Autonomous Robots
https://doi.org/10.1007/s10514-021-09990-9

How to train your differentiable filter

Alina Kloss[1] · **Georg Martius**[1] · **Jeannette Bohg**[2]

Received: 15 December 2020 / Accepted: 27 May 2021
© The Author(s) 2021

Abstract
In many robotic applications, it is crucial to maintain a belief about the state of a system, which serves as input for planning and decision making and provides feedback during task execution. Bayesian Filtering algorithms address this state estimation problem, but they require models of process dynamics and sensory observations and the respective noise characteristics of these models. Recently, multiple works have demonstrated that these models can be learned by end-to-end training through differentiable versions of recursive filtering algorithms. In this work, we investigate the advantages of *differentiable filters* (DFs) over both unstructured learning approaches and manually-tuned filtering algorithms, and provide practical guidance to researchers interested in applying such differentiable filters. For this, we implement DFs with four different underlying filtering algorithms and compare them in extensive experiments. Specifically, we (i) evaluate different implementation choices and training approaches, (ii) investigate how well complex models of uncertainty can be learned in DFs, (iii) evaluate the effect of end-to-end training through DFs and (iv) compare the DFs among each other and to unstructured LSTM models.

Keywords Differentiable filters · State estimation · Learned noise models · Learning and structure

1 Introduction

In many robotic applications, it is crucial to maintain a belief about the state of the system over time, like tracking the location of a mobile robot or the pose of a manipulated object. These state estimates serve as input for planning and decision making and provide feedback during task execution. In addition to tracking the system state, it can also be desirable to estimate the uncertainty associated with the state predictions. This information can be used to detect failures and enables risk-aware planning, where the robot takes more cautious actions when its confidence in the estimated state is low (Todorov 2005; Pontón et al. 2020).

The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Alina Kloss.

✉ Alina Kloss
Alina.Kloss@tuebingen.mpg.de

Georg Martius
Georg.Martius@tuebingen.mpg.de

Jeannette Bohg
bohg@stanford.edu

[1] Max Planck Institute for Intelligent Systems, Tübingen, Germany

[2] Stanford University, Stanford, USA

Recursive Bayesian filters are a class of algorithms that combine perception and prediction for probabilistic state estimation in a principled way. To do so, they require an observation model that relates the estimated state to the sensory observations and a process model that predicts how the state develops over time. Both have associated noise models that reflect the stochasticity of the underlying system and determine how much trust the filter places in perception and prediction.

Formulating good observation and process models for the filters can, however, be difficult in many scenarios, especially when the sensory observations are high-dimensional and complex, like camera images. Over the last years, deep learning has become the method of choice for processing such data. While (recurrent) neural networks can be trained to address the full state estimation problem directly, recent work (Jonschkowski and Brock 2016; Haarnoja et al. 2016; Jonschkowski et al. 2018; Karkus et al. 2018) showed that it is also possible to include data-driven models into Bayesian filters and train them end-to-end through the filtering algorithm. For Histogram filters (Jonschkowski and Brock 2016), Kalman filters (Haarnoja et al. 2016) and Particle filters (Jonschkowski et al. 2018; Karkus et al. 2018), the respective authors showed that such *differentiable filters* (DF) systematically outperform unstructured neural networks like LSTMs

Published online: 09 June 2021

(Hochreiter and Schmidhuber 1997). In addition, the end-to-end training of the models also improved the filtering performance compared to using observation and process models that had been trained separately.

A further interesting aspect of differentiable filters is that they allow for learning sophisticated models of the observation and process noise. This is useful because finding appropriate values for the noise models is often difficult and despite much research on identification methods (e.g. Bavdekar et al. (2011); Valappil and Georgakis (2000)) they are often tuned manually in practice. To reduce the tedious tuning effort, the noise is then typically assumed to be uncorrelated Gaussian noise with zero mean and *constant* covariance. Many real systems are, however, better described by *heteroscedastic* noise models, where the level of uncertainty depends on the state of the system and/or possible control inputs. Taking heterostochasticity of the dynamics into account has been demonstrated to improve filtering performance in many robotic tasks (Bauza and Rodriguez 2017; Kersting et al. 2007). Haarnoja et al. (2016) also show that learning heteroscedastic observation noise helps a Kalman filter dealing with occlusions during object tracking.

In this paper, we perform a thorough evaluation of differentiable filters. Our main goals are to highlight the advantages of DFs over both unstructured learning approaches and manually-tuned filtering algorithms, and to provide guidance to practitioners interested in applying differentiable filtering to their problems.

To this end, we review and implement existing work on differentiable Kalman and Particle filters and introduce two novel variants of differentiable Unscented Kalman filters. Our implementation for TensorFlow (Abadi et al. 2015) is publicly available.[1] In extensive experiments on three different tasks, we compare the DFs and evaluate different design choices for implementation and training, including loss functions and training sequence length. We also investigate how well the different filters can learn complex heteroscedastic and correlated noise models, evaluate how end-to-end training through the DFs influences the learned models and compare the DFs to unstructured LSTM models.

2 Related work

2.1 Combining learning and algorithms

Integrating algorithmic structure into learning methods has been studied for many robotic problems, including state estimation (Haarnoja et al. 2016; Jonschkowski and Brock 2016; Jonschkowski et al. 2018; Karkus et al. 2018; Ma et al. 2020), planning (Tamar et al. 2016; Karkus et al. 2017; Oh et al. 2017; Farquhar et al. 2018; Guez et al. 2018) and control (Donti et al. 2017; Okada et al. 2017; Amos et al. 2018; Pereira et al. 2018; Holl et al. 2020). Most notably, Karkus et al. (2019) combine multiple differentiable algorithms into an end-to-end trainable "Differentiable Algorithm Network" to address the complete task of navigating to a goal in a previously unseen environment using visual observations. Here, we focus on addressing the state estimation problem with differentiable implementations of Bayesian filters.

2.2 Differentiable Bayesian filters

There have been few works on differentiable filters so far. Haarnoja et al. (2016) propose the BackpropKF, a differentiable implementation of the (extended) Kalman filter. Jonschkowski and Brock (2016) present a differentiable Histogram filter for discrete localization tasks in one or two dimensions and Jonschkowski et al. (2018) and Karkus et al. (2018) both implement differentiable Particle filters for localization and tracking of a mobile robot. In the following, we focus our discussion on differentiable Kalman and Particle filters, since Histogram filters as used by Jonschkowski and Brock (2016) are usually not feasible in practice, due to the need of discretizing the complete state space.

Observation model and noise All three works have in common that the raw observations are processed by a learned neural network that can be trained end-to-end through the filter. In Haarnoja et al. (2016), the network outputs a low-dimensional representation of the observations together with input-dependent observation noise (see Sec. 4.2), while in Jonschkowski et al. (2018); Karkus et al. (2018), a neural network learns to predict the likelihood of the observations under each particle given an image and (in (Karkus et al. 2018)) a map of the environment.

As a result, all three works use heteroscedastic observation noise, but only Haarnoja et al. (2016) evaluate this choice: They show that conditioning the observation noise on the raw image observations drastically improves filter performance when the tracked object can be occluded.

Process model and noise For predicting the next state, all three works use a given analytical process model. While Haarnoja et al. (2016) and Karkus et al. (2018) also assume known process noise, Jonschkowski et al. (2018) train a network to predict it conditioned on the actions. The effect of learning action dependent process noise is, however, not evaluated.

Effect of end-to-end learning Jonschkowski et al. (2018) compare the results of an end-to-end trained filter with one where the observation model and process noise were trained separately. The end-to-end trained variant performs better, presumably because it learns to overestimate the process noise. Possible differences between the learned observation models are not discussed. The best performance for the filter

[1] https://github.com/akloss/differentiable_filters.

could be reached by first pretraining the models individually and the finetuning end-to-end through the filter.

Comparison to unstructured models All works compare their differentiable filters to LSTM models trained for the same task and find that including the structural priors of the filtering algorithm and the known process models improves performance. Jonschkowski et al. (2018) also evaluate a Particle filter with a learned process model in one experiment, which performs worse than the filter with an analytical process model but still beats the LSTM.

In contrast to the existing work on differentiable filtering, the main purpose of this paper is not to present a new method for solving a robotic task. Instead, we present a thorough evaluation of differentiable filtering and of implementation choices made by the aforementioned seminal works. We also implement two novel differentiable filters based on variants of the Unscented Kalman filter and compare the differentiable filters with different underlying Bayesian filtering algorithms in a controlled way.

2.3 Variational inference

A second line of research closely related to differentiable filters is variational inference in temporal state space models (Krishnan et al. 2016; Karl et al. 2017; Watter et al. 2015; Fraccaro et al. 2017; Archer et al. 2015). For a recent review of this work, see Girin et al. (2020). In contrast to DFs, the focus of this research lies more on finding generative models that explain the observed data sequences and are able to generate new sequences. The representation of the underlying state of the system is often not assumed to be known. But even though the goals are different, recent results in this field show that structuring the variational models similarly to Bayesian filters improves their performance (Karl et al. 2017; Fraccaro et al. 2017; Naesseth et al. 2018; Maddison et al. 2017; Le et al. 2018).

3 Bayesian filtering for state estimation

Filtering refers to the problem of estimating the latent state $\mathbf{x}$ of a stochastic dynamic system at time step t given an initial belief $\text{bel}(\mathbf{x}_0) = p(\mathbf{x}_0)$, a sequence of observations $\mathbf{z}_{1\ldots t}$ and actions $\mathbf{u}_{0\ldots t-1}$. Formally, we seek the posterior distribution $\text{bel}(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{x}_{0\ldots t-1}, \mathbf{u}_{0\ldots t-1}, \mathbf{z}_{1\ldots t})$.

Bayesian Filters make the Markov assumption, i.e. that the distributions of the future states and observations are conditionally independent from the history of past states and observations given the current state. This assumption makes it possible to compute the belief at time t recursively as

$$\begin{aligned}\text{bel}(\mathbf{x}_t) &= \eta\, p(\mathbf{z}_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1})\text{bel}(\mathbf{x}_{t-1})d\mathbf{x}_{t-1} \\ &= \eta\, p(\mathbf{z}_t|\mathbf{x}_t)\overline{\text{bel}}(\mathbf{x}_t)\end{aligned}$$

where η is a normalization factor. Computing $\overline{\text{bel}}(\mathbf{x}_t)$ is referred to as the *prediction step* of Bayesian filters, while updating the belief with $p(\mathbf{z}_t|\mathbf{x}_t)$ is called *(observation) update step*.

