of RTSNet do not come at the expense of computational complexity during inference and of increased DNN size.

In Table IX we report the average inference time, for all filters (without parallelism), on the Lorenz attractor state estimation task . The stopwatch timings were measured on the same platform – *Google Colab* with CPU: Intel(R) Xeon(R) CPU @ 2.20GHz, GPU: Tesla P100-PCIE-16GB. We see that RTSNet is very competitive comparing with classical methods, and outperforms GNN-BP. This is mainly thanks to its highly efficient neural network computations and the fact that, unlike the MB filters, it does not involve linearization and matrix inversions for each time step.

Next, we compare the DNN size, i.e., the number of trainable parameters, of RTSNet with that of GNN-BP for a few representative use cases. The resulting number of parameters are reported in Fig. X, where we can see that RTSNet is more compact and therefore also easier to train. In particular, it noted that RTSNet repeated achieves improved performance over the DD GNN-BP benchmark of [36] for the use cases tested, while using a lower complexity architecture with less trainable parameters, despite the fact that the architecture of [36] is also specifically designed to be compact and efficient. The reduced parameterization is translated into faster training and inference.

V. Conclusion

In this work we presented RTSNet, a hybrid combination of deep learning with the classic KS. Our design identifies the SS-model-dependent computations of the KS, replacing them with a dedicated RNNs operating on specific features encapsulating the information needed for its operation, while unfolding the algorithm to enable multiple trainable forward-backward passes. Our empirical study shows that doing so enables RTSNet to carry out offline state estimation in the same manner as KS, while learning to overcome model mismatches and non-linearities. RTSNet uses a relatively compact RNN that can be trained with a relatively small data set and infers a reduced complexity.

References

[1] X. Ni, G. Revach, N. Shlezinger, R. J. G. van Sloun, and Y. C. Eldar, "RTSNet: Deep learning aided Kalman smoothing," in *Proc. IEEE ICASSP*, 2022, pp. 5902–5906.
[2] J. Durbin and S. J. Koopman, *Time series analysis by state space methods*. Oxford University Press, 2012.
[3] N. Wiener, *Extrapolation, interpolation, and smoothing of stationary time series: With engineering applications*. MIT press Cambridge, MA, 1949, vol. 113, no. 21.
[4] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.
[5] H. E. Rauch, F. Tung, and C. T. Striebel, "Maximum likelihood estimates of linear dynamic systems," *AIAA Journal*, vol. 3, no. 8, pp. 1445–1450, 1965.
[6] M. Gruber, "An approach to target tracking," MIT Lexington Lincoln Lab, Tech. Rep. AD0654272, 02 1967.
[7] R. E. Larson, R. M. Dressler, and R. S. Ratner, "Application of the Extended Kalman filter to ballistic trajectory estimation," Stanford Research Institute, Tech. Rep. AD0815377, 01 1967.
[8] S. J. Julier and J. K. Uhlmann, "New extension of the Kalman filter to nonlinear systems," in *Signal Processing, Sensor Fusion, and Target Recognition VI*, vol. 3068. International Society for Optics and Photonics, 1997, pp. 182–193.
[9] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," in *IEE proceedings F (radar and signal processing)*, vol. 140, no. 2. IET, 1993, pp. 107–113.
[10] P. Del Moral, "Nonlinear filtering: Interacting particle resolution," *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, vol. 325, no. 6, pp. 653–658, 1997.
[11] J. S. Liu and R. Chen, "Sequential Monte Carlo methods for dynamic systems," *Journal of the American Statistical Association*, vol. 93, no. 443, pp. 1032–1044, 1998.
[12] Z. Ghahramani and G. E. Hinton, "Parameter estimation for linear dynamical systems," University of Totronto, Dept. of Computer Science, Tech. Rep. CRG-TR-96-2, 02 1996.
[13] J. Dauwels, A. W. Eckford, S. Korl, and H. Loeliger, "Expectation maximization as message passing - part I: principles and gaussian messages," *CoRR*, vol. abs/0910.2832, 2009. [Online]. Available: http://arxiv.org/abs/0910.2832
[14] K.-V. Yuen and S.-C. Kuok, "Online updating and uncertainty quantification using nonstationary output-only measurement," *Mechanical Systems and Signal Processing*, vol. 66, pp. 62–77, 2016.
[15] H.-Q. Mu, S.-C. Kuok, and K.-V. Yuen, "Stable robust Extended Kalman filter," *Journal of Aerospace Engineering*, vol. 30, no. 2, p. B4016010, 2017.
[16] L. Martino, J. Read, V. Elvira, and F. Louzada, "Cooperative parallel particle filters for online model selection and applications to urban mobility," *Digital Signal Processing*, vol. 60, pp. 172–185, 2017.

