How to Train Your Differentiable Filter

Alina Kloss[1], Georg Martius[1] and Jeannette Bohg[1,2]

***Abstract*—In many robotic applications, it is crucial to maintain a belief about the state of a system. These state estimates serve as input for planning and decision making and provide feedback during task execution. Recursive Bayesian Filtering algorithms address the state estimation problem, but they require models of process dynamics and sensory observations as well as 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. The aim of this work is to improve understanding and applicability of such *differentiable filters* (DF). We implement DFs with four different underlying filtering algorithms and compare them in extensive experiments. We find that long enough training sequences are crucial for DF performance and that modelling heteroscedastic observation noise significantly improves results. And while the different DFs perform similarly on our example task, we recommend the differentiable Extended Kalman Filter for getting started due to its simplicity.**

I. Introduction

In many robotic applications, it is crucial to maintain a belief about the state of the system, 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.

Recursive Bayesian filters are a class of algorithms that combine perception and prediction for 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 for many problems, 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 state estimation problem directly, recent work [9, 7, 10, 13] 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 [9], Kalman Filters [7] and Particle Filters [10, 13], the respective authors showed that such *differentiable filters* (DF) systematically outperform unstructured neural networks like LSTMs. 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.

[1] Max Planck Institute for Intelligent Systems, <akloss, gmartius>@tue.mpg.de

[2] Stanford University, bohg@stanford.edu

A further interesting aspect of differentiable filters is that they allow for learning sophisticated models of the observation and process noise. This is useful as finding appropriate values for the process and observation noise is often difficult. Despite much research on identification methods (e.g. [4, 19]) the noise models 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 [3, 15].

The main goal of this paper is to provide practical guidance to researchers interested in applying differentiable filtering to their problem. To this end, we review and implement existing work on differentiable Kalman and Particle Filters and introduce two variants of differentiable Unscented Kalman Filters. We will also release our tensorflow [1] implementation.

In extensive experiments, 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 noise models and compare the DFs to unstructured LSTM [8] models.

We find that long enough training sequences are crucial for DF performance and that modelling heteroscedastic noise, especially on the observations, improves results significantly. While all DFs perform similarly on our example task, some require more implementation choices and parameter tuning. For getting started, we thus recommend the differentiable Extended Kalman Filter as the most simple of our DFs.

II. Background: Bayesian Filtering

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 $\mathbf{x}_0$, a sequence of observations $\mathbf{z}_{0\ldots t}$ and control inputs $\mathbf{u}_{0\ldots t-1}$. Formally, we seek the posterior distribution $p(\mathbf{x}_t|\mathbf{x}_{0\ldots t-1}, \mathbf{u}_{0\ldots t-1}, \mathbf{z}_{0\ldots t})$. The dynamics of the system is modelled by a *process model* f that describes how the state changes over time and 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 $\mathbf{q} \sim N(0, \mathbf{Q})$ and $\mathbf{r} \sim N(0, \mathbf{R})$ are the process and observation noise and represent the stochasticity of the system.

Bayesian Filters make the Markov assumption, i.e. that the distribution of the future states and observations is conditionally independent from the history of past states and observations given the current state. This assumption makes it possible to compute $p(\mathbf{x}_t | \mathbf{x}_{0 \ldots t-1}, \mathbf{u}_{0 \ldots t-1}, \mathbf{z}_{0 \ldots t})$ recursively from $p(\mathbf{x}_{t-1} | \mathbf{x}_{0 \ldots t-2}, \mathbf{u}_{0 \ldots t-2}, \mathbf{z}_{0 \ldots t-1})$ and so forth.

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 Appendix VII-A. For more details on EKF, UKF and PF, we refer to Thrun et al. [18]. We denote the differentiable filters as dEKF, dUKF, dMCUKF and dPF.

III. Related Work

Haarnoja et al. [7] proposed the BackpropKF, a differentiable implementation of the Kalman Filter. Jonschkowski and Brock [9] presented a differentiable Histogram Filter for discrete localization tasks in one or two dimensions and Jonschkowski et al. [10], Karkus et al. [13] both implemented differentiable Particle Filters for localization and tracking of a mobile robot. In the following, we focus our discussion on [7, 10, 13], since Histogram Filters are rarely applied in practice, due to the need for discretizing the state space.

All three works have in common that the raw observations are processed by a neural network that can be trained end-to-end through the filter. Haarnoja et al. [7] showed that this network can also learn to predict the observation noise $\mathbf{R}$ conditioned on the raw images. This drastically improves filter performance when facing heavy occlusions. For predicting the next state, all three works use an analytical instead of a learned process model. While [7] and [13] also assume known process noise, [10] predicts $\mathbf{Q}$.

Jonschkowski et al. [10] compared 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 performed better, presumably because it learned to overestimate the process noise. All works also compared their differentiable filters to unstructured LSTMs and found that including the structural priors of the filter algorithm and the known process models improved performance.

A second line of research closely related to differentiable filters is variational inference in temporal state space models [14, 21, 5, 2]. Recent results in this field showed that structuring the variational models similarly to Bayesian filters improves their performance [14, 5].

IV. Differentiable Bayesian Filters

We now describe how we implement the Bayesian filters presented in Sec. II as differentiable neural networks.

