RECURRENT KALMAN NETWORKS: FACTORIZED INFERENCE IN HIGH-DIMENSIONAL DEEP FEATURE SPACES

Anonymous authors
Paper under double-blind review

ABSTRACT

State estimation together with state prediction is a crucial task for many applications. Typically, sensory observations give only partial and noisy information about the state of the environment. A well-known tool for performing state estimation under these conditions is the Kalman filter (Kalman et al., 1960). However, the Kalman filter is limited to problems with known, linear models and good estimates about the system noise. Recent deep learning approaches integrate a nonlinear encoder into the KF equations that maps the high-dimensional observation to the typically low-dimensional state of the system (Haarnoja et al., 2016). However, these approaches are still limited to systems with known dynamics that are either linear or it requires approximations such as an extended Kalman filter. In contrast, our approach does not use a pre-defined state representation but learns a high-dimensional factorized representation that is used for inference using locally linear models. While our locally linear modelling and factorization assumptions are in general not true for the original low-dimensional state space of the system, the network finds a high-dimensional latent space where these assumptions hold to perform efficient inference. This state representation is learned jointly with the transition and noise models by backpropagation. The resulting network architecture, which we call *Recurrent Kalman Network* (RKN), can be used for any time-series data, similar to a LSTM (Hochreiter and Schmidhuber, 1997) but uses an explicit representation of uncertainty. As shown by our experiments, the RKN obtains much more accurate uncertainty estimates than an LSTM or Gated Recurrent Units (GRUs) (Cho et al., 2014) while also showing a slightly improved prediction performance.

1 INTRODUCTION

In order to employ robots outside of factories they need to be able to reason about their own state as well as the state of their environment. Both can often only be perceived through observations and measurements. While the state of a robot can usually be estimated easily from sensor measurements in joints and motors, observing the state of an unstructured environment is a big challenge. Such observations are often high-dimensional and only provide partial information about the state. Images are a good example: Already for images with low resolution the number of pixels can quickly exceed tens or hundreds of thousands. Furthermore, it is impossible to obtain any information about the dynamics, such as velocities, from a single image. Additionally, those observations may be noisy or may not contain any information useful for the task at hand. If we again consider images as an example, such noise can be introduced by poor illumination or motion blur and occlusions can prevent us from observing some or all relevant aspects. In order to keep reasoning about the state in such cases, a filter needs to be applied.

Another scenario is predicting future states which is essential for autonomous robots, for example to assess the consequences of actions. To this end, an initial estimate of the current state is necessary which again has to be inferred from observations.

A common approach to form state estimates from observations is the Kalman filter (Kalman et al., 1960) (KF). It assumes linear models of the system dynamics to iteratively predict the next state and

updates it using the current observation. While being a well-known and studied tool, the classical version of the KF relies on a couple of assumptions that limit its applicability. First, it relies on a transition and an observation model, describing the system dynamics and the process generating the observations respectively. These models are often not available and obtaining them by classical means is a hard and time consuming task which requires considerable knowledge of the system domain. Therefore, it is desirable to circumvent the manual modeling by learning the models directly from exemplary data. The second assumption of the traditional KF model is that the observation and transition models are linear. While approximations for non-linear models such as the extended KF or the unscented KF exists, these approximations render an end-to-end learning of the KF models difficult.

We introduce the *Recurrent Kalman Network*, an end-to-end learning approach for Kalman filtering and prediction. While Kalman filtering in the original state space requires approximations due to the non-linear models as well as matrix inversions that are hard to back-propagate and computationally expensive, the RKN uses a learned high-dimensional latent state representation that allows for efficient inference using locally linear transition models and a factorized belief state representation which avoids expensive and numerically problematic matrix inversions. Conceptually, this idea is related to kernel methods which use high-dimensional feature spaces to approximate nonlinear functions with linear models (Gebhardt et al., 2017). However, in difference to kernel feature spaces, our feature space is given by a deep encoder and learned in an end-to-end manner.

The RKN can be used for any time-series data set for which LSTMs and GRUs are currently the state of the art. In contrast to those, the RKN uses an explicit representation of uncertainty which governs the gating between keeping the current information in memory or updating it with the current observation. While the RKN shows a slightly improved performance in terms of state estimation errors, both LSTMs and GRUs struggle with estimating the uncertainty of the prediction while the RKN can provide accurate uncertainty estimates.

1.1 Related Work

State estimation for dynamical systems is an important and well studied problem. Various approaches exist and are used in many everyday applications. The most famous of those approaches is the previously mentioned Kalman filter (Kalman et al., 1960). The Kalman filter assumes linear observation and transition models and Gaussian distributions which restrict its applicability in practical scenarios. Some more sophisticated methods, such as the extended Kalman filter (Smith et al., 1962) or the unscented Kalman filter (Julier and Uhlmann, 1997) are capable of dealing with nonlinear models. Others, such as particle filters (Del Moral, 1996), are not only capable of dealing with nonlinear models but also support arbitrary multimodal distributions. However, all of the above mentioned methods assume the observation and transition models as well as good estimates of the noise covariances to be known.

Deep Learning provides the tools to learn those models, even for nonlinear systems and complex observations. Convolutional neural networks (CNNs) are powerful models to process images and learn to interpret complex scenes (LeCun et al., 2015) and can be used to learn observation models. Recurrent neural networks (RNNs) are capable of dealing with time series data and extracting the underlying dynamics and dependencies between time steps (Lipton et al., 2015) and are well suited for learning transition models of dynamical systems.

One family of approaches utilizing deep learning interprets the problem as a variational inference problem (Fraccaro et al., 2017; Karl et al., 2016; Watter et al., 2015). They optimize a variation or extension of the variational lower bound by means of the stochastic gradient variational bayes approach (Kingma and Welling, 2013). Those approaches do not have an explicit notion of the state uncertainty and are used in a generative manner, i.e., they are trained to predict future or impute missing observations.

Another recent approach, the LSTM-KF (Coskun et al., 2017), employs LSTM networks to learn a transition function together with estimates of the transition and observation noise for pose estimation from images.

The framework of nonparametric inference (Song et al., 2013) provides an alternative way of dealing with nonlinear systems. The Kernel Kalman filter (Gebhardt et al., 2017) was derived as a tool

to perform state estimation using this framework. By embedding the states and observations into potentially infinite dimensional reproducing kernel Hilbert spaces the dynamics can be approximated with linear models. After making the computations tractable by virtue of the kernel trick the models can be learned using regression.

The BackpropKF (Haarnoja et al., 2016) applies a CNN to estimate the observable parts of the true state given the observation. This CNN additionally outputs a covariance matrix indicating the models certainty about the estimate and allows the subsequent use of an (extended) Kalman filter with known transition model. By propagating the error through the Kalman filter the CNN learns to give reasonable covariance estimates. In contrast, we let our model chose the feature space that is used for the inference such that locally linear models can be learned and the KF computations can be simplified due to our factorization assumptions.