TABLE VII: MSE [dB] - Lorenz attractor with sampling mismatch.

Noise	EKF	PF	KalmanNet	EKS	PS	BRNN	GNN-BP	RTSNet-1	RTSNet-2
-0.024 $\pm$ 0.049	-6.316 $\pm$ 0.135	-5.333 $\pm$ 0.136	-11.106 $\pm$ 0.224	-10.075 $\pm$ 0.191	-7.222 $\pm$ 0.202	-2.342 $\pm$ 0.092	-16.479 $\pm$ 0.352	-15.436 $\pm$ 0.329	-16.803 $\pm$ 0.301

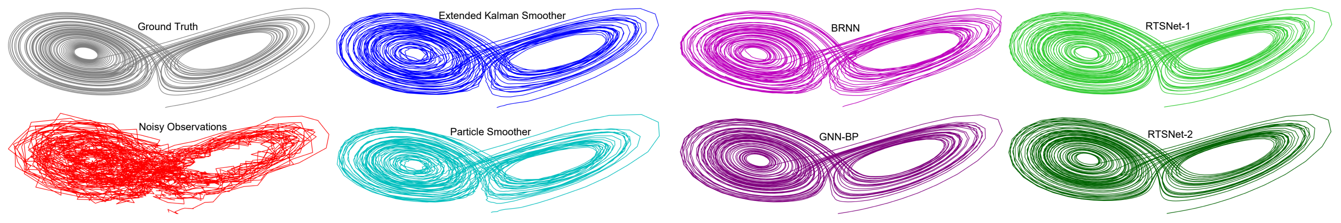


Fig. 7: Lorenz attractor with sampling mismatch (decimation), $T = 3000$.

TABLE VIII: Lorenz attractor with non-linear observations

r^2 [dB]		10	0	-10	-20	-30
EKF	Full	24.693 $\pm$ 4.147	12.197 $\pm$ 8.061	-6.343 $\pm$ 1.961	-15.574 $\pm$ 3.451	-26.418 $\pm$ 1.743
EKS	Full	24.739 $\pm$ 4.313	12.045 $\pm$ 8.260	-7.613 $\pm$ 2.474	-16.134 $\pm$ 5.157	-28.211 $\pm$ 1.548
PS	Full	20.490 $\pm$ 6.187	7.612 $\pm$ 10.071	-7.093 $\pm$ 1.822	-17.293 $\pm$ 1.704	-27.138 $\pm$ 1.743
RTSNet-1	Full	21.094 $\pm$ 2.901	10.804 $\pm$ 8.999	-8.074 $\pm$ 1.500	-17.941 $\pm$ 1.712	-27.476 $\pm$ 1.553
RTSNet-2	Full	19.849 $\pm$ 4.183	6.100 $\pm$ 6.614	-8.122 $\pm$ 1.521	-17.960 $\pm$ 1.676	-27.630 $\pm$ 1.558

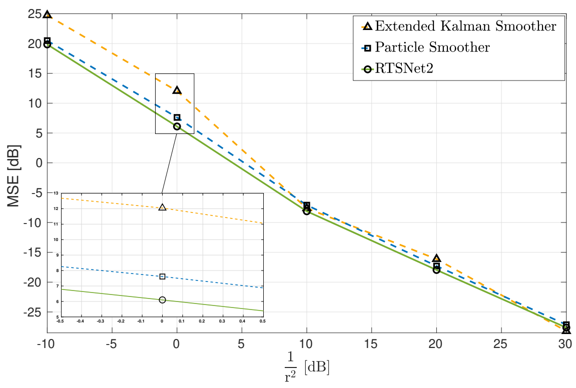


Fig. 8: $T = 20$, $\nu = 0$ [dB], $\mathbf{h}(\cdot)$ non-linear.