For the prediction step, the dynamics of the system is modeled by the *process model* f that describes how the state changes over time. The observation update step uses an *observation model* h that generates observations given the current state:

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \mathbf{q}_{t-1}) \qquad \mathbf{z}_t = h(\mathbf{x}_t, \mathbf{r}_t)$$

The random variables $\mathbf{q}$ and $\mathbf{r}$ are the process and observation noise and capture the stochasticity of the system.

In this paper, we investigate differentiable versions of four different nonlinear Bayesian filtering algorithms: The Extended Kalman Filter (EKF), the Unscented Kalman Filter (UKF), a sampling-based variant of the UKF that we call Monte Carlo Unscented Kalman Filter (MCUKF) and the Particle Filter (PF). We briefly review these algorithms in Online Material 1, Sec. A.

4 Implementation

In this section, we describe how we embed model-learning into differentiable versions of the aforementioned nonlinear filtering algorithms. These differentiable versions will be denoted by dEKF, dUKF etc. in the following.

4.1 Differentiable filters

We implement the filtering algorithms as recurrent neural network layers in TensorFlow. For UKF and MCUKF, this is straight-forward, since all necessary operations are differentiable and available in TensorFlow.

In contrast, the dEKF requires the Jacobian of the process model $\mathbf{F}$. TensorFlow implements a method for computing Jacobians, with or without vectorization. The former is fast but has a high memory demand, while the latter can become very slow for large batch sizes. Therefore, we recommend to derive the Jacobians manually where applicable.

4.1.1 dPF

The Particle filter is the only filter we investigate that is not fully differentiable: In the resampling step, a new set of particles with uniform weights is drawn (with replacement) from the old set according to the old particle weights.

While the drawn particles can propagate gradients to their ancestors, gradient propagation to other old particles or to the weights of the old particle set is disrupted (Jonschkowski et al. 2018; Karkus et al. 2018; Zhu et al. 2020). If we place the resampling step at the beginning of the per-timestep computations, this only affects the gradient propagation through time, i.e. from one timestep $t+1$ to its predecessor t. At time t, both particles and weights still receive gradient information about the corresponding loss at this timestep. We therefore hypothesize that the missing gradients through time are not problematic as long as we provide a loss at every timestep.

As an alternative to simply ignoring the disrupted gradients, we can also apply the resampling step less frequently or use soft resampling as proposed by Karkus et al. (2018). We evaluate these options in Sec. 6.2.5.

In addition, we investigate two alternative implementation choices for the dPF: The likelihood used for updating the particle weights in the observation update step can be implemented either with an analytical Gaussian likelihood function or with a trained neural network as in Jonschkowski et al. (2018) and Karkus et al. (2018). The learned observation likelihood is potentially more expressive than the analytical solution and can be advantageous for problems where formulating the observation and sensor model is not as straight-forward as in our experiments. A potential drawback is that in contrast to the analytical solution, no explicit noise model or sensor network is learned. We compare these two options in Sec. 6.2.4.

4.2 Observation model

In Bayesian filtering, the observation model $h(\cdot)$ is a *generative* model that predicts observations from the state $\mathbf{z}_t = h(\mathbf{x}_t)$. In practice, it is often hard to find such models that directly predict the potentially high-dimensional raw sensory signals without making strong assumptions.

We therefore use the method first proposed by Haarnoja et al. (2016) and train a *discriminative* neural network n_s with parameters $\mathbf{w}_s$ to preprocess the raw sensory data $\mathbf{D}$ and create a more compact representation of the observations $\mathbf{z} = n_s(\mathbf{D}, \mathbf{w}_s)$. This network can be seen as a virtual sensor, and we thus call it *sensor network*. In addition to $\mathbf{z}_t$, the sensor network can also predict the heteroscedastic observation noise covariance matrix $\mathbf{R}_t$ (see Sec. 4.4) for the current input $\mathbf{D}_t$.

In our experiments, $\mathbf{z}$ contains a subset of the state vector $\mathbf{x}$. The actual observation model $h(\mathbf{x})$ thus reduces to a simple linear selection matrix of the observable components, which we provide to the DFs.

4.3 Process model

Depending on the user's knowledge about the system, the process model $f(\cdot)$ for the prediction step can be implemented using a known analytical model or a neural network $n_p(\cdot)$ with weights $\mathbf{w}_p$. When using neural networks, we train $n_p(\cdot)$ to output the change from the last state $n_p(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_p) = \Delta\mathbf{x}_t$ such that $\mathbf{x}_{t+1} = \mathbf{x}_t + \Delta\mathbf{x}_t$. This form ensures stable gradients between timesteps (since $\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t} = 1 + \frac{\partial p}{\partial \mathbf{x}_t}$) and provides a reasonable initialization of the process model close to identity.

4.4 Noise models

For learning the observation and process noise, we consider two different conditions: constant and heteroscedastic. In both cases, we assume that the process and observation noise at time t can be described by zero-mean Gaussian distributions with covariance matrices $\mathbf{Q}_t$ and $\mathbf{R}_t$.

A common assumption in state-space modeling is that $\mathbf{Q}_t$ and $\mathbf{R}_t$ are diagonal matrices, but we can also use full covariance matrices to model correlated noise. In this case, we follow Haarnoja et al. (2016) and train the noise models to output upper-triangular matrices $\mathbf{L}_t$, such that e.g. $\mathbf{Q}_t = \mathbf{L}_t\mathbf{L}_t^T$. This form ensures that the resulting matrices are positive definite.

For constant noise, the filters directly learn the diagonal or triangular elements of $\mathbf{Q}$ and $\mathbf{R}$. In the heteroscedastic case, $\mathbf{Q}_t$ is predicted from the current state $\mathbf{x}_t$ and (if available) the control input $\mathbf{u}_t$ by a neural network $n_q(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_q)$ with weights $\mathbf{w}_q$. In dUKF, dMCUKF and dPF, $n_q(\cdot)$ outputs separate $\mathbf{Q}^i$ for each sigma point/particle and $\mathbf{Q}_t$ is computed as their weighted mean. The heteroscedastic observation noise covariance matrix $\mathbf{R}_t$ is an additional output of the sensor model $n_s(\mathbf{D}_t, \mathbf{w}_s)$.

We initialize the diagonals $\mathbf{Q}_t$ and $\mathbf{R}_t$ close to given target values by adding a trainable bias variable to the output of the noise models. To prevent numerical instabilities, we also add a small fixed bias to the diagonals as a lower bound for the predicted noise.

4.5 Loss function

For training the filters, we always assume that we have access to the ground truth trajectory of the state $\mathbf{x}^l_{t=0\ldots T}$. In our experiments, we test the two different loss functions used in related work: The first, used by Karkus et al. (2018) is simply the mean squared error (MSE) between the mean of the belief and true state at each timestep:

$$L_{\text{MSE}} = \frac{1}{T}\sum_{t=0}^{T}(\mathbf{x}^l_t - \boldsymbol{\mu}_t)^T(\mathbf{x}^l_t - \boldsymbol{\mu}_t). \tag{1}$$

For the dPF, we compute $\boldsymbol{\mu}$ as the weighted mean of the particles.

The second loss function, used by Haarnoja et al. (2016) and Jonschkowski et al. (2018), is the negative log likelihood (NLL) of the true state under the predicted distribution of the belief. In dEKF, dUKF and dMCUKF, the belief is represented by a Gaussian distribution with mean $\boldsymbol{\mu}_t$ and covariance $\boldsymbol{\Sigma}_t$ and the negative log likelihood is computed as

$$L_{\text{NLL}} = \frac{1}{2T}\sum_{t=0}^{T} \log(|\boldsymbol{\Sigma}_t|) + (\mathbf{x}_t^l - \boldsymbol{\mu}_t)^T \boldsymbol{\Sigma}_t^{-1}(\mathbf{x}_t^l - \boldsymbol{\mu}_t). \quad (2)$$

The dPF represents its belief using the particles $\boldsymbol{\chi}_i \in \mathbf{X}$ and their weights π_i. We consider two alternative ways of calculating the NLL for training the dPF: The first is to represent the belief by fitting a single Gaussian to the particles, with $\boldsymbol{\mu} = \sum_{i=0}^{N} \pi_i \boldsymbol{\chi}_i$ and $\boldsymbol{\Sigma} = \sum_{i=0}^{N} \pi_i (\boldsymbol{\chi}_i - \boldsymbol{\mu})(\boldsymbol{\chi}_i - \boldsymbol{\mu})^T$ and then apply Eq. 2. We refer to this variant as dPF-G.

However, this is only a good representation of the belief if the distribution of the particles is unimodal. To better reflect the potential multimodality of the particle distribution, the belief can also be represented with a Gaussian Mixture Model (GMM) as proposed by Jonschkowski et al. (2018). Every particle contributes a separate Gaussian $N_i(\boldsymbol{\chi}^i, \boldsymbol{\Sigma})$ in the GMM and the mixture weights are the particle weights. The drawback of this approach is that the fixed covariance $\boldsymbol{\Sigma}$ of the individual distributions is an additional tuning parameter for the filter. We call this version dPF-M and calculate the negative log likelihood with

$$L_{\text{NLL}} = \frac{1}{T}\sum_{t=0}^{T} \log \sum_{i=0}^{|\mathbf{X}|} \frac{\pi^i}{\sqrt{|\boldsymbol{\Sigma}|}} \exp(\mathbf{x}_t^l - \boldsymbol{\chi}_t^i)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_t^l - \boldsymbol{\chi}_t^i) \quad (3)$$

5 Experimental setup

In the following, we will evaluate the DFs on three different filtering problems. We start with a simple simulation setting that gives us full control over parameters of the system such as the true process noise (Sec. 6). In Sects. 7 and 8, we then study the performance of the DFs on two real-robot tasks: The first is the KITTI Visual Odometry problem, where the filters are used to track the position and heading of a moving car given only RGB images. The second is planar pushing, where the filters track the pose of an object while a robot performs a series of pushes.