2 PRELIMINARIES

The Kalman filter (Kalman et al., 1960) works by iteratively applying two steps, predict and update. It assumes additive Gaussian noise with zero mean and covariances $\mathbf{\Sigma}^{\text{trans}}$ and $\mathbf{\Sigma}^{\text{obs}}$ on both transitions and observations. During the prediction step the transition model $\mathbf{A}$ is used to infer the next prior state estimate $\left(\mathbf{x}_t^-, \mathbf{\Sigma}_t^-\right)$, i.e., a-priori to the observation, by

$$\mathbf{x}_t^- = \mathbf{A}\mathbf{x}_{t-1}^+ \quad \text{and} \quad \mathbf{\Sigma}_t^- = \mathbf{A}\mathbf{\Sigma}_{t-1}^+\mathbf{A}^T + \mathbf{\Sigma}^{\text{trans}}.$$

The prior estimate is then updated using the current observation $\mathbf{w_t}$ and the observation model $\mathbf{H}$ to obtain the posterior estimate $\left(\mathbf{x}_t^+, \mathbf{\Sigma}_t^+\right)$, i.e.,

$$\mathbf{x}_t^+ = \mathbf{x}_t^- + \mathbf{Q}_t\left(\mathbf{w}_t - \mathbf{H}\mathbf{x}_t^-\right) \text{ and } \mathbf{\Sigma}_t^+ = \left(\mathbf{I} - \mathbf{Q}_t\mathbf{H}\right)\mathbf{\Sigma}_t^-, \text{ with } \mathbf{Q}_t = \mathbf{\Sigma}_t^-\mathbf{H}^T\left(\mathbf{H}\mathbf{\Sigma}_t^-\mathbf{H}^T + \mathbf{\Sigma}^{\text{obs}}\right)^{-1},$$

where $\mathbf{I}$ denotes the identity matrix. The matrix $\mathbf{Q}$ is referred to as the Kalman gain. The whole update step can be interpreted as a weighted average between state and observation estimate, where the weighting, i.e., $\mathbf{Q}$, depends on the uncertainty about those estimates. Besides $\mathbf{A}$ and $\mathbf{H}$, the transition covariance $\mathbf{\Sigma}^{\text{trans}}$ and the observation covariance $\mathbf{\Sigma}^{\text{obs}}$ need to be given to the filter. If currently no observation is present or future states should be predicted, the update step is omitted.

3 FACTORIZED INFERENCE IN DEEP FEATURE SPACES

Lifting the original input space to a high-dimensional feature space where linear operations are feasible is a common approach in machine learning, e.g., in kernel regression and SVMs. The *Recurrent Kalman Network* (RKN) transfers this idea to state estimation and filtering, i.e., we learn a high dimensional deep feature space that allows for efficient inference using the Kalman update equations even for complex systems with high dimensional observations. To achieve this we assume a simple relation between latent observations and latent states and that the belief state representation can be factorized into independent Gaussian distributions as described in the following sections.

We train an encoder, i.e., a convolutional neural network, to extract latent observations $\mathbf{w}_t$ and their uncertainty estimates $\mathbf{\Sigma}_t^{\text{obs}}$ from high-dimensional and potentially noisy observations. Additionally, we learn a locally linear transition model that models the system dynamics in the latent space. Finally, a decoder is needed to map the latent state estimates to interpretable outputs. Encoder, transition model, and decoder are learned jointly in an end-to-end manner.

This model can be used for various tasks such as filtering, prediction and imputing missing data. Furthermore, we can obtain different types of output. In this work we consider outputting images $\mathbf{o}_t$ and low dimensional observations $\mathbf{s}_t^+$ e.g. positions. We train our model using the log-likelihood of the prediction. Hence, in either case, the model does not only outputs the actual prediction of image or state, but also a reasonable estimate about its certainty in that prediction.

3.1 LEARNING HIGH-DIMENSIONAL LATENT SPACES FOR NONLINEAR DYNAMICS

In the case of nonlinear system dynamics, the RKN encoder does not learn a mapping to the true state but rather a mapping into a high-dimensional latent observation space $\mathcal{W} = \mathbb{R}^m$. The transition

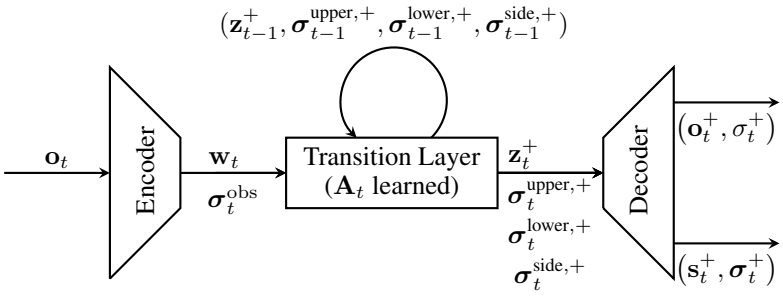


Figure 1: The *Recurrent Kalman Network*. An encoder network extracts latent features $\mathbf{w}_t$ from the current observation $\mathbf{o}_t$. Additionally, it emits an estimate of the uncertainty in the features via the variance $\boldsymbol{\sigma}_t^{\text{obs}}$. Subsequently, a Kalman filter with learned transition model $\mathbf{A}_t$ is used to infer the current latent state $\left(\mathbf{z}_t^+, \boldsymbol{\sigma}_t^+\right)$ using the last state $\left(\mathbf{z}_{t-1}^+, \boldsymbol{\sigma}_{t-1}^+\right)$ and current latent observation $\left(\mathbf{w}_t, \boldsymbol{\sigma}_t^{\text{obs}}\right)$. The current latent state $\mathbf{z}_t$ consists of the observable units $\mathbf{p}_t$ as well as the corresponding memory units $\mathbf{m}_t$. Finally, a decoder produces either $\left(\mathbf{s}_t^+, \boldsymbol{\sigma}_t^+\right)$, a low dimensional observation and an element-wise uncertainty estimate, or $\left(\mathbf{o}_t^+, \sigma_t\right)$, a noise free image and a global uncertainty estimate.

model is then learned in another latent space $\mathcal{Z} = \mathbb{R}^n$, i.e., the latent state space. Those two spaces are related by a linear latent observation model $\mathbf{H} = \begin{pmatrix}\mathbf{I}_m & \mathbf{0}_{m\times(n-m)}\end{pmatrix}$, i.e.,

$$\mathbf{w} = \mathbf{H}\mathbf{z} \quad \text{with } \mathbf{w} \in \mathcal{W} \text{ and } \mathbf{z} \in \mathcal{Z},$$

where $\mathbf{I}_m$ denotes the $m \times m$ identity matrix and $\mathbf{0}_{m\times(n-m)}$ denotes a $m \times (n-m)$ matrix filled with zeros.

The idea behind this choice is to split the latent state vector $\mathbf{z}_t$ into two parts, the vector $\mathbf{p}_t$ for holding information that can directly be extracted from the observations and the vector $\mathbf{m}_t$ to store information inferred over time, e.g., velocities. We refer to the former as the observation or upper part and the latter as the memory or lower part of the latent state. For an ordinary dynamical system and images as observations the former may correspond to positions while the latter corresponds to velocities. Clearly, this choice only makes sense for $m \leq n$ and in this work we assume $n = 2m$, i.e., for each observation unit p_i, we also represent a memory unit m_i that stores the velocity information.

Depending on the desired model output and the available training data, we use two different decoder types. If we want to directly infer the position components of the state of the system $\mathbf{s}_t^+$, we use a dense neural network that maps the latent state estimate $\left(\mathbf{z}_t^+, \boldsymbol{\sigma}_t^+\right)$ to a low dimensional output $\left(\mathbf{s}_t^+, \boldsymbol{\sigma}_t^+\right)$. If we want to reconstruct noise free images, we use a transposed-convolutional network to obtain $\left(\mathbf{o}_t^+, \sigma_t^+\right)$. Since it does not make sense to compute an output variance for each pixel individually, a scalar σ_t^+ is used for the whole image.