A. Observation Model

In Bayesian filtering, the observation model h is a *generative* model that predicts observations from the state $\mathbf{z}_t = h(\mathbf{x}_t)$. In practice, it is, however, 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. [7] 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. IV-C) for the current input $\mathbf{D}_t$.

In our experiment, $\mathbf{z}$ contains a subset of the state vector $\mathbf{x}$. The observation model $h(\mathbf{x})$ is thus a simple linear selection matrix of the observable components.

B. Process Model

Depending on the user's knowledge about the system, the process model f can be implemented using a known analytical model or a neural network n_p with weights $\mathbf{w}_p$. When using neural networks, n_p outputs 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 time steps and provides a reasonable initialization of the process model close to identity.

C. 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 diagonal covariance matrices $\mathbf{Q}_t$ and $\mathbf{R}_t$.

For constant noise, the filters directly learn the diagonal 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 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 $n_s(\mathbf{D}_t, \mathbf{w}_s)$.

D. Training

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 [13] is simply the mean squared error (MSE) between the mean of the belief and true state at each timestep:

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

The second loss function [7, 10] is the negative log likelihood 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

$$L_{nll} = \frac{1}{2T} \sum_{t=0}^{T} \log(|\boldsymbol{\Sigma}_t|) + (\mathbf{x}^l_t - \boldsymbol{\mu}_t)^T \boldsymbol{\Sigma}_t^{-1} (\mathbf{x}^l_t - \boldsymbol{\mu}_t) \tag{2}$$

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 using L_{nll} for training dPF: The first is to fit 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 as dPF-G.

To better reflect the multimodality of the particle distribution, the belief can also be represented with a Gaussian Mixture Model (GMM) as in [10]: Every particle gets 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. We call this version dPF-M and calculate the negative log likelihood with

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

V. Experiments

A. Simulated Task

We evaluate the DFs in a simulated environment similar to the one in [7]: The task is to track a red disc moving amongst varying numbers of distractor discs, as shown in Fig. 3. The state consists of the position $\mathbf{p}$ and velocity $\mathbf{v}$ of the red disc.

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.

For details on the system dynamics and training data, please refer to the Appendix VII-B. General information on the training process can be found in Appendix VII-C.

B. Implementation and Hyperparameters

We evaluate different design choices and hyperparameters for the DFs to find settings that perform well and increase the stability of the filters during training. We only summarize the results here, more details can be found in Appendix VII-D.

Summary of Results: The dEKF is the only DF without any parameters or design choices. For the dUKF, we use $\lambda = 0.5$. In contrast to the dUKF, the dMCUKF samples pseudo sigma points around the belief. We use 100 samples for training and 1000 for testing.

This also applies to the number of particles for the dPF. The dPF has the highest number of design choices: In contrast to [13, 10], we use an analytical Gaussian likelihood function for the observation update. We resample at every step and use soft resampling with a very small coefficient $\alpha = 0.05$. For the dPF-M, we use $\boldsymbol{\Sigma} = \mathbf{I}$ to compute L_{nll} (Eq. 3).

C. Loss Function

In this experiment we compare the two loss functions introduced in Sec. IV-D, and a combination of the two $L_{mix} = 0.5(L_{mse} + L_{nll})$. Our hypothesis is that L_{nll} is better suited for learning noise models, while L_{mse} only optimizes the tracking performance.

We also compare the two alternative belief representations for the dPF (dPF-G and dPF-M). As our test scenario does not

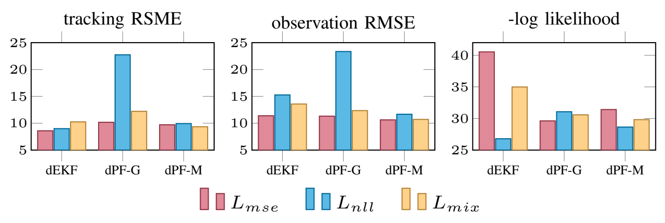

Fig. 1: Tracking error, observation error and negative log likelihood of dEKF dPF-G and dPF-M trained with loss functions L_{mse}, L_{nll} or L_{mix}.

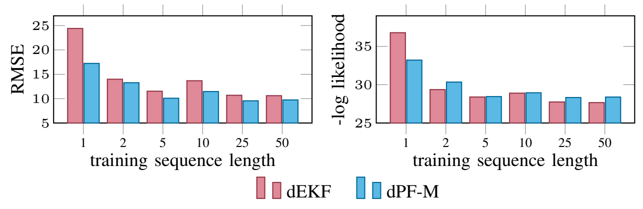

Fig. 2: Tracking error and negative log likelihood of dEKF and dPF-M trained with different sequence lengths.

require tracking multiple hypotheses, the representation by a single Gaussian in dPF-G should be accurate. More details on the experiment and results can be found in Appendix VII-E.

Summary of Results: For judging the quality of a DF, both likelihood 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 score means that uncertainty about the state is communicated correctly, which enables e.g. risk-aware planning.

As expected, training on L_{nll} leads to better likelihood scores than training on L_{mse}, see Fig. 1. The best tracking errors on the other hand are reached with L_{mse}, as well as better sensor models. The combined loss L_{mix} trades off the two objectives, but does not outperform the single losses in their respective scores. In contrast to dPF-M, the dPF-