Unless stated otherwise, we will train the DFs end-to-end for 15 epochs using the Adam optimizer (Kingma and Ba 2015) and select the model state at the training step with the best validation loss for evaluation. We also evaluate different learning rates for all DFs. During training, the initial state is perturbed with noise sampled from a Normal distribution $N_{\text{init}}(0, \boldsymbol{\Sigma}_{\text{init}})$. For testing, we evaluate all DFs with the true initial state as well as with few fixed perturbations (sampled from N_{init}) and average the results.

More detailed information about the experimental conditions as well as extended results can be found in Online Material 1, Sec. B-D.

6 Simulated disc tracking

We first evaluate the DFs in a simulated environment similar to the one in Haarnoja et al. (2016): the task is to track a red disc moving among varying numbers of distractor discs, as shown in Fig. 1. The state consists of the position $\mathbf{p}$ and linear velocity $\mathbf{v}$ of the red disc.

The dynamics model that we use for generating the training data is

$$\mathbf{p}_{t+1} = \mathbf{p}_t + \mathbf{v}_t + \mathbf{q}_{p,t}$$
$$\mathbf{v}_{t+1} = \mathbf{v}_t - f_p \mathbf{p}_t - f_d \mathbf{v}_t^2 \text{sgn}(\mathbf{v}_t) + \mathbf{q}_{v,t}$$

The velocity update contains a force that pulls the discs towards the origin ($f_p = 0.05$) and a drag force that prevents too high velocities ($f_d = 0.0075$). $\mathbf{q}$ represents the Gaussian process noise and $\text{sgn}(x)$ returns the sign of x or 0 if $x = 0$.

The sensor network receives the current image at each step, from which it can estimate the position but not the velocity of the target. As we do not model collisions, the red disc can be occluded by the distractors or leave the image temporarily.

6.1 Data

We create multiple datasets with varying numbers of distractors, different levels of constant process noise for the disc

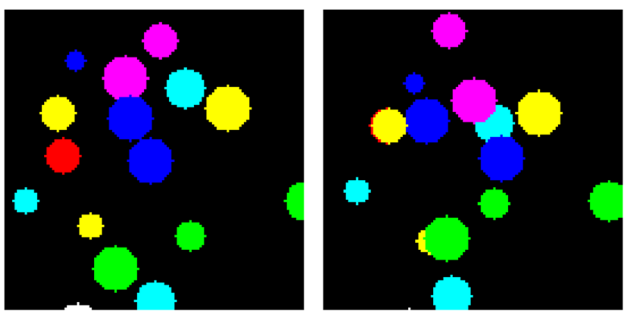

Fig. 1 Two sequential observations from our simulated tracking task. The filters need to track the red disc, which can be occluded by the other discs or leave the image temporarily (Color figure online)

position and constant or heteroscedastic process noise for the disc velocity. All datasets contain 2400 sequences for training, 300 validation sequences and 303 sequences for testing. The sequences have 50 steps and the colors and sizes of the distractors are drawn randomly for each sequence.

6.2 Filter implementation and parameters

We first evaluated different design choices and filter-specific parameters for the DFs to find settings that perform well and increase the stability of the filters during training. For detailed information about the experiments and results, please refer to Online Material 1, Sec. B.2.

6.2.1 dUKF

The dUKF has three filter-specific scaling parameters, α, κ and β. α and κ determine how far from the mean of the belief the sigma points are placed and how the mean is weighted in comparison to the other sigma points. β only affects the weight of the central sigma point when computing the covariance of the transformed distribution.

We evaluated different parameter settings but found no significant differences between them. In all following experiments, we use $\alpha = 1$, $\kappa = 0.5$ and $\beta = 0$. In general, we recommend values for which $\lambda = \alpha^2(\kappa + n) - n$ is a small positive number, so that the sigma points are not spread out too far and the central sigma point is not weighted negatively (which happens for negative λ). See Online Material 1, Sec. A.3 for a more detailed explanation.

6.2.2 dMCUKF

In contrast to the dUKF, the dMCUKF simply samples pseudo sigma points from the current belief. Its only parameter thus is the number N of sampled points during training and testing.

We trained the dMCUKF with $N \in \{5, 10, 50, 100, 500\}$ and evaluated with 500 pseudo sigma points. The results show that as few as ten sigma points are enough for training the dMCUKF relatively successfully. The best results are obtained with 100 sigma points and using more does not reliably increase the performance.

In the following, we use 100 points for training and 500 for testing. More complex problems with higher-dimensional states could, however, require more sigma points.

6.2.3 dPF: belief representation

When training the dPF on L_{NLL}, we have to choose how to represent the belief of the filter for computing the likelihood (see Sec. 4.5). We investigate using a single Gaussian (dPF-G) or a Gaussian Mixture Model (dPF-M). For the dPF-M, the covariance $\mathbf{\Sigma}$ of the single Gaussians in the Mixture Model is an additional parameter that has to be tuned.

As our test scenario does not require tracking multiple hypotheses, the representation by a single Gaussian in dPF-G should be accurate for this task. Nonetheless, we find that the dPF-G performs much worse than the dPF-M. This could either mean that Eq. 3 facilitates training or that approximating the belief with a single Gaussian removes useful information even when the task does not obviously require tracking multiple hypotheses. Interestingly, when using a learned observation update, this effect is not noticeable, which suggests that the first hypothesis is correct. In the following, we only report results for the dPF-M. Results for dPF-G can be found in Online Material 1.

For the dPF-M, $\mathbf{\Sigma} = 0.25\mathbf{I}_4$ ($\mathbf{I}_4$ denotes an identity matrix with 4 rows and columns) resulted in the best tracking errors, but the best NLL was achieved with $\mathbf{\Sigma} = \mathbf{I}_4$. We thus use $\mathbf{\Sigma} = \mathbf{I}_4$ for the dPF-M in all following experiments. It is, however, possible that different tasks could require different settings.

6.2.4 dPF: observation update

As mentioned before, the likelihood for the observation update step of the dPF can be implemented with an analytical Gaussian likelihood function (dPF-(G/M)) or with a neural network (dPF-(G/M)-lrn).

Our experiments showed that using a learned likelihood function for updating the particle weights can improve both tracking error and NLL of the dPF significantly. We attribute this mainly to the fact that the learned update relaxes some of the assumptions encoded in the particle filter: With the analytical version, we restrict the filter to use additive Gaussian noise that is either constant or depends only on the raw sensory observations. The learned update, in contrast, enforces no functional form of the noise model. In addition, the noise can depend not only on the raw sensory data, but also on the observable components of the particle states. This means that the learned observation update is potentially much more expressive than the analytical one, which pays off when the Gaussian assumption made by the other filtering algorithms does not hold.

While learning the observation update improves the performance of the dPF, we will still use the analytical variant in most of the following evaluations. The main reason for this is that the analytical observation update has explicit models for the sensor network and observation noise. This facilitates comparing between the dPF and the other DF variants and gives us control over the form of the learned observation noise.

6.2.5 dPF: resampling

The resampling step of the particle filter discards particles with low weights and prevents particle depletion. It may, however, be disadvantageous during training since it is not fully differentiable. Karkus et al. (2018) proposed soft resampling, where the resampling distribution is traded off with a uniform distribution to enable gradient flow between the weights of the old and new particles. This trade-off is controlled by a parameter $\alpha_{\text{re}} \in [0, 1]$. The higher α_{re}, the more weight is put on the uniform distribution. An alternative to soft resampling is to not resample at every timestep.

We tested the dPF-M with different values of α_{re} and when resampling every 1, 2, 5 or 10 steps and found that resampling frequently generally improves the filter performance. Soft resampling also did not have much of a positive effect in our experiments, presumably because higher values of α_{re} decrease the effectiveness of the resampling step. In the following, we use $\alpha_{\text{re}} = 0.05$ and resample at every timestep.

6.2.6 dPF: number of particles

Finally, the user also has to decide how many particles to use during training and testing. As for the dMCUKF, we trained the dPF-M with $N \in \{5, 10, 50, 100, 500\}$. The results were very similar to dMCUKF and we also use 100 particles during training and 500 particles for testing.

6.3 Loss function

In this experiment we compare the different loss functions introduced in Sec. 4.5, as well as a combination of the two $L_{\text{mix}} = 0.5(L_{\text{MSE}} + L_{\text{NLL}})$. Our hypothesis is that L_{NLL} is better suited for learning noise models, since it requires predicting the uncertainty about the state, while L_{MSE} only optimizes the tracking performance.

Experiment

We use a dataset with 15 distractors and constant process noise ($\sigma_{q_p} = 0.1$, $\sigma_{q_v} = 2$). The filters learn the sensor and process model as well as heteroscedastic observation noise and constant process noise models.

Results As expected, training on L_{NLL} leads to much better likelihoods scores than training on L_{MSE} for all DFs, see Fig. 2. The best tracking errors on the other hand are reached with L_{MSE}, as well as more precise sensor models.

For judging the quality of a DF, both NLL and tracking error should be taken into account: While a low RMSE is important for all tasks that use the state estimate, a good likelihood means that the uncertainty about the state is communicated correctly, which enables e.g. risk-aware planning and failure detection.

The combined loss L_{mix} trades off these two objectives during training. It does, however, not outperform the single losses in their respective objective. A possible explanation is that they can result in opposing gradients: All DFs tend to overestimate the process noise when trained only on L_{MSE}. This lowers the tracking error by giving more weight to the observations in dEKF, dUKF and dMCUKF and allowing more exploration in the dPF. But it also results in a higher uncertainty about the state, which is undesirable when optimizing the likelihood.

We generally recommend using L_{NLL} during training to ensure learning accurate noise models. If learning the process and sensor model does not work well, L_{NLL} can either be combined with L_{MSE} or the models can be pretrained.

6.4 Training sequence length

Karkus et al. (2018) evaluated training their dPF on sequences of length $k \in \{1, 2, 4\}$ and found that using more steps improved results. Here, we want to test if increasing the sequence length even further is beneficial. However, longer training sequences also mean longer training times (or more memory consumption). We thus aim to find a value for k with a good trade off between training speed and model performance.