For each sequence we initialize $\mathbf{z}_0^+$ with an all zeros vector and $\boldsymbol{\Sigma}_0^+$ with $10 \cdot \mathbf{I}$. In practice, it is beneficial to normalize $\mathbf{w}_t$ since the statistics of noisy and noise free images differ considerably.

3.2 The Transition Model

To obtain a locally linear transition model we learn K constant transition matrices $\mathbf{A}^{(k)}$ and combine them using state dependent coefficients $\alpha^{(k)}(\mathbf{z_t})$, i.e.,

$$\mathbf{A}_t = \sum_{k=0}^{K} \alpha^{(k)}(\mathbf{z_t})\mathbf{A}^{(k)}.$$

A small neural network with softmax output is used to learn $\alpha^{(k)}$. Similar approaches are used in (Fraccaro et al., 2017; Karl et al., 2016).

Using a dense transition matrix in high-dimensional latent spaces is not feasible as it contains too many parameters and causes numerical instabilities and overfitting, as preliminary experiments

showed. Therefore, we design each $\mathbf{A}^{(k)}$ to consist of four band matrices

$$\mathbf{A}^{(k)} = \begin{pmatrix} \mathbf{B}_{11}^{(k)} & \mathbf{B}_{12}^{(k)} \\ \mathbf{B}_{21}^{(k)} & \mathbf{B}_{22}^{(k)} \end{pmatrix}.$$

with bandwidth b. This choice reduces the number of parameters while not affecting performance since the network is free to choose the state representation.

Moreover, it is crucial to correctly initialize the transition matrix. Initially, the transition model should focus on copying the encoder output so that the encoder can learn how to extract good features if observations are available and useful. On the other hand it is crucial that $\mathbf{A}$ does not yield an instable system. We choose $\mathbf{B}_{11}^{(k)} = \mathbf{B}_{22}^{(k)} = \mathbf{I}$ and $\mathbf{B}_{12}^{(k)} = -\mathbf{B}_{21}^{(k)} = 0.2 \cdot \mathbf{I}$.

3.3 Factorized Covariance Representation

Since the RKN learns high-dimensional representations, we can not work with the full covariance matrices $\mathbf{\Sigma}_t^+$, $\mathbf{\Sigma}_t^-$, $\mathbf{\Sigma}_t^{\text{obs}}$ and $\mathbf{\Sigma}^{\text{trans}}$. Instead, we represent $\mathbf{\Sigma}_t^{\text{obs}}$ and $\mathbf{\Sigma}^{\text{trans}}$ by diagonal matrices and denote the vector containing the diagonal values as $\boldsymbol{\sigma}_t^{\text{obs}}$ and $\boldsymbol{\sigma}_t^{\text{trans}}$ respectively.

However, we can not fully factorize state covariances by diagonal matrices as this neglects the correlation between the memory and the observation parts. As the memory part is excluded from the observation model $\mathbf{H}$, the Kalman update step would not change the memory nodes nor their uncertainty if we would only use a diagonal covariance matrix $\mathbf{\Sigma}_t^+$. Hence, for each observation node p_i, we compute the covariance with its corresponding memory node m_i. All the other covariances are neglected. This might be a crude approximation for many systems, however, as the state representation is not fixed for our approach but learned by the network, it is free to choose a state representation where such a factorization of the covariance matrix works well in practice. Thus, we use matrices of the form

$$\mathbf{\Sigma}_t = \begin{pmatrix} \mathbf{\Sigma}_t^{\text{upper}} & \mathbf{\Sigma}_t^{\text{side}} \\ \mathbf{\Sigma}_t^{\text{side}} & \mathbf{\Sigma}_t^{\text{lower}} \end{pmatrix},$$

where each of $\mathbf{\Sigma}_t^{\text{upper}}, \mathbf{\Sigma}_t^{\text{side}}, \mathbf{\Sigma}_t^{\text{lower}} \in \mathbb{R}^{m \times m}$ is a diagonal matrix. Again, we denote the vectors containing the diagonal values by $\boldsymbol{\sigma}_t^{\text{upper}}, \boldsymbol{\sigma}_t^{\text{lower}}, \boldsymbol{\sigma}_t^{\text{side}}$.

3.4 The Recurrent Kalman Network

The RKN works with the same prediction and update steps as the classical Kalman filter. However, by exploiting the simple latent observation model $\mathbf{H} = \begin{pmatrix} \mathbf{I}_m & \mathbf{0}_{m \times (n-m)} \end{pmatrix}$ and the factorized covariances, the equations can be simplified significantly which results in a new type of recurrent neural network, that we call *Recurrent Kalman Network*.

While being straight forward, the full derivation of the simplified equations is rather lengthy, thus, we refer to the supplementary material. The simplifications have three effects that allow working with high dimensional state spaces while keeping numerical stability, computational efficiency and (relatively) low memory consumption.

First, the matrix inversion in the Kalman gain is avoided and instead only an element-wise division of two vectors is needed. Second, we avoid using the full state covariance matrices by working with the three vectors $\boldsymbol{\sigma}^{\text{upper}}, \boldsymbol{\sigma}^{\text{lower}}$ and $\boldsymbol{\sigma}^{\text{side}}$ instead. This factorization reduces the total amount of numbers to store per matrix from n^2 to $3m$. It also avoids all matrix multiplications in the update of the state covariances. Third, working with $\boldsymbol{\sigma}^{\text{side}}$ makes it trivial to ensure that the symmetry and positive definiteness of the state covariance is not affected by numerical issues.

Similar to the input gate in LSTMs (Hochreiter and Schmidhuber, 1997) and GRUs (Cho et al., 2014) the Kalman gain can be seen as a gate controlling how much the current observation influences the state. However, this gating explicitly depends on the uncertainty estimates of the latent state and observation and is computed in a principled manner.

Using sparse transition models allows working in higher dimensional spaces with considerably less parameters than LSTMs or GRUs. For a fixed bandwidth b and number of basis matrices k, the number of parameters of the RKN scales linear in the state size while it scales quadratically for LSTMs and GRUs.

Model	Log Likelihood	Model	Log Likelihood
RKN $(m = 15, b = 3, K = 15)$	6.182 ± 0.155	LSTM $m = 50$	5.773 ± 0.231
RKN $m = b = 15, K = 3)$	6.248 ± 0.1715	LSTM $m = 6$	6.019 ± 0.122
RKN $(m = 15, b = 3, K = 15,$ fc $)$	6.161 ± 0.23	GRU $m = 50$	5.649 ± 0.197
RKN $(m = b = 15, K = 15,$ fc $)$	6.197 ± 0.249	GRU $m = 8$	6.051 ± 0.145

Table 1: We can see that our approach outperforms the generic LSTM and GRU Baselines. The GRU with $m = 8$ and the LSTM with $m = 6$ where designed to have roughly the same amount of parameters as the RKN with $b = 3$. In the case where $m = b$ the RKN uses a full transition matrix. *fc* stands for full covariance matrix, i.e., we do not use factorization of the belief state.

Moreover, the RKN provides a principled method to deal with absent inputs by just omitting the update step and setting the posterior to the prior.

3.5 Loss and Training

The model is trained by maximizing the log-likelihood of the training data under the model output. Gradients are computed using (truncated) backpropagation through time (BPTT) (Werbos, 1990) and clipped. We optimize the objective using the Adam (Kingma and Ba, 2014) stochastic gradient descent optimizer with default parameters.