[17] L. Xu and R. Niu, "EKFNet: Learning system noise statistics from measurement data," in *Proc. IEEE ICASSP*, 2021, pp. 4560–4564.

[18] S. T. Barratt and S. P. Boyd, "Fitting a Kalman smoother to data," in *IEEE American Control Conference (ACC)*, 2020, pp. 1526–1531.

[19] M. Zorzi, "On the robustness of the Bayes and Wiener estimators under model uncertainty," *Automatica*, vol. 83, pp. 133–140, 2017.

[20] A. Longhini, M. Perbellini, S. Gottardi, S. Yi, H. Liu, and M. Zorzi, "Learning the tuned liquid damper dynamics by means of a robust ekf," in *2021 American Control Conference (ACC)*, 2021, pp. 60–65.

[21] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[22] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing*, 2017, pp. 5998–6008.

[24] P. Becker, H. Pandya, G. Gebhardt, C. Zhao, C. J. Taylor, and G. Neumann, "Recurrent Kalman networks: Factorized inference in high-dimensional deep feature spaces," in *International Conference on Machine Learning*. PMLR, 2019, pp. 544–552.

[25] M. Zaheer, A. Ahmed, and A. J. Smola, "Latent LSTM allocation: Joint clustering and non-linear dynamic modeling of sequence data," in *International Conference on Machine Learning*, 2017, pp. 3967–3976.

[26] R. G. Krishnan, U. Shalit, and D. A. Sontag, "Deep Kalman filters," *CoRR*, vol. abs/1511.05121, 2015.

[27] E. Archer, I. M. Park, L. Buesing, J. Cunningham, and L. Paninski, "Black box variational inference for state space models," *arXiv preprint arXiv:1511.07367*, 11 2015.

[28] M. Karl, M. Soelch, J. Bayer, and P. van der Smagt, "Deep variational Bayes filters: Unsupervised learning of state space models from raw data," in *International Conference on Learning Representations*, 2017.

[29] R. Krishnan, U. Shalit, and D. Sontag, "Structured inference networks for nonlinear state space models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[30] M. Fraccaro, S. D. Kamronn, U. Paquet, and O. Winther, "A disentangled recognition and nonlinear dynamics model for unsupervised learning," in *Advances in Neural Information Processing Systems*, 2017.

[31] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel, "Backprop kf: Learning discriminative deterministic state estimators," in *Advances in Neural Information Processing Systems*, 2016, pp. 4376–4384.

[32] B. Laufer-Goldshtein, R. Talmon, and S. Gannot, "A hybrid approach for speaker tracking based on TDOA and data-driven models," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 26, no. 4, pp. 725–735, 2018.

[33] L. Zhou, Z. Luo, T. Shen, J. Zhang, M. Zhen, Y. Yao, T. Fang, and L. Quan, "KFNet: Learning temporal camera relocalization using Kalman filtering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 4919–4928.

[34] H. Coskun, F. Achilles, R. DiPietro, N. Navab, and F. Tombari, "Long short-term memory Kalman filters: Recurrent neural estimators for pose regularization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5524–5532.

[35] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, "Deep state space models for time series forecasting," in *Advances in Neural Information Processing Systems*, 2018, pp. 7785–7794.

[36] V. G. Satorras, Z. Akata, and M. Welling, "Combining generative and discriminative models for hybrid inference," in *Advances in Neural Information Processing Systems*, 2019, pp. 13 802–13 812.

[37] G. Revach, N. Shlezinger, X. Ni, A. L. Escoriza, R. J. G. van Sloun, and Y. C. Eldar, "KalmanNet: Neural network aided Kalman filtering for partially known dynamics," *IEEE Trans. Signal Process.*, vol. 70, pp. 1532–1547, 2022.