Experiment

We evaluate the DFs on a dataset with 15 distractors and constant process noise ($\sigma_{q_p} = 0.1$, $\sigma_{q_v} = 2$). The filters learn the sensor and process model as well as heteroscedastic observation noise and constant process noise models. We train using L_{NLL} on sequence lengths $k \in \{1, 2, 5, 10, 25, 50\}$ while keeping the total number of examples per batch (steps $\times$ batch size) constant.

Results

Our results in Fig. 3 show that all filters benefit from longer training sequences much more than the results in Karkus et al. (2018) indicated. However, while only one time step is clearly too little, returns diminish after around ten steps.

Why are longer training sequences helpful? One issue with short sequences is that we use noisy initial states during training. This reflects real-world conditions, but the noisy inputs hinder learning the process model. On longer sequences, the observation updates can improve the state estimate and thus provide more accurate input values.

We repeated the experiment without perturbing the initial state, but the results with $k \in \{1, 2\}$ got even worse: Since the DFs could now learn accurate process models, they did not need the observations to achieve a low training loss and thus did not learn a proper sensor model. On the longer test sequences, however, even small errors from the noisy dynamics accumulate over time if they are not corrected by the observations.

To summarize, longer sequences are beneficial for training DFs, because they demonstrate error accumulation during filtering and allow for convergence of the state estimate

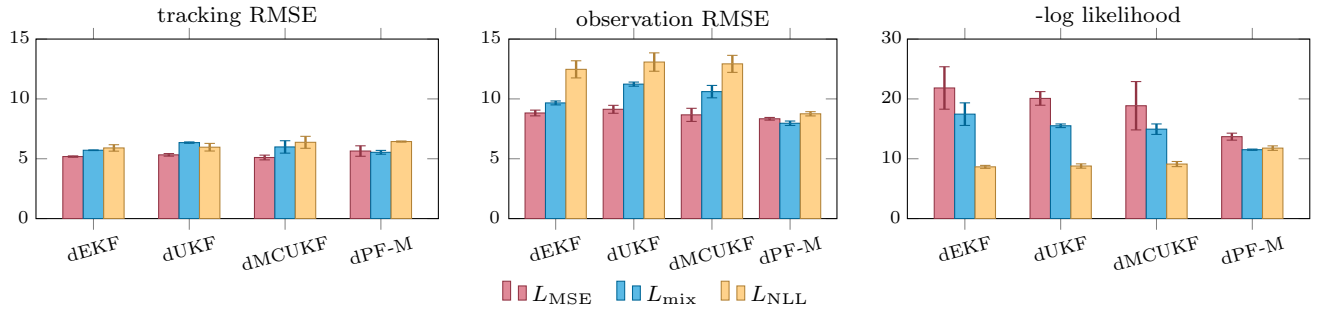

Fig. 2 Results on disc tracking: performance of the DFs when trained with different loss functions L_{MSE}, L_{NLL} or L_{mix} averaged over all steps in the trajectory. The first plot shows the RSME of the estimated state while the second shows the difference between the output of the sensor network and the corresponding ground truth state components. The negative log likelihood is computed as shown in Eqs. 2 or 3 for the dPF-M (Color figure online)

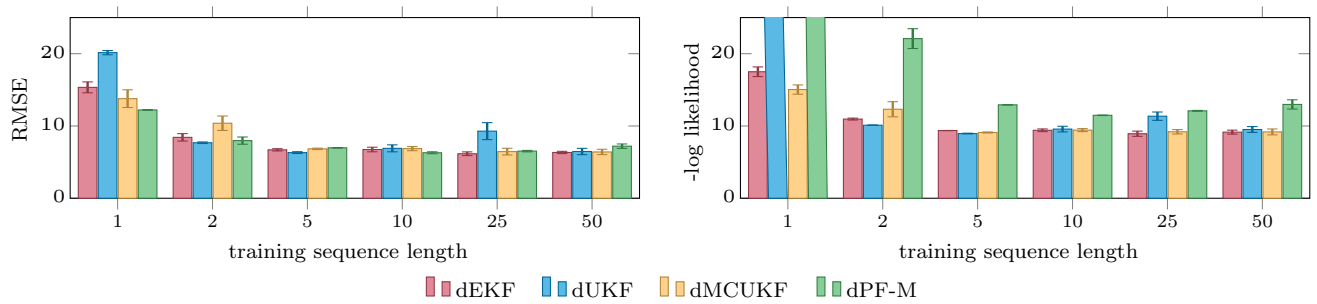

Fig. 3 Results on disc tracking: performance of the DFs trained with different sequence lengths. The cut-off NLL values for sequence length 1 are 47.4±3.9 for the dUKF and 79.0± 2.7 for the dPF-M (Color figure online)

when the initial state is noisy. However, performance eventually saturates and increasing k also increased our training times. We therefore chose $k = 10$ for all experiments, which provides a good trade-off between training speed and performance.

6.5 Learning noise models

The following experiments analyze how well complex models of the process and observation noise can be learned through the filters and how much this improves the filter performance. To isolate the effect of the noise models, we use a fixed, pretrained sensor model and the true analytical process model, such that only the noise models are trained. We initialize $\mathbf{Q}$ and $\mathbf{R}$ with $\mathbf{Q} = \mathbf{I}_4$ and $\mathbf{R} = 100\mathbf{I}_2$. All DFs are trained on L_{NLL}.

Online Material 1 contains extended experimental results on additional datasets as well as data for the dPF-G.

6.5.1 Heteroscedastic observation noise

We first test if learning more complex, heteroscedastic observation noise models improves the performance of the filters as compared to learning constant noise models. For this, we compare DFs that learn constant or heteroscedastic observation noise (the process noise is constant) on a dataset with constant process noise ($\sigma_{q_p} = 3, \sigma_{q_v} = 2$) and 30 distractors.

To measure how well the predicted observation noise reflects the visibility of the target disc, we compute the correlation coefficient between the predicted $\mathbf{R}$ and the number of visible target pixels. We also evaluate the similarity between the learned and the true process noise model using the Bhattacharyya distance.

Results

Results are shown in Table 1. When learning constant observation noise, all DFs perform relatively bad in terms of the tracking error. Upon inspection, we find that all filters learn a very high $\mathbf{R}$ and thus mostly rely on the process model for their prediction. For example, the dEKF predicts $\sigma_{r_p} = 25.4$. This is expected, since trusting the observations would result in wrong updates to the mean state estimate when the target disc is occluded.

Like Haarnoja et al. (2016), we find that heteroscedastic observation noise significantly improves the tracking performance of all DFs (except for the dPF-M). The strong negative correlation between $\mathbf{R}$ and the visible disc pixels shows that the DFs correctly predict higher uncertainty when the target is occluded. For example, the dEKF predicts values as low

Table 1 Results for disc tracking: end-to-end learning of the noise models through the DFs on datasets with 30 distractors and different levels of process noise

	R	RMSE	NLL	Corr.	$D_{\mathbf{Q}}$
dEKF	Const.	16.2	14.0	–	0.121
	Hetero.	**8.8**	**10.7**	−0.78	**0.002**
dUKF	Const.	16.8	14.1	–	0.161
	Hetero.	**8.8**	**10.7**	−0.78	**0.013**
dMCUKF	Const.	16.7	14.1	–	0.152
	Hetero.	**9.0**	**10.9**	−0.78	**0.006**
dPF-M	Const.	16.1	34.3	–	0.435
	Hetero.	**9.6**	**20.8**	−0.77	**0.280**

While $\mathbf{Q}$ is always constant, we evaluate learning constant (const.) or heteroscedastic (hetero) observation noise $\mathbf{R}$. We show the tracking error (RMSE), negative log likelihood (NLL), the correlation coefficient between predicted $\mathbf{R}$ and the number of visible pixels of the target disc (corr.) and the Bhattacharyya distance between true and learned process noise model ($D_{\mathbf{Q}}$). The best results per DF are highlighted in bold

as $\sigma_{r_p} = 0.9$ when the disc is perfectly visible and as high as $\sigma_{r_p} = 29.3$ when it is fully occluded.

Finally, all DFs learn values of $\mathbf{Q}$ that are close to the ground truth. For dEKF, dUKF and dMCUKF, the results improve significantly when heteroscedastic observation noise is learned. This could be because the worse tracking performance with constant observation noise impedes learning an accurate process model and thus requires higher process noise.

6.5.2 Heteroscedastic process noise

The effect of learning heteroscedastic process noise has not yet been evaluated in related work. We create datasets with heteroscedastic ground truth process noise, where the magnitude of $\mathbf{q}_v$ increases in three steps the closer to the origin the disc is. The positional process noise $\mathbf{q}_p$ remains constant ($\sigma_{q_p} = 3.0$).

We compare the performance of DFs that learn constant and heteroscedastic process noise while the observation noise is heteroscedastic in all cases.

Results

As shown in Table 2, learning heteroscedastic models of the process noise is a bit more difficult than for the observation noise. This is not surprising, as the input values for predicting the process noise are the noisy state estimates.

Plotting the predicted values for $\mathbf{Q}$ (see Fig. 4 for an example from the dEKF) reveals that all DFs learn to follow the real values for the heteroscedastic velocity noise relatively well, but also predict state dependent values for $\mathbf{q}_p$, which is actually constant. This could mean that the models have difficulties distinguishing between $\mathbf{q}_p$ and $\mathbf{q}_v$ as sources of uncertainty about the disc position. However, we see the same behavior also on a dataset with constant ground truth process noise. We thus assume that the models rather pick up an unintentional pattern in our data: The probability of the disc being occluded turned out to be higher in the middle of the image. The filters react to this by overestimating $\mathbf{q}_p$ in the center, which results in an overall higher uncertainty about the state in regions where occlusions are more likely.

Despite not being completely accurate, learning heteroscedastic noise models still increases performance of all DFs by a small but reliable value. Even when the ground-truth process noise model is constant, most of the DFs were able to improve their RSME and likelihood scores slightly by learning "wrong" heteroscedastic noise models.

6.5.3 Correlated noise

So far, we have only considered noise models with diagonal covariance matrices. In this experiment, we want to see if DFs can learn to identify correlations in the noise. We compare the performance of DFs that learn noise models with diagonal or full covariance matrix on datasets with and without correlated process noise. Both the learned process and the observation noise model are also heteroscedastic.