4 Evaluation and Experiments

A full listing of hyperparameters and data set specifications can be found in the supplementary material[1]. We compare to LSTM and GRU Baselines for which we replaced the RKN Transition Layer with generic LSTM and GRU Layers. Those were given the encoder output as inputs and have an internal state size of $2n$. The internal state was split into two equally large parts, the first one was used to compute the mean and the second to compute the variance.

In order to further demonstrate that our model is not only capable of producing reasonable uncertainty estimates but produces good state estimates in general we additionally executed some of the following experiments using the root mean squared error, ignoring the variance. The results can be found in the appendix.

4.1 Pendulum

We evaluated the RKN on a simple simulated pendulum with images of size 24×24 pixels as observations. Gaussian transition noise with standard deviation of $\sigma = 0.1$ was added to the angular velocity after each step. For a first experiment filtering in the presence of high observation noise was performed. The amount of noise varies between no noise at all and the whole image consisting of pure noise. Furthermore, the noise is correlated over time hence the model may only observe pure noise for several consecutive time steps. Details about the noise sampling process can be found in the appendix. We represent the pendulums angle θ_t as a two dimensional vector $\mathbf{s}_t = \boldsymbol{\theta}_t = (\sin(\theta_t), \cos(\theta_t))^T$ to avoid discontinuities. We compared different instances of our model to evaluate the effects of assuming sparse transition models and factorized state covariances. See Table 1 for the results. The results show that the assumptions not affect performance, while we need to learn less parameters and can use much more efficient computations using the factorized representation. Note that for the more complex experiments, that require a higher dimensional latent state space, we were not able to experiment with full covariance matrices due to lack of memory and massive computation times. Moreover, our results show that the RKN slightly outperforms LSTM and GRU in all settings. This difference becomes more prominent for the more complex experiments.

For a second experiment we randomly removed half of the images from the sequences and tasked the models with imputing those missing frames, i.e., we train the models to predict images instead of the

[1] A link to source code will be added here in the final version.

Model	Log Likelihood
RKN $m = 45$ $b = 3, k = 15$	11.51 ± 1.703
LSTM $m = 50$	7.5224 ± 1.564
LSTM $m = 12$	7.429 ± 1.307
GRU $m = 50$	7.541 ± 1.547
GRU $m = 12$	5.602 ± 1.468

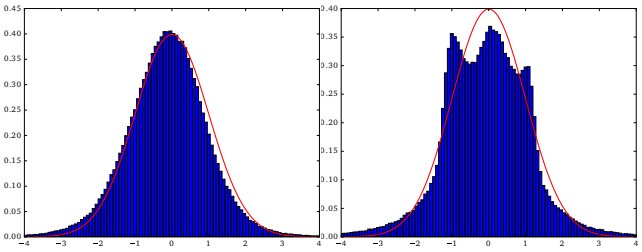


Figure 2: Results of the multiple pendulum experiments. To evaluate the quality of our uncertainty prediction we compute the normalized error $\frac{s^{(j)} - s^{(j),+}}{\sigma^{(j),+}}$ for each entry j of $\mathbf{s}$ for all time steps in all test sequences. This normalized error should follow a Gaussian distribution with unit variance if the prediction is correct. We compare the resulting error histograms with the a unit variance Gaussian. The left histogram shows the RKN, the right one the LSTM. The RKN perfectly fits the normal distribution while the LSTM's normalized error distribution has several modes. Again we designed the smaller LSTM and GRU to have roughly the same amount of parameters as the RKN.

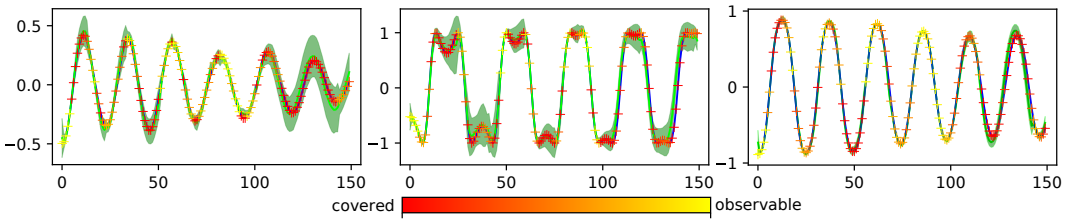


Figure 3: Predicted sine value of the tree links with 2 times standard deviation (green). Ground truth displayed in blue. The crosses visualize the current visibility of the link with yellow corresponding to fully visible and red to fully occluded. If there is no observation for a considerable time the predictions become less precise due to transition noise, however the variance also increases.

real states. We compared our approach to the Kalman Variational Autoencoder (KVAE) (Fraccaro et al., 2017), which shares many ideas to our approach but can not be used to predict positions $\mathbf{s}_t$ from images. Our approach achieved a log-likelihood of 1751 ± 15.1. Since the RKN and the KVAE optimize different objectives and the later has no notion of variance in the outputs, comparing their performance directly is not possible. We show a sample sequence in the supplementary material which show that the results are visually identical while our approach in addition provides reasonable variance estimates. All hyperparameters are the same as for the previous experiment.

4.2 Multiple Pendulums

Dealing with uncertainty in a principled manner becomes even more important if the observation noise affects different parts of the observation in different ways. In such a scenario the model has to infer which parts are currently observable and which parts are not. To obtain a simple experiment with that property, we repeated the pendulum experiments with three colored pendulums in one image. The image was split into four equally sized square parts and the noise generated individually for each part such that some pendulums may be occluded while others are visible. Exemplary images are shown in the supplementary material. A comparison to LSTM and GRU baselines can be found in Figure 2 and an exemplary trajectory in Figure 3.

4.3 Quad Link

We repeated the filtering experiment on a system with much more complicated dynamics, a quad link pendulum on images of size 48×48 pixels. Since the individual links of the quad link may occlude each other different amounts of noise are induced for each link. Two versions of this experiment were evaluated. One without additional noise and one were we added noise generated by the same process used in the pendulum experiments. You can find the results in Table 2.

Model	without noise Log Likelihood	with noise Log Likelihood
RKN $(m = 100, b = 3, K = 15)$	14.534 ± 0.176	6.259 ± 0.412
LSTM $(m = 50)$	11.960 ± 1.24	5.21 ± 0.305
LSTM $(m = 100)$	7.858 ± 4.680	3.87 ± 0.938
GRU $(m = 50)$	10.346 ± 2.70	4.696 ± 0.699
GRU $(m = 100)$	5.82 ± 2.80	1.2 ± 1.105

Table 2: Comparison of our approach with the LSTM and GRU Baselines on the Quad Link Pendulum. Again the RKN performs significantly better than LSTM and GRU who fail to perform well

Furthermore, we repeated the imputation experiment with the quad link and again compared to the KVAE. Our approach achieves a Log-Likelihood of 4272 ± 47. Sample images can be found in the appendix.

4.4 KITTI Dataset for Visual Odometry

We evaluated the RKN on the KITTI Data set for visual odometry (Geiger et al., 2012). Following (Zhao et al., 2018) we constructed an encoder consisting of FlowNet2 (Ilg et al., 2017), four convolutional and two branches of fully connected layers.