[38] N. Shlezinger, J. Whang, Y. C. Eldar, and A. G. Dimakis, "Model-based deep learning," *CoRR*, vol. abs/2012.08405, 2020.

[39] N. Shlezinger, Y. C. Eldar, and S. P. Boyd, "Model-based deep learning: On the intersection of deep learning and optimization," *IEEE Access*, vol. 10, pp. 115 384–115 398, 2022.

[40] A. López Escoriza, G. Revach, N. Shlezinger, and R. J. G. van Sloun, "Data-driven Kalman-based velocity estimation for autonomous racing," in *Proc. IEEE ICAS*, 2021.

[41] I. Klein, G. Revach, N. Shlezinger, J. E. Mehr, R. J. G. van Sloun, and Y. C. Eldar, "Uncertainty in data-driven Kalman filtering for partially known state-space models," in *Proc. IEEE ICASSP*, 2022, pp. 3194–3198.

[42] G. Revach, N. Shlezinger, T. Locher, X. Ni, R. J. van Sloun, and Y. C. Eldar, "Unsupervised learned Kalman filtering," in *European Signal Processing Conference (EUSIPCO)*, 2022.

[43] V. Monga, Y. Li, and Y. C. Eldar, "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing," *IEEE Signal Process. Mag.*, vol. 38, no. 2, pp. 18–44, 2021.

[44] N. Shlezinger and T. Routtenberg, "Discriminative and generative learning for linear estimation of random signals [lecture notes]," *arXiv preprint arXiv:2206.04432*, 2022.

TABLE IX: Inference Time $[\text{sec}]$ - Lorenz attractor.

Use Case	Trajectory Length	KF	PF	KalmanNet	KS	PS	GNN-BP	RTSNet-1	RTSNet-2
Non-Linear Observations	$T = 20$	0.0501	NA	NA	0.0946	5.0175	NA	0.0605	0.1178
Linear Observations	$T = 100$	0.2194	NA	NA	0.4344	24.4158	1.2513	0.2950	NA
Decimation $(K = 2)$	$T = 3000$	4.3583	45.4791	4.9226	6.5164	452.8513	25.4527	7.3587	14.6174
Decimation $(K = 5)$	$T = 3000$	6.2641	71.6549	NA	10.3243	723.9320	NA	NA	NA

TABLE X: Network Size - Number of Trainable Parameters

Use Case	Linear - 2×2	Linear - 5×5	Lorentz - Decimation
RTSNet	$7,370$	$28,285$	$33,270$ $(\#1)$, $66,540$ $(\#2)$
GNN-BP	$40,947$	$41,814$	$41,236$

[45] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation: Theory algorithms and software*. John Wiley & Sons, 01 2004.

[46] S. Särkkä, "Unscented rauch-tung-striebel smoother," *IEEE Trans. Autom. Control.*, vol. 53, no. 3, pp. 845–849, 2008.

[47] G. J. Bierman, *Factorization methods for discrete sequential estimation*. Courier Corporation, 1977.

[48] ——, "Fixed interval smoothing with discrete measurements," *International Journal of Control*, vol. 18, no. 1, pp. 65–75, 1973.

[49] F. Wadehn, *State space methods with applications in biomedical signal processing*. ETH Zurich, 2019, vol. 31.

[50] N. Samuel, T. Diskin, and A. Wiesel, "Learning to detect," *IEEE Trans. Signal Process.*, vol. 67, no. 10, pp. 2554–2564, 2019.

[51] N. Shlezinger, R. Fu, and Y. C. Eldar, "DeepSIC: Deep soft interference cancellation for multiuser MIMO detection," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1349–1362, 2021.

[52] S. J. Godsill, A. Doucet, and M. West, "Monte carlo smoothing for nonlinear time series," *Journal of the american statistical association*, vol. 99, no. 465, pp. 156–168, 2004.

[53] Jerker Nordh, "pyParticleEst - Particle based methods in Python," 2015. [Online]. Available: https://pyparticleest.readthedocs.io/en/latest/index.html

[54] W. Gilpin, "Chaos as an interpretable benchmark for forecasting and data-driven modelling," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.