The results (see Online Material 1, Sec. B.3.3) show that learning correlated noise models leads to a further small improvement of the performance of all DFs when the true process noise is correlated. However, uncovering correlations in the noise seems to be even more difficult than learning accurate heteroscedastic noise models, as indicated by the still high Bhattacharyya distance between true and learned $\mathbf{Q}$.

6.6 Benchmarking

In the final experiment on this task, we compare the performance of the DFs among each other and to two LSTM models. We use an LSTM architecture similar to Jonschkowski et al. (2018), with one or two layers of LSTM cells (512 units each). The LSTM state is decoded into mean and covariance of a Gaussian state estimate.

Experiment All models are trained for 30 epochs. The DFs learn the sensor and process models with heteroscedastic, diagonal noise models. We compare their performance on datasets with 30 distractors and different levels of constant or heteroscedastic process noise. Each experiment is repeated two times to account for different initializations of the weights.

Results

The results in Table 3 show that all models (except for the dPF-G, see Online Material 1, Table S7) learn to track the target disc well and make reasonable uncertainty predictions.

Table 2 Results on disc tracking: end-to-end learning of constant or heteroscedastic process noise **Q** on datasets with 30 distractors and heteroscedastic or constant ($\sigma_{q_p} = 3.0$, $\sigma_{q_v} = 2.0$) process noise

		Heteroscedastic noise			Constant noise		
	Q	RMSE	NLL	$D_{\mathbf{Q}}$	RMSE	NLL	$D_{\mathbf{Q}}$
dEKF	Const.	8.09	11.620	0.879	8.80	10.687	**0.002**
	Hetero.	**7.36**	**11.289**	**0.402**	**8.77**	**10.684**	0.033
dUKF	Const.	7.85	11.318	0.874	8.80	10.743	**0.013**
	Hetero.	**7.60**	**11.167**	**0.391**	**8.68**	**10.727**	0.030
dMCUKF	const.	8.13	11.493	0.891	8.98	10.898	**0.006**
	Hetero.	**7.45**	**11.321**	**0.464**	**8.73**	**10.739**	0.044
dPF-M	Const.	8.48	15.232	1.072	**9.61**	20.789	**0.280**
	Hetero.	**8.23**	**14.725**	**0.787**	9.76	**19.833**	0.413

$D_{\mathbf{Q}}$ is the Bhattacharyya distance between true and learned process noise. The best results per DF are highlighted in bold

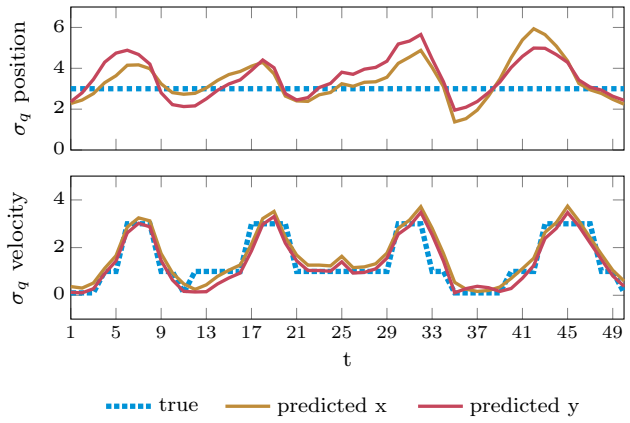

Fig. 4 Predicted and true process noise from the dEKF over one test sequence of the disc tracking task. Our model predicts separate values for the x and y-coordinates of position and velocity, but the ground truth process noise has the same σ for both coordinates (Color figure online)

In terms of tracking error, the dPF with learned observation update performs best on all evaluated datasets. This, however, often does not extend to the likelihood scores. For the NLL, the dMCUKF instead mostly achieves the best results, however, not with a significant advantage over the other DFs.

If we exclude the dPF variant with learned observation model (which is more expressive than the other DFs), we can see that the choice of the underlying filtering algorithm does not make a big difference for the performance on this task. The unstructured LSTM model, in contrast, requires two layers of LSTM cells (each with 512 units per layer) to reach the performance of the DFs. Unstructured models like LSTM can thus learn to perform similar to differentiable filters, but require a much higher number of trainable parameters than the DFs which increases computational demands and the risk of overfitting.

7 KITTI visual odometry

As a first real-world application we study the KITTI Visual Odometry problem (Geiger et al. 2012) that was also evaluated by Haarnoja et al. (2016) and Jonschkowski et al. (2018). The task is to estimate the position and orientation of a driv-

Table 3 Results on disc tracking: comparison between the DFs and LSTM models with one or two LSTM layers on two different datasets with 30 distractors and constant process noise with increasing magnitude

	$\sigma_{q_p} = 3.0$		$\sigma_{q_p} = 9.0$	
	RMSE	NLL	RMSE	NLL
dEKF	6.31±0.12	9.24±0.10	11.83±0.28	11.10±0.20
dUKF	6.46±0.20	9.26±0.26	11.49±0.18	10.75±0.16
dMCUKF	6.53±0.18	**9.23±0.17**	11.59±0.10	**10.81±0.11**
dPF-M	6.75±0.07	12.33±0.09	11.52±0.07	20.50±0.36
dPF-M-lrn	**5.89±0.15**	11.43±0.15	**9.98±0.13**	19.17±0.18
LSTM-1	9.44±0.77	10.64±0.25	14.62±0.70	11.83±0.22
LSTM-2	7.13±0.86	9.76±0.56	13.95±0.51	11.93±0.07

Each experiment is repeated two times and we report mean and standard errors

ing car given a sequence of RGB images from a front facing camera and the true initial state.

The state is 5-dimensional and includes the position $\mathbf{p}$ and orientation θ of the car as well as the current linear and angular velocity v and $\dot{\theta}$. The real control input $\mathbf{u} = \left(\dot{v}\ \ddot{\theta}\right)^T$ is unknown and we thus treat changes in v and $\dot{\theta}$ as results of the process noise. The position and heading estimate can be updated analytically by Euler integration.

While the dynamics model is simple, the challenge in this task comes from the unknown actions and the fact that the absolute position and orientation of the car cannot be observed from the RGB images. At each timestep, the filters receive the current images as well as a difference image between the current and previous timestep. From this, the filters can estimate the angular and linear velocity to update the state, but the uncertainty about the absolute position and heading will inevitably grow due to missing feedback. Please refer to Online Material 1, Sec. C.1 for details on the implementation of the sensor network, the learned process model and the learned noise models.

7.1 Data

The KITTI Visual Odometry dataset consists of eleven trajectories of varying length (from 270 to over 4500 steps) with ground truth annotations for position and heading and image sequences from two different cameras collected at 10 Hz.

Following Haarnoja et al. (2016) and Jonschkowski et al. (2018), we build eleven different datasets. Each of the original trajectories is used as the test split of one dataset, while the remaining 10 sequences are used to construct the training and validation split.

To augment the data, we use the images from both cameras for each trajectory and also mirror the sequences. For training and validation, we extract 200 sequences of length 50 with different random starting points from each augmented trajectory. This results in 1013 training and 287 validation sequences. For testing, we extract sequences of length 100 from the augmented test-trajectory. The number of test sequences depends on the overall length of the test-trajectory.

When looking at the statistics of the eleven trajectories in the original KITTI dataset, Trajectory 1 can be identified as an outlier: It shows driving on a highway, where the velocity of the car is much higher than in all the other trajectories. As a result, the sensor models trained on the other sequences will yield bad results when evaluated on Trajectory 1. We will therefore mostly report results for only a ten-fold cross-validation that excludes the dataset for testing on Trajectory 1. We will refer to this as *KITTI-10* while the full, eleven-fold cross validation will be denoted as *KITTI-11*. In Sec. 7.4, results for both settings are reported, such that the influence of Trajectory 1 becomes visible.

7.2 Learning noise models

In this experiment, we want to test how much the DFs profit from learning the process and observation noise models end-to-end through the filters, as compared to using hand-tuned or individually learned noise models.

We also again compare learning constant or heteroscedastic noise models. In contrast to the previous task, we do not expect as large a difference between constant or heteroscedastic observation noise for this task, as the visual input does not contain occlusions or other events that would drastically change the quality of the predicted observations $\mathbf{z}$.

Experiment

As in the experiments on simulated data (Sec. 6.5), we use a fixed, pretrained sensor model and the analytical process model, and only train the noise models. We initialize $\mathbf{Q}$ and $\mathbf{R}$ with $\mathbf{Q} = \mathbf{I}_5$ and $\mathbf{R} = \mathbf{I}_2$. All DFs are trained with L_{NLL} and a sequence length of 25, which we found to be beneficial for learning the noise models in a preliminary experiment.

We compare the DFs when learning different combinations of constant or heteroscedastic process and observation noise. As on baseline, we use DFs with fixed constant noise models that reflect the average validation error of the pretrained sensor model and the analytical process model. A second baseline fixes the noise models to those obtained by individual pretraining, where we evaluate both constant and heteroscedastic models. All DFs are evaluated on *KITTI-10*.

Results

The results in Table 4 show that learning the noise models end-to-end through the filters greatly improves the NLL but has no big effect on the tracking errors for this task. The DFs with the hand-tuned, constant noise model have the by far worst NLL because they greatly underestimate the uncertainty about the vehicle pose. The DFs that use individually trained noise models perform better, but are still overly confident.

For most of the DFs, we achieve the best results when learning constant observation and heteroscedastic process noise. The worst results are achieved when instead the observation noise is heteroscedastic and the process noise constant. This could indicate that the true process noise can be better modeled by a state-dependent noise model while learning heteroscedastic observation noise leads to overfitting to the training data. However, the differences are overall not very pronounced.

Finally, we also evaluated the DFs with full covariance matrices for the noise models. For the setting with constant observation and heteroscedastic process noise, using full instead of diagonal covariance matrices barely had any effect on the tracking error and only slightly improved the NLL (e.g. from 27.1±5.0 to 26.5±4.6 for the dEKF).