Following the standard KITTI evaluation procedure achieve an transnational error of 4.24% and a rotational error of 0.0404rad /100m . Those result are competitive to other recent results on the KITTI dataset using only monocular images (Zhao et al., 2018; Wang et al., 2018) and demonstrate the applicability of our approach to real world applications.

5 Conclusion

In this paper, we introduced the *Recurrent Kalman network* that jointly learns high-dimensional representations of the system in a latent space with locally linear transition models and factorized covariances. The update equations in the high-dimensional space are based on the update equations of the classical Kalman filter, however due to simplifying assumptions they can be performed much faster and with greater numerical stability. Furthermore, our experiments show that the models performance does not suffer from the assumptions. In fact our model outperforms generic LSTMs and GRUs on various state estimation tasks while providing reasonable uncertainty estimates. Training is straight forward and can be done in an end-to-end fashion.

In future work we want to exploit the principled notion of a variance provided by our approach in scenarios where such a notion is beneficial, e.g. reinforcement learning. Similar to (Fraccaro et al., 2017) we could expand our approach to not just filter but smooth over trajectories in offline post-processing scenarios which could potentially increase the estimation performance significantly. While initializing the latent state as an all zeros vector and a high covariance works well it is clearly not optimal. Karl et al. (2016) for example use a bidirectional LSTM to learn an initialization for their filter given the whole sequence.

References

Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop kf: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems*, pages 4376–4384, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

G.H.W. Gebhardt, A.G. Kupcsik, and G. Neumann. The kernel kalman rule - efficient nonparametric inference with recursive least squares. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Gerald L Smith, Stanley F Schmidt, and Leonard A McGee. *Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle*. National Aeronautics and Space Administration, 1962.

Simon J Julier and Jeffrey K Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Int. symp. aerospace/defense sensing, simul. and controls*, volume 3, pages 182–193. Orlando, FL, 1997.

Pierre Del Moral. Non-linear filtering: interacting particle resolution. *Markov processes and related fields*, 2(4):555–581, 1996.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

Marco Fraccaro, Simon Kamronn, Ulrich Paquet, and Ole Winther. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in Neural Information Processing Systems*, pages 3601–3610, 2017.

Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Huseyin Coskun, Felix Achilles, Robert DiPietro, Nassir Navab, and Federico Tombari. Long short-term memory kalman filters: Recurrent neural estimators for pose regularization. *arXiv preprint arXiv:1708.01885*, 2017.

Le Song, Kenji Fukumizu, and Arthur Gretton. Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models. *IEEE Signal Processing Magazine*, 30(4):98–111, 2013.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

Cheng Zhao, Li Sun, Pulak Purkait, Tom Duckett, and Rustam Stolkin. Learning monocular visual odometry with dense 3d mapping from dense 3d flow. *arXiv preprint arXiv:1803.02286*, 2018.

Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE conference on computer vision and pattern recognition (CVPR)*, volume 2, page 6, 2017.

Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. End-to-end, sequence-to-sequence probabilistic visual odometry through deep neural networks. *The International Journal of Robotics Research*, 37(4-5):513–542, 2018. doi: 10.1177/0278364917734298. URL https://doi.org/10.1177/0278364917734298.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

APPENDIX

A SIMPLIFIED KALMAN FILTER FORMULAS

As stated above the simple latent observation model $\mathbf{H} = \begin{pmatrix}\mathbf{I}_m & \mathbf{0}_{m\times(n-m)}\end{pmatrix}$, as well as the assumed factorization of the covariance matrices allow us to simplify the Kalman Filter equations.

A.1 NOTATION

In the following derivations we neglect the time indices t and $t+1$ and abbreviate the superscripts upper, lower and side with u, l and s respectively. For any matrix $\mathbf{M}$, $\hat{\mathbf{M}}$ denotes a diagonal matrix with the same diagonal as $\mathbf{M}$, $\mathbf{m}$ denotes a vector containing those diagonal elements and M_{ij} denotes the entry at row i and column j. Similarly, v_i denotes the i-th entry of a vector $\mathbf{v}$. The point wise product between two vectors of same length (Hadamat Product) will be denoted by $\odot$ and the point wise division by $\oslash$.

A.2 PREDICTION STEP

$$\begin{aligned}
&\text{Mean:} && \mathbf{z}^- = \mathbf{A}\mathbf{z}^+ \\
&\text{Covariance:} && \hat{\mathbf{\Sigma}}^- = \mathbf{A}\hat{\mathbf{\Sigma}}^+\mathbf{A}^T + \hat{\mathbf{\Sigma}}^{\mathrm{trans}}
\end{aligned}$$

The computation of the mean can not be further simplified, however, depending on the state size and bandwidth, sparse matrix multiplications may be exploited. For the covariance, let $\mathbf{T} = \mathbf{A}\mathbf{\Sigma}^+$. Then,

$$\mathbf{T} = \mathbf{A}\mathbf{\Sigma}^+ = \begin{pmatrix}\mathbf{B}_{11} & \mathbf{B}_{12}\\ \mathbf{B}_{21} & \mathbf{B}_{22}\end{pmatrix}\begin{pmatrix}\hat{\mathbf{\Sigma}}^{\mathrm{u},+} & \hat{\mathbf{\Sigma}}^{\mathrm{s},+}\\ \hat{\mathbf{\Sigma}}^{\mathrm{s},+} & \hat{\mathbf{\Sigma}}^{\mathrm{l},+}\end{pmatrix} = \begin{pmatrix}\mathbf{B}_{11}\hat{\mathbf{\Sigma}}^{\mathrm{u},+} + \mathbf{B}_{12}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} & \mathbf{B}_{11}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} + \mathbf{B}_{12}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\\ \mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{u},+} + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} & \mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\end{pmatrix}$$

and

$$\begin{aligned}
\mathbf{\Sigma}^- &= \mathbf{T}\mathbf{A}^T + \mathbf{\Sigma}^{\mathrm{trans}} \\
&= \begin{pmatrix}\mathbf{B}_{11}\hat{\mathbf{\Sigma}}^{\mathrm{u},+} + \mathbf{B}_{12}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} & \mathbf{B}_{11}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} + \mathbf{B}_{12}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\\ \mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{u},+} + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} & \mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\end{pmatrix}\begin{pmatrix}\mathbf{B}_{11}^T & \mathbf{B}_{21}^T\\ \mathbf{B}_{12}^T & \mathbf{B}_{22}^T\end{pmatrix} + \mathbf{\Sigma}^{\mathrm{trans}} = \begin{pmatrix}\hat{\mathbf{\Sigma}}^{\mathrm{u},-} & \hat{\mathbf{\Sigma}}^{\mathrm{s},-}\\ \hat{\mathbf{\Sigma}}^{\mathrm{s},-} & \hat{\mathbf{\Sigma}}^{\mathrm{l},-}\end{pmatrix},
\end{aligned}$$