Table 4 Results on *KITTI-10*: performance of the DFs with different noise models (mean and standard error)

		Hand-tuned $\mathbf{R}_c\mathbf{Q}_c$	Pretrained $\mathbf{R}_c\mathbf{Q}_c$	Pretrained $\mathbf{R}_h\mathbf{Q}_h$	$\mathbf{R}_c\mathbf{Q}_c$	$\mathbf{R}_c\mathbf{Q}_h$	$\mathbf{R}_h\mathbf{Q}_c$	$\mathbf{R}_h\mathbf{Q}_h$
RMSE	dEKF	9.67±0.8	**9.65±0.8**	10.53±1.0	9.70±0.8	9.69±0.8	9.74±0.8	9.68±0.8
	dUKF	9.73±0.7	**9.71±0.8**	10.68±1.0	**9.71±0.8**	**9.71±0.8**	9.81±0.8	9.72±0.8
	dMCUKF	9.73±0.7	9.71±0.8	10.68±1.0	9.71±0.8	9.70±0.8	9.80±0.8	**9.68±0.8**
	dPF-M	11.79±0.5	10.18±0.7	10.66±0.9	**9.72±0.8**	9.74±0.8	9.74±0.8	9.77±0.8
NLL	dEKF	304.4±43.8	139.6±16.7	107.7±15.6	39.5±4.0	38.9±5.0	40.7±3.7	**38.0±4.6**
	dUKF	305.9±43.7	140.0±16.6	108.1±15.5	40.5±4.0	**39.2±5.1**	41.3±4.0	40.1±5.4
	dMCUKF	306.0±43.8	140.0±16.6	108.2±15.5	33.9±3.2	**29.8±3.5**	33.3±3.2	30.3±3.7
	dPF-M	103.2±6.4	75.8±8.5	**71.1±6.5**	74.7±9.9	71.4±10.1	74.2±10.1	72.4±9.7

Hand-tuned and Pretrained use fixed noise models whereas for the other variants, the noise models are trained end-to-end through the DFs. $\mathbf{R}_c$ indicates a constant observation noise model and $\mathbf{R}_h$ a heteroscedastic one (same for $\mathbf{Q}$). The best results per DF are highlighted in bold

7.3 End-to-end versus individual training

Previous work (Jonschkowski et al. 2018) has shown that end-to-end training through differentiable filters leads to better results than running the DFs with models that were trained individually. Specifically, pretraining the models individually and finetuning end-to-end resulted in the best tracking performance. As a possible explanation, the authors found that the individually trained process noise model predicted noise close to the ground truth whereas the end-to-end trained model overestimated to noise, which is believed to be beneficial for filter performance.

Does this mean that end-to-end training through DFs mostly affects the noise models? To test this, we pretrain all models individually and compare the performance of the DFs without finetuning, when finetuning only the noise models or only the sensor and process model and when finetuning everything. We also report results for training the DFs from scratch.

Experiment

We pretrain sensor and process model and their associated (constant) noise models individually for 30 epochs. For finetuning, we load the pretrained models and finetune the desired parts for 10 epochs, while the end-to-end trained versions are trained for 30 epochs. All variants are evaluated using *KITTI-10* and trained using L_{NLL}.

Results

The results shown in Table 5 support our hypothesis that end-to-end training through the DFs is most important for learning the noise models: Finetuning only the noise models improved both RMSE and NLL of all DFs in comparison to the variants without finetuning or with finetuning only the sensor and process model (except for the dMCUKF). For dEKF and dPF, finetuning the sensor and process model even decreased the performance on both measures.

In terms of tracking error, individual pretraining plus finetuning the noise models lead to the best results on dEKF and dPF, while dUKF and dMCUKF performed slightly better when finetuning both sensor and process model and their

Table 5 Results on *KITTI-10*: RMSE and negative log likelihood for the DFs with different training schemes (mean and standard error)

		Individual	Finetune models	Finetune noise	Finetune all	From scratch
RMSE	dEKF	9.58±0.7	10.38±1.0	**9.54±0.7**	9.83±0.8	10.05±0.8
	dUKF	9.64±0.7	9.663±0.8	9.57±0.7	9.33±0.8	**9.29±0.6**
	dMCUKF	9.64±0.7	9.53±0.8	9.58±0.7	**9.35±0.7**	9.72±0.6
	dPF-M	10.29±0.6	10.86±0.8	**9.59±0.6**	10.09±0.9	10.20±0.9
NLL	dEKF	130.0±16.3	160.0±28.8	**51.3±5.1**	57.7±5.2	61.8±7.7
	dUKF	126.7±15.6	118.4±14.5	**57.3±5.5**	87.1±9.9	59.3±7.2
	dMCUKF	127.9±15.9	117.8±14.8	**50.0±4.6**	74.4±8.1	50.3±8.1
	dPF-M	76.9±7.6	86.3±11.3	**72.6±9.4**	80.9±12.2	82.4±12.2

We compare individually trained process, sensor and noise models against finetuning only the sensor and process models, finetuning only the noise models and finteuning all models through the DFs. We also report results for DFs trained *from scratch* without individual pretraining. The best results per DF are marked in bold

noise models (dMCUKF) or even learning both from scratch (dUKF). For the NLL, finetuning only the noise models lead to the best results for all DFs, followed in most cases by training from scratch.

To summarize, the results indicate that individual pretraining is helpful for learning the sensor and process models, but not for the noise models. End-to-end training through the DFs, on the other hand, again proved to be important for optimizing the noise models for the respective filtering algorithm but did not offer advantages for learning the sensor and process model.

7.4 Benchmarking

In the final experiment on this task, we compare the performance of the DFs to an LSTM model. We again use an LSTM architecture similar to Jonschkowski et al. (2018), but with only one layer of LSTM cells with 256 units. The LSTM state is decoded into an update for the mean and the covariance of a Gaussian state estimate. Like the process model of the DFs, the LSTM does not get the full initial state as input, but only those components that are necessary for computing a state update (velocities and sine and cosine of the heading). We chose this architecture in an attempt to make the learning task easier for the LSTM.

Experiment

All models are trained for 30 epochs using L_{NLL}, except for the LSTM, for which L_{mix} lead to better results. The DFs learn the sensor and process models with constant noise models. We report their performance on *KITTI-10* and *KITTI-11*, for comparison with prior work.

Results

The results in Table 6 show that by training all the models in the DFs from scratch, we can reach a performance that is competitive with prior work by Haarnoja et al. (2016), despite not relying on an analytical process model. We were, however, not able to reach the very good performance of the dPF reported by Jonschkowski et al. (2018). A possible cause for this could be that the normalization of the particles in the learned observation update used by Jonschkowski et al. (2018) helps the method to better deal with the higher overall velocity in Trajectory 1 of the KITTI dataset.

In contrast to the DF, we were not able to train LSTM models that reached a good evaluation performance on this task, despite trying multiple different architectures and loss functions. Different from the experiments on the simulation task, increasing the number of units per LSTM-layer or using multiple LSTM layers even decreased the performance here. To complement our results, we also report an LSTM result from Haarnoja et al. (2016) that does better on the position error but worse on the orientation error. While these findings do not mean that a better performance could not be reached with unstructured models given different architectures or training routines, it still shows that the added structure of the filtering algorithms greatly facilitates learning in more complex problems.

For this task, the dPF-M-lrn again achieves the overall best tracking result, closely followed by the dUKF which

Table 6 Results on KITTI: comparison between the DFs and LSTM (mean and standard error)

		RMSE	NLL	$\frac{\text{m}}{\text{m}}$	$\frac{\text{deg}}{\text{m}}$
KITTI-11	dEKF	15.8±5.8	338.8±277.1	0.24±0.04	0.080±0.005
	dUKF	14.9±5.7	326.7±267.5	**0.21 ± 0.04**	0.079±0.008
	dMCUKF	15.2±5.5	266.3±216.1	0.23±0.04	0.083±0.012
	dPF-M	16.3±6.1	115.2±34.6	0.24±0.04	**0.078 ± 0.006**
	dPF-M-lrn	**14.3 ± 5.2**	**94.2 ± 33.3**	0.22±0.04	0.088±0.013
	LSTM	25.7±5.7	3970.6±2227.4	0.55±0.05	0.081±0.008
	LSTM*	–	–	0.26	0.29
	BKF*	–	–	0.21	0.08
	DPF*	–	–	0.15±0.015	0.06±0.009
KITTI-10	dEKF	10.1±0.8	61.8±7.7	0.21±0.03	0.079±0.006
	dUKF	9.3±0.6	59.3±7.2	**0.18 ± 0.02**	0.080±0.008
	dMCUKF	9.7±0.6	**50.3 ± 8.1**	0.2 ±0.03	0.082±0.013
	dPF-M	10.2±0.9	82.4±12.2	0.21±0.02	**0.077 ± 0.007**
	dPF-M-lrn	**9.2 ± 0.7**	61.3±6.1	0.19±0.03	0.090±0.014
	LSTM	20.2±2.0	1764.6±340.4	0.54±0.06	0.079±0.008

Numbers for prior work BKF*, LSTM* taken from Haarnoja et al. (2016) and DPF* taken from Jonschkowski et al. (2018). BKF* and DPF* use a fixed analytical process model while our DFs learn both, sensor and process model. $\frac{\text{m}}{\text{m}}$ and $\frac{\text{deg}}{\text{m}}$ denote the translation and rotation error at the final step of the sequence divided by the overall distance traveled

reaches the lowest normalized endpoint position error ($\frac{\text{m}}{\text{m}}$). One reason for the comparably bad performance of the dEKF could be that the dynamics of the Visual Odometry task are more strongly non-linear than in the previous experiments. Both UKF and PF can convey the uncertainty more faithfully in this case, which could lead to better overall results when training on L_{NLL}. Given the relatively large standard errors, the differences between the DFs are, however, not significant.

8 Planar pushing

In the KITTI Visual Odometry problem, the main challenges were the unknown actions and dealing with the inevitably increasing uncertainty about the vehicle pose. With planar pushing, our second real-robot experiment in contrast addresses a task with much more complex dynamics. Apart from having non-linear and discontinuous dynamics (when the pusher makes or breaks contact with the object), Bauza and Rodriguez (2017) also showed that the noise in the system can be best captured by a heteroscedastic noise model.

With 10 dimensions, the state representation we use is also much larger than in our previous experiments. $\mathbf{x}$ contains the 2D position $\mathbf{p}_o$ and orientation θ of the object, as well as the two friction-related parameters l and α_m. In addition, we include the 2D contact point between pusher and object $\mathbf{r}$, the normal to the object's surface at the contact point $\mathbf{n}$ and a contact indicator s. The control input $\mathbf{u}$ contains the start position $\mathbf{p}_u$ and movement $\mathbf{v}_u$ of the pusher.

An additional challenge of this task is that $\mathbf{r}$ and $\mathbf{n}$ are only properly defined and observable when the pusher is in contact with the object. We thus set the labels for $\mathbf{n}$ to zeros and $\mathbf{r} = \mathbf{p}_u$ for non-contact cases.

Dynamics

We use an analytical model by Lynch et al. (1992) to predict the linear and angular velocity of the object $(\mathbf{v}_o, \omega)$ given the previous state and the pusher motion $\mathbf{v}_u$. However, predicting the next $\mathbf{r}$, $\mathbf{n}$ and s is not possible with this model since this would require access to a representation of the object shape.

For $\mathbf{r}$, we thus use a simple heuristic that predicts the next contact point as $\mathbf{r}_{t+1} = \mathbf{r}_t + \mathbf{v}_{u,t}$. $\mathbf{n}$ and s are only updated when the angle between pusher movement and (inwards facing) normal is greater than 90°. In this case, we assume that the pusher moves away from the object and set s_{t+1} and $\mathbf{n}_{t+1}$ to zeros.

Observations

Our sensor network receives simulated RGBXYZ images as input and outputs the pose of the object, the contact point and normal as well as whether the push will be in contact with the object during the push or not.

Apart from from the latent parameters l and α_m, the orientation of the object, θ, is the only state component that cannot be observed directly. Estimating the orientation of an object from a single image would require a predefined "zero-orientation" for each object, which is impractical. Instead, we train the sensor network to predict the orientation relative to the object pose in the initial image of each pushing sequence.

8.1 Data

We use the data from the MIT Push dataset (Yu et al. 2016) as a basis for constructing our datasets. Further annotations for contact points and normals as well as rendered images are obtained using the tools described by Kloss et al. (2020). However, in contrast to Kloss et al. (2020), the images we use here also show the robot arm and are taken from a more realistic view-point. As a result, the robot frequently occludes parts of the object, but complete occlusions are rare. Figure 5 shows example views.

We use pushes with a velocity of 50 $\frac{\text{mm}}{\text{s}}$ and render images with a frequency of 5 Hz. This results in short sequences of about five images for each push in the original dataset. We extend them to 20 steps for training and validation and 50 steps for testing by chaining multiple pushes and adding in-between pusher movement when necessary. The resulting dataset contains 5515 sequences for training, 624 validation sequences and 751 sequences for testing.

8.2 Learning noise models

In this experiment, we again evaluate how much the DFs profit from learning the process and observation noise models end-to-end through the filters. In contrast to the KITTI task, for pushing, we expect both heteroscedastic observation and process noise to be advantageous, since the visual observations feature at least partial occlusions and the dynamics of pushing have been previously shown to exhibit heterostochasticity (Bauza and Rodriguez 2017).

To test this hypothesis, we compare DFs that learn constant or heteroscedastic noise models to DFs with hand-tuned, constant noise models that reflect the average test error of the pretrained sensor model and the analytical process model.

Experiment As in the corresponding experiments on the previous tasks (Sec. 6.5 and Sec. 7.2), we use a fixed, pretrained

Fig. 5 Examples of the rendered RGB images that we use as observations for the pushing task. The last example shows that the robot arm can partially occlude the object in some positions (Color figure online)

Table 7 Results for planar pushing: translation (tr) and rotation (rot) error and negative log likelihood for the DFs with different noise models

		Hand-tuned $\mathbf{R}_c\mathbf{Q}_c$	$\mathbf{R}_c\mathbf{Q}_c$	$\mathbf{R}_h\mathbf{Q}_c$	$\mathbf{R}_c\mathbf{Q}_h$	$\mathbf{R}_h\mathbf{Q}_h$
tr [mm]	dEKF	6.22	4.45	4.61	4.44	**4.38**
	dUKF	4.87	4.44	5.25	**4.43**	4.45
	dMCUKF	4.73	4.42	4.8	4.39	**4.35**
	dPF-M	18.13	5.07	4.92	5.32	**4.64**
rot [°]	dEKF	10.49	10.00	**9.71**	10.15	9.97
	dUKF	9.87	9.91	**9.73**	10.05	10.00
	dMCUKF	**9.78**	9.95	9.93	10.04	9.85
	dPF-M	16.18	10.18	**9.92**	10.39	10.06
NLL	dEKF	265.17	126.69	33.09	79.24	**26.48**
	dUKF	378.08	84.12	33.06	81.55	**27.61**
	dMCUKF	130.22	78.53	30.43	64.12	**30.1**
	dPF-M	353.25	128.15	104.40	103.21	**82.46**

The hand-tuned DFs use fixed noise models whereas for the other variants, the noise models are trained end-to-end through the DFs. $\mathbf{R}_c$ indicates a constant observation noise model and $\mathbf{R}_h$ a heteroscedastic one (same for $\mathbf{Q}$). The best result per DF are highlighted in bold

sensor model and the analytical process model, and only train the noise models. All DFs are trained for 15 epochs on L_{NLL}.

Results The results shown in Table 7 again demonstrate that learning the noise models end-to-end through the structure of the filtering algorithms is beneficial. With learned models, all DFs reach much better likelihood scores than with the hand-tuned variants. For the dEKF and especially the dPF, the tracking performance also improves significantly.

Comparing the results between constant and heteroscedastic noise models also confirms our hypothesis that for the pushing task, heteroscedastic noise models are beneficial for both observation and process noise. While all DFs reach the best NLL when both noise models are state-dependent, the effect on the tracking error is, however, less clear.

For dEKF, dUKF and dMCUKF, learning a heteroscedastic observation noise model leads to a much bigger improvement of the NLL than learning heteroscedastic process noise. Similar to the simulated disc tracking task, the input dependent noise model allows the DFs to better deal with occlusions in the observations, which again reflects in a negative correlation between the number of visible object pixels and the predicted positional observation noise.

8.3 Benchmarking

In the final experiment, we compare the performance of the DFs to an LSTM model on the pushing task. As before, we use a model with one LSTM layer with 256 units. The LSTM state is decoded into an update for the mean and the covariance of a Gaussian state estimate.

Experiment All models are trained for 30 epochs using L_{mix}. As initial experiments showed that learning sensor and process model jointly from scratch is very difficult for this task due to the more complex architectures, we pretrain both models. The sensor and process models are finetuned through the DFs and they learn heteroscedastic noise models. The LSTM, too, uses the pretrained sensor model, but not the process model.

Results As shown in Table 8, even with a learned process model, all DFs (except for the dPF-M-lrn) perform at least similar to their pendants in the previous experiment where we used the analytical process model. dEKF, dUKF and dMCUKF even reach a higher tracking performance than before. As noted by Kloss et al. (2020), this can be explained by the quasi-static assumption of the analytical model being violated for push velocities above 20 $\frac{\text{mm}}{\text{s}}$.

The LSTM model, again, does not reach the performance of the DFs. One disadvantage of the LSTM here is that in contrast to the DFs, we cannot isolate and pretrain the process model. In contrast to the previous tasks, the dPF variant with the learned likelihood function, however, performs even worse than the LSTM for planar pushing. This is likely due to the complex sensor model and the high-dimensional state that make learning the observation likelihood much more challenging.

9 Conclusions

Our experiments show that all evaluated DFs are well suited for learning both sensor and process model, and the associated noise models. For simpler tasks like the simulated tracking task and the KITTI Visual Odometry problem, all of these models can be learned end-to-end. Only the pushing problem with its large state and complex dynamics and sensor model requires pretraining to achieve good results.

Table 8 Results on pushing: comparison between the DFs and LSTM. Process and sensor model are pretrained and get finetuned end-to-end

	RMSE	NLL	tr [mm]	rot [°]
dEKF	14.9±0.46	33.9±3.86	**3.5 ± 0.02**	8.8±0.22
dUKF	**13.7 ± 0.15**	**31.1 ± 1.90**	3.7±0.06	8.8±0.14
dMCUKF	13.8±0.10	34.1±3.57	3.7±0.06	**8.8 ± 0.06**
dPF-M	18.3±0.38	120.4±5.70	5.7±0.16	10.5±0.36
dPF-M-lrn	29.0±0.73	486.0±3.27	12.0±0.78	18.9±0.04
LSTM	27.36±0.2	35.4±0.24	8.8±0.17	19.0±0.001

The DFs learn heteroscedastic noise models. Each experiment is repeated three times and we report mean and standard errors

In comparison to unstructured LSTM models, the DFs generally use fewer weights and achieve better results, especially on complex tasks. While training better LSTM models might be possible for more experienced LSTM users, using the algorithmic structure of the filtering algorithms definitely facilitated the learning problem and thus made it much easier to reach good performance with the DFs. In addition, the structure of DFs allows us to pretrain components such as the process model that are not explicitly accessible in LSTMs.

The direct comparison between DFs with different underlying filtering algorithms showed no clear winner. Only the dPF with learned observation update performed notably better than the other variants on the simulated example task and was least affected by the outlier-trajectory of the KITTI-task. This variant relaxes some of the assumptions that the filtering algorithms encode by not relying on an explicit sensor or observation noise model. Its good performance thus shows that the priors enforced by the algorithm choice can also be harmful if they do not hold in practice, such as the Gaussian noise assumption.

Our experiments suggest that for learning the sensor and process model, end-to-end training through the filters is convenient, but provides no advantages over training the models individually. End-to-end training, however, proved to be essential for optimizing the noise models for their respective filtering algorithm. In contrast to end-to-end trained models, both hand-tuned and individually trained noise models did not result in optimal performance of the DFs. Training noise models through DFs also enables learning more complex noise models than the ones used in learning-free, hand-tuned filters. We demonstrate that noise models with full (instead of diagonal) covariance matrices and especially heteroscedastic noise model, can significantly improve the tracking accuracy and uncertainty estimates of DFs.