with

$$\begin{aligned}
\mathbf{\Sigma}^{\mathrm{u},-} &= \left(\mathbf{B}_{11}\hat{\mathbf{\Sigma}}^{\mathrm{u},+} + \mathbf{B}_{12}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\right)\mathbf{B}_{11}^T + \left(\mathbf{B}_{11}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} + \mathbf{B}_{12}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\right)\mathbf{B}_{12}^T + \hat{\mathbf{\Sigma}}^{\mathrm{u,trans}} \\
&= \mathbf{B}_{11}\hat{\mathbf{\Sigma}}^{\mathrm{u},+}\mathbf{B}_{11}^T + \mathbf{B}_{12}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\mathbf{B}_{11}^T + \mathbf{B}_{11}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\mathbf{B}_{12}^T + \mathbf{B}_{12}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\mathbf{B}_{12}^T + \hat{\mathbf{\Sigma}}^{\mathrm{u,trans}} \\
\mathbf{\Sigma}^{\mathrm{l},-} &= \left(\mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{u},+} + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\right)\mathbf{B}_{21}^T + \left(\mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\right)\mathbf{B}_{22}^T + \hat{\mathbf{\Sigma}}^{\mathrm{l,trans}}, \\
&= \mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{u},+}\mathbf{B}_{21}^T + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\mathbf{B}_{21}^T + \mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\mathbf{B}_{22}^T + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\mathbf{B}_{22}^T + \hat{\mathbf{\Sigma}}^{\mathrm{l,trans}}, \\
\mathbf{\Sigma}^{\mathrm{s},-} &= \left(\mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{u},+} + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\right)\mathbf{B}_{11}^T + \left(\mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{s},+} + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\right)\mathbf{B}_{12}^T + \hat{\mathbf{\Sigma}}^{\mathrm{s,trans}}, \\
&= \mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{u},+}\mathbf{B}_{11}^T + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\mathbf{B}_{11}^T + \mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\mathbf{B}_{12}^T + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\mathbf{B}_{12}^T + \hat{\mathbf{\Sigma}}^{\mathrm{s,trans}}.
\end{aligned}$$

Since we are only interested in the diagonal parts of $\mathbf{\Sigma}^{\mathrm{u},-}$, $\mathbf{\Sigma}^{\mathrm{l},-}$ and $\mathbf{\Sigma}^{\mathrm{s},-}$ i.e. $\hat{\mathbf{\Sigma}}^{\mathrm{u},-}$, $\hat{\mathbf{\Sigma}}^{\mathrm{l},-}$ and $\hat{\mathbf{\Sigma}}^{\mathrm{s},-}$, we can further simplify these equations by realizing two properties of the terms above. First, for any matrix $\mathbf{M}, \mathbf{N}$ and a diagonal matrix $\hat{\mathbf{\Sigma}}$ it holds that

$$\left(\mathbf{M}\hat{\mathbf{\Sigma}}\mathbf{N}^T\right)_{ii} = \sum_{k=1}^{n} A_{ik}B_{ik}\sigma_k = \left(\mathbf{N}\hat{\mathbf{\Sigma}}\mathbf{M}^T\right)_{ii}.$$

Hence, we can simplify the equations for the upper and lower part to

$$\begin{aligned}
\mathbf{\Sigma}^{u,-} &= \mathbf{B}_{11}\hat{\mathbf{\Sigma}}^{\mathrm{u},+}\mathbf{B}_{11}^T + 2\cdot\mathbf{B}_{12}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\mathbf{B}_{11}^T + \mathbf{B}_{12}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\mathbf{B}_{12}^T + \hat{\mathbf{\Sigma}}^{\mathrm{u},trans}, \\
\mathbf{\Sigma}^{\mathrm{l},-} &= \mathbf{B}_{21}\hat{\mathbf{\Sigma}}^{\mathrm{u},+}\mathbf{B}_{21}^T + 2\cdot\mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{s},+}\mathbf{B}_{21}^T + \mathbf{B}_{22}\hat{\mathbf{\Sigma}}^{\mathrm{l},+}\mathbf{B}_{22}^T + \hat{\mathbf{\Sigma}}^{\mathrm{l,trans}}.
\end{aligned}$$

Second, since we are only interested in the diagonal of the result it is sufficient to compute only the diagonals of the individual parts of the sums which are almost all of the same structure i.e. $\mathbf{S} = \mathbf{M}\hat{\mathbf{\Sigma}}^{+}\mathbf{N}^{T}$. Let $\mathbf{T} = \mathbf{M}\hat{\mathbf{\Sigma}}^{+}$, then each element of $\mathbf{T}$ can be computed as

$$T_{ij} = \sum_{k=1}^{n} M_{ik}\hat{\Sigma}_{kj} = M_{ij}\sigma_j.$$

Consequently, the elements of $\mathbf{S} = \mathbf{T}\mathbf{A}^{T}$ can be computed as

$$S_{ij} = \sum_{k=1}^{n} T_{ik}A^{T}_{kj} = \sum_{k=1}^{n} M_{ik}\sigma_k N_{jk}.$$

Ultimately, we are not interested in $\mathbf{S}$ but only in $\hat{\mathbf{S}}$

$$\hat{S}_{ii} = \sum_{k=1}^{n} M_{ik}N_{ik}\sigma_k,$$

which can efficiently be implemented using point-wise matrix multiplication and sum reduction.

A.3 Update Step

Kalman Gain	$\mathbf{Q} = \mathbf{\Sigma}^{-}\mathbf{H}^{T}\left(\mathbf{H}\mathbf{\Sigma}^{-}\mathbf{H}^{T} + \mathbf{\Sigma}^{\text{obs}}\right)^{-1}$
Mean	$\mathbf{z}^{+} = \mathbf{z}^{-} + \mathbf{Q}\left(\mathbf{w} - \mathbf{H}\mathbf{z}^{-}\right)$
Covariance	$\mathbf{\Sigma}^{+} = \left(\mathbf{I} - \mathbf{Q}\mathbf{H}\right)\mathbf{\Sigma}^{-}$

First, note that

$$\mathbf{\Sigma}^{-}\mathbf{H}^{T} = \begin{pmatrix}\hat{\mathbf{\Sigma}}^{\text{u},-}\\ \hat{\mathbf{\Sigma}}^{\text{s},-}\end{pmatrix} \quad \text{and} \quad \mathbf{H}\mathbf{\Sigma}^{-}\mathbf{H}^{T} + \hat{\mathbf{\Sigma}}^{\text{obs}} = \hat{\mathbf{\Sigma}}^{u,-} + \hat{\mathbf{\Sigma}}^{\text{obs}}$$

and thus the computation of the Kalman Gain only involves diagonal matrices. Hence the Kalman Gain matrix also consists of two diagonal matrices, i.e., $\mathbf{Q} = \begin{pmatrix}\hat{\mathbf{Q}}^{\text{u}}\\ \hat{\mathbf{Q}}^{\text{l}}\end{pmatrix}$ whose diagonals can be computed by

$$\mathbf{q}^{\text{u}} = \boldsymbol{\sigma}^{\text{u},-} \oslash \left(\boldsymbol{\sigma}^{\text{u},-} + \boldsymbol{\sigma}^{\text{obs}}\right) \text{ and } \mathbf{q}^{\text{l}} = \boldsymbol{\sigma}^{\text{s},-} \oslash \left(\boldsymbol{\sigma}^{\text{u},-} + \boldsymbol{\sigma}^{\text{obs}}\right).$$

Using this result, the mean update can be simplified to

$$\mathbf{z}^{+} = \mathbf{z}^{-} + \begin{pmatrix}\mathbf{q}_{\text{u}}\\ \mathbf{q}_{\text{l}}\end{pmatrix} \odot \begin{pmatrix}\mathbf{w} - \mathbf{z}^{\text{u},-}\\ \mathbf{w} - \mathbf{z}^{\text{u},-}\end{pmatrix}.$$