The main challenge in working with differentiable filters is keeping the training stable and finding good choices for the numerous hyper-parameters and implementation options of the filters. While we hope that this work provides some orientation about which parameters matter and how to set them, we still recommend using the dEKF for getting started with differentiable filters. It is not only the most simple of the DFs we evaluated, but it also proved to be relatively insensitive to sub-optimal initialization of the noise models and was the most numerically stable during training. On the other hand, for tasks with strongly non-linear dynamics, the dUKF, dMCUKF or dPF can, however, ultimately achieve a better tracking performance.

One interesting direction for future research that we have not attempted here is to optimize parameters of the filtering algorithms, such as the scaling parameters of the dUKF or the fixed covariance of the mixture model components in the dPF-M, by end-to-end training. It could also be interesting to implement DFs with other underlying filtering algorithms. For example, the pushing task could potentially be better handled by a Switching Kalman filter (Murphy 1998) that explicitly treats the contact state as a binary decision variable. In addition, all of our DFs perform badly on the outlier trajectory of the KITTI dataset which features a much higher driving velocity than the other trajectories we used for training the model. This shows that the ability to detect input values outside of the training distribution would be a valuable addition to current DFs. Finally, it would be interesting to compare learning in DFs to similar variational methods such as the ones introduced by Karl et al. (2017); Fraccaro et al. (2017); Le et al. (2018) or the model-free PF-RNNs introduced by Ma et al. (2020).

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10514-021-09990-9.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org

Amos, B., Jimenez, I., Sacks, J., Boots, B., & Kolter, J.Z. (2018) Differentiable mpc for end-to-end planning and control. In *Advances in neural information processing systems*, Curran Associates, Inc., pp. 8289–8300

Archer, E., Park, I.M., Buesing, L., Cunningham, J., & Paninski, L. (2015) Black box variational inference for state space models. arXiv preprint arXiv:1511.07367.

Bauza, M., & Rodriguez, A. (2017) A probabilistic data-driven model for planar pushing. In *IEEE international conference on robotics and automation* (pp. 3008–3015). https://doi.org/10.1109/ICRA.2017.7989345.

Bavdekar, V. A., Deshpande, A. P., & Patwardhan, S. C. (2011). Identification of process and measurement noise covariance for state and parameter estimation using extended kalman filter. *Journal of Process Control*, *21*(4), 585–601. https://doi.org/10.1016/j.jprocont.2011.01.001.

Donti, P., Amos, B., & Kolter, J.Z. (2017). Task-based end-to-end model learning in stochastic optimization. In *Advances in neural information processing systems*, Curran Associates, Inc., pp. 5484–5494.

Farquhar, G., Rocktaeschel, T., Igl, M., & Whiteson, S. (2018). TreeQN and ATreec: Differentiable tree planning for deep reinforcement learning. In *International conference on learning representations*.

Fraccaro, M., Kamronn, S., Paquet, U., & Winther, O. (2017) A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in neural information processing systems* (pp. 3601–3610).

Geiger, A., Lenz, P., & Urtasun, R. (2012) Are we ready for autonomous driving? The kitti vision benchmark suite. In *Conference on computer vision and pattern recognition*.

Girin, L., Leglaive, S., Bie, X., Diard, J., Hueber, T., & Alameda-Pineda, X. (2020) Dynamical variational autoencoders: A comprehensive review. arXiv preprint arXiv:2008.12595.

Guez, A., Weber, T., Antonoglou, I., Simonyan, K., Vinyals, O., Wierstra, D., Munos, R., & Silver, D. (2018) Learning to search with mctsnets. In *International conference on machine learning*, PMLR (Vol. 80, pp. 1817–1826).

Haarnoja, T., Ajay, A., Levine, S., Abbeel, P. (2016) Backprop KF: Learning discriminative deterministic state estimators. In *Advances in neural information processing systems* (pp. 4376–4384).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Holl, P., Thuerey, N., & Koltun, V. (2020). Learning to control pdes with differentiable physics. In *International conference on learning representations*.

Jonschkowski, R., & Brock, O. (2016). End-to-end learnable histogram filters. In *Workshop on deep learning for action and interaction at NIPS*.

Jonschkowski, R., Rastogi, D., & Brock, O. (2018). Differentiable particle filters: End-to-end learning with algorithmic priors. In *Robotics: science and systems*, Pittsburgh, USA.

Karkus, P., Hsu, D., & Lee, W. S. (2017). QMDP-Net: Deep learning for planning under partial observability. In *Advances in neural information processing systems* (pp. 4694–4704).

Karkus, P., Hsu, D., & Lee, W.S. (2018) Particle filter networks with application to visual localization. In *Conference on robot learning* (pp 169–178).

Karkus, P., Ma, X., Hsu, D., Kaelbling, L.P., Lee, W.S., & Lozano-Pérez, T. (2019). Differentiable algorithm networks for composable robot learning. In *Robotics: Science and systems*.

Karl, M., Soelch, M., Bayer, J., & van der Smagt, P. (2017) Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *International conference on learning representations*.

Kersting, K., Plagemann, C., Pfaff, P., & Burgard, W. (2007) Most likely heteroscedastic gaussian process regression. In *International conference on machine learning*, ACM (pp. 393–400).

Kingma, D.P., & Ba, J. (2015) Adam: A method for stochastic optimization. In Bengio, Y., & LeCun, Y. (eds.) *International conference on learning representations*.

Kloss, A., Schaal, S., & Bohg, J. (2020). Combining learned and analytical models for predicting action effects from sensory data. *The International Journal of Robotics Research*. https://doi.org/10.1177/0278364920954896.

Krishnan, R.G., Shalit, U., & Sontag, D. (2016) Structured inference networks for nonlinear state space models. arXiv preprint arXiv:1609.09869.

Le, T.A., Igl, M., Rainforth, T., Jin, T., & Wood, F. (2018) Auto-encoding sequential monte carlo. In *International conference on learning representations*, https://openreview.net/forum?id=BJ8c3f-0b.

Lynch, K.M., Maekawa, H., & Tanie, K. (1992). Manipulation and active sensing by pushing using tactile feedback. In *IEEE international conference intelligent robots and systems* (Vol. 1, pp. 416–421). https://doi.org/10.1109/IROS.1992.587370.

Ma, X., Karkus, P., Hsu, D., & Lee, W. S. (2020). Particle filter recurrent neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*, 5101–5108.

Maddison, C.J., Lawson, D., Tucker, G., Heess, N., Norouzi, M., Mnih, A., Doucet, A., & Teh, Y.W. (2017) Filtering variational objectives. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 6576–6586).

Murphy, K.P. (1998) Switching kalman filters.

Naesseth, C., Linderman, S., Ranganath, R., & Blei, D. (2018) Variational sequential monte carlo. In *International conference on artificial intelligence and statistics*, PMLR (pp. 968–977).

Oh, J., Singh, S., & Lee, H. (2017) Value prediction network. In *Advances in neural information processing systems*, Curran Associates, Inc. (pp. 6118–6128).

Okada, M., Rigazio, L., & Aoshima, T. (2017) Path integral networks: End-to-end differentiable optimal control. arXiv preprint arXiv:1706.09597.

Pereira, M., Fan, D. D., An, G. N.,& Theodorou, E. (2018) Mpc-inspired neural network policies for sequential decision making. arXiv preprint arXiv:1802.05803.

Pontón, B., Schaal, S., & Righetti, L. (2020) On the effects of measurement uncertainty in optimal control of contact interactions. In *Algorithmic foundations of robotics XII*, Springer (pp. 784–799).

Tamar, A., Wu, Y., Thomas, G., Levine, S., & Abbeel, P. (2016) Value iteration networks. In *Advances in neural information processing systems* (pp. 2154–2162).

Todorov, E. (2005). Stochastic optimal control and estimation methods adapted to the noise characteristics of the sensorimotor system. *Neural Computation*, *17*(5), 1084–1108.

Valappil, J., & Georgakis, C. (2000). Systematic estimation of state noise statistics for extended kalman filters. *AIChE Journal*, *46*(2), 292–308.

Watter, M., Springenberg, J., Boedecker, J., & Riedmiller, M. (2015) Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems* (pp. 2746–2754).

Yu, K.T., Bauza, M., Fazeli, N., & Rodriguez, A. (2016) More than a million ways to be pushed. A high-fidelity experimental dataset of planar pushing. In *IEEE international conference on intelligent robots and systems* (pp. 30–37). https://doi.org/10.1109/IROS.2016.7758091, data available from http://web.mit.edu/mcube//push-dataset.

Zhu, M., Murphy, K., & Jonschkowski, R. (2020) Towards differentiable resampling. arXiv preprint arXiv:2004.11938.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Alina Kloss did her Ph.D. at the International Max Planck Research School for Intelligent Systems (IMPRS-IS) in Tübingen. She received a B.Sc. degree in bioinformatics and an M.Sc. degree in computer science from the University of Tübingen. Her research focuses on combing traditional, model-based approaches with deep learning methods for vision-based robotic manipulation.

Georg Martius is leading a research group on Autonomous Learning at the Max Planck Institute for Intelligent Systems in T übingen, Germany. Before joining the MPI in T übingen, he was a postdoc fellow at the IST Austria in the groups of Christoph Lampert and Gašper Tkačik after being a postdoc at the Max Planck Institute for Mathematics in the Sciences in Leipzig. He pursues research in autonomous learning, that is how an embodied agent can determine what to learn, how to learn, and how to judge the learning success. His research focus is on machine learning for robotics, including internal model learning, reinforcement learning, representation learning, differentiable reasoning and haptics.

Jeannette Bohg is an Assistant Professor of Computer Science at Stanford University. She was a group leader at the Autonomous Motion Department (AMD) of the MPI for Intelligent Systems until September 2017. Before joining AMD in January 2012, Jeannette Bohg was a PhD student at the Division of Robotics, Perception and Learning (RPL) at KTH in Stockholm. In her thesis, she proposed novel methods towards multi-modal scene understanding for robotic grasping. She also studied at Chalmers in Gothenburg and at the Technical University in Dresden where she received her Master in Art and Technology and her Diploma in Computer Science, respectively. Her research focuses on perception and learning for autonomous robotic manipulation and grasping. She is specifically interested in developing methods that are goal-directed, real-time and multi-modal such that they can provide meaningful feedback for execution and learning.