For the covariance we get:

$$\begin{aligned}\begin{pmatrix}\hat{\mathbf{\Sigma}}^{\text{u},+} & \hat{\mathbf{\Sigma}}^{\text{s},+}\\ \hat{\mathbf{\Sigma}}^{\text{s},+} & \hat{\mathbf{\Sigma}}^{\text{l},+}\end{pmatrix} &= \left(\mathbf{I}_n - \mathbf{Q}\mathbf{H}\right)\mathbf{\Sigma}^{-} = \begin{pmatrix}\mathbf{I}_m - \hat{\mathbf{Q}}^{\text{u}} & \mathbf{0}_{m\times m}\\ -\hat{\mathbf{Q}}^{\text{l}} & \mathbf{I}_m\end{pmatrix}\begin{pmatrix}\hat{\mathbf{\Sigma}}^{\text{u},-} & \hat{\mathbf{\Sigma}}^{\text{s},-}\\ \hat{\mathbf{\Sigma}}^{\text{s},-} & \hat{\mathbf{\Sigma}}^{\text{l},-}\end{pmatrix}\\ &= \begin{pmatrix}\left(\mathbf{I}_m - \hat{\mathbf{Q}}^{\text{u}}\right)\hat{\mathbf{\Sigma}}^{\text{u},-} & \left(\mathbf{I}_m - \hat{\mathbf{Q}}^{\text{u}}\right)\hat{\mathbf{\Sigma}}^{\text{s},-}\\ -\hat{\mathbf{Q}}^{\text{l}}\hat{\mathbf{\Sigma}}^{\text{u},-} + \hat{\mathbf{\Sigma}}^{\text{s},-} & -\hat{\mathbf{Q}}^{\text{l}}\hat{\mathbf{\Sigma}}^{\text{s},-} + \hat{\mathbf{\Sigma}}^{\text{l},-}\end{pmatrix}.\end{aligned}$$

Hence the diagonals of the individual parts can be computed as

$$\begin{aligned}\boldsymbol{\sigma}^{\text{u},+} &= \left(\mathbf{1}_m - \mathbf{q}^{\text{u}}\right) \odot \boldsymbol{\sigma}^{\text{u},-}\\ \boldsymbol{\sigma}^{\text{s},+} &= \left(\mathbf{1}_m - \mathbf{q}^{\text{u}}\right) \odot \boldsymbol{\sigma}^{\text{s},-}\\ \boldsymbol{\sigma}^{\text{l},+} &= \boldsymbol{\sigma}^{\text{l},-} - \mathbf{q}^{\text{l}} \odot \boldsymbol{\sigma}^{\text{s},-}.\end{aligned}$$

Model	Root Mean Squared Error
Pendulum	
RKN ($m = 15, b = 3, K = 15$)	0.0779 ± 0.0082
RKN ($m = b = 15, K = 15$)	0.0758 ± 0.0094
LSTM ($m = 50$)	0.0920 ± 0.0774
LSTM ($m = 6$)	0.0959 ± 0.0100
GRU ($m = 50$)	0.0821 ± 0.0084
GRU ($m = 8$)	0.0916 ± 0.0087
Multiple Pendulums	
RKN ($m = 45, b = 3, k = 15$)	0.0878 ± 0.0036
LSTM ($m = 50$)	0.098 ± 0.0036
LSTM ($m = 12$)	0.104 ± 0.0043
GRU ($m = 50$)	0.112 ± 0.0371
GRU ($m = 14$)	0.105 ± 0.0055
Quad Link (without additional noise)	
RKN ($m = 100, b = 3, k = 15$)	$0.118 + -0.0016$
LSTM ($m = 100$)	0.175 ± 0.182
LSTM ($m = 25$)	0.118 ± 0.0049
GRU ($m = 100$)	0.278 ± 0.105
GRU ($m = 25$)	0.121 ± 0.0021

Table 3: RMSE Results

B Root Mean Square Error Results

To evaluate the actual prediction performance of our approach we repeated some experiments using the RMSE as loss function. Other than that and removing the variance output of the decoder no changes were made to the model, hyperparameters and learning procedure. The results can be found in Table 3.

C Visualization of Imputation Results

Exemplary results of the data imputation experiments conducted for the Pendulum and Quad Link experiment can be found in Figure 4

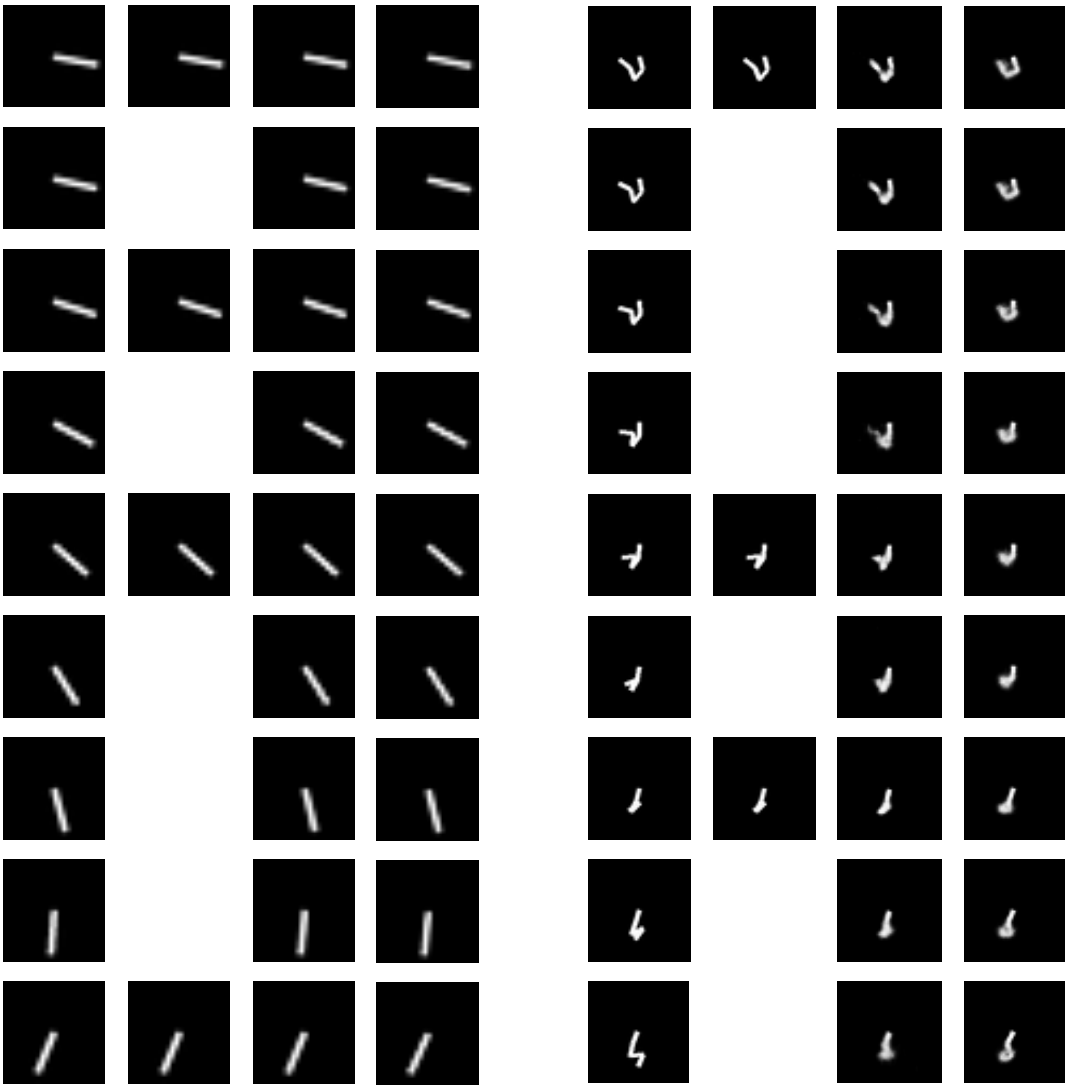
D Network Architectures and Hyper Parameters

For all experiments Adam (Kingma and Ba, 2014) with default parameters ($\alpha = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$) was used as an optimizer. The gradients were computed using (truncated) Backpropagation Trough Time (BpTT) (Werbos, 1990). Further in all (transposed) convolutional layers layer normalization (LN) (Ba et al., 2016) was employed to normalize the filter responses. "Same" padding was used. The elu activation function (Clevert et al., 2015) plus a constant 1 is denoted by $(\text{elu} + 1)$ was used to ensure that the variance outputs are positive.

D.1 Pendulum and Multiple Pendulum Experiments

Observations *Pendulum*: Grayscale images of size 24×24 pixels. *Multiple Pendulum*: RGB images of size 24×24 pixels. See Figure 5 for examples.
Dataset: 1000 Train and 500 Test sequences of length 150. For the filtering experiments noise according to section E was added , for imputation 50% of the images were removed randomly.
Encoder: 2 convolution + 1 fully connected + linear output & $(\text{elu} + 1)$ output:

- Convolution 1: 12, 5×5 filter, ReLU, 2×2 max pool with 2×2 stride
- Convolution 2: 12, 3×3 filter with 2×2 stride, ReLU, 2×2 max pool with 2×2 stride
- *Pendulum*: Fully Connected 1: 30, ReLU
- *Multiple Pendulum*: Fully Connected 1: 90, ReLU



(a) Pendulum (b) Quad Link

Figure 4: Each of (a) and (b) shows from left to right: true images, input to the models, imputation results for RKF, imputation results for KVAE(Fraccaro et al., 2017).

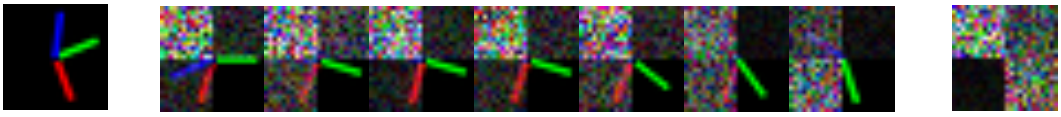


Figure 5: Example images for the multiple pendulum experiments. **Left**: Noise free image. **Middle**: sequence of images showing how the noise affects different pendulums differently. **Right**: Image without useful information.

Transition Model *Pendulum*: 15 dimensional latent observation, 30 dimensional latent state. *Multiple Pendulum*: 45 dimensional latent observation, 90 dimensonal latent state. *Both:* bandwidth: 3, number of basis: 15

- $\alpha(\mathbf{z}_t)$: No hidden layers - softmax output

Decoder (for $\mathbf{s}_t^+$): 1 fully connected + linear output:

- Fully Connected 1: 10, ReLU

Decoder (for $\mathbf{o}_t^+$): 1 fully connected + 2 transposed convolution + transposed convolution output:

- Fully Connected 1: 144 ReLU
- Transposed Convolution 1: 16, 5×5 filter with 4×4 stride, ReLU
- Transposed Convolution 2: 12, 3×3 filter with 2×2 stride, ReLU
- Transposed Convolution Out: *Pendulum*: 1 *Multiple Pendulum*: 3, 3×3 filter with 1×1 stride, Sigmoid

Decoder (for $\boldsymbol{\sigma}_t^+$ or σ_t^+): 1 fully connected + $(\text{elu} + 1)$:

- Fully Connected 1: 10, ReLU

D.2 Quad Link

Observations: Grayscale images of size $48x48$ pixels.
Dataset: 4000 Train and 1000 Test sequences of length 150. For the filtering with additional noise experiments noise according to section D was added, for imputation 50% of the images were removed randomly.
Encoder: 2 convolution + 1 fully connected + linear output & $(\text{elu} + 1)$ output:

- Convolution 1: 12, 5×5 filter with 2×2 stride, ReLU, 2×2 max pool with 2×2 stride
- Convolution 2: 12, 3×3 filter with 2×2 stride, ReLU, 2×2 max pool with 2×2 stride
- Fully Connected 1: 200 ReLU

Transition Model 100 dimensional latent observation, 200 dimensional latent state, bandwidth: 3, number of basis: 15

- $\alpha(\mathbf{z}_t)$: No hidden layers - softmax output

Decoder (for $\mathbf{s}_t^+$): 1 fully connected + linear output:

- Fully Connected 1: 10, ReLU

Decoder (for $\mathbf{o}_t^+$): 1 fully connected + 2 transposed convolution + transposed convolution output:

- Fully Connected 1: 144 ReLU
- Transposed Convolution 1: 16, 5×5 filter with 4×4 stride, ReLU
- Transposed Convolution 2: 12, 3×3 filter with 4×4 stride, ReLU
- Transposed Convolution Out: 1, 1×1 stride, Sigmoid

Decoder (for $\boldsymbol{\sigma}_t^+$ or σ_t^+): 1 fully connected + $(\text{elu} + 1)$:

- Fully Connected 1: 10, ReLU

D.3 Kitti

Observation and Data Set: Images from the KITTI (Geiger et al., 2012) dataset for visual odometry. Sequences 00, 01, 02, 08, 09 used for training. Sequences 03, 04, 05, 06, 07, 10 for testing.

Encooder: FlowNet2 + 4 Convolutional + 2 Branches of dense Layers. See (Zhao et al., 2018)
Transition Model: 100 dimensional latent observations, 200 dimensional latent state, bandwidth: 5, number of basis: 20

- $\alpha(\mathbf{z}_t)$: No hidden layers - softmax output

Decoder (for $\mathbf{s}_t^+$): 2 fully connected + linear output:

- Fully Connected 1: 128, ReLU
- Fully Connected 1: 128, ReLU

Decoder (for $\boldsymbol{\sigma}_t^+$ or σ_t^+): 2 fully connected + $(\text{elu} + 1)$:

- Fully Connected 1: 128, ReLU
- Fully Connected 1: 128, ReLU

E OBSERVATION NOISE GENERATION PROCESS

Let $\mathcal{U}(x, y)$ denote the uniform distribution from x to y.To generate the noise for the pendulum task for each sequence a sequence of factors f_t of same length was generated. To correlate the factors they were sampled as $f_0 \sim \mathcal{U}(0, 1)$ and $f_{t+1} = \min(\max(0, f_t + r_t), 1)$ with $r_t \sim \mathcal{U}(-0.2, 0.2)$. Afterwards, for each sequence two thresholds $t_1 \sim \mathcal{U}(0.0, 0.25)$ and $t_2 \sim \mathcal{U}(0.75, 1)$ were sampled. All $f_t < t_1$ were set to 0, all $f_t > t_2$ to 1 and the rest was linearly mapped to the interval $[0, 1]$. Finally, for each image $\mathbf{i}_t$ an image consisting of pure uniformly distributed noise $\mathbf{i}_t^{noise}$ was sampled and the observation computed as $\mathbf{o}_t = f_t \cdot \mathbf{i}_t + (1 - f_t) \cdot \mathbf{i}_t^{noise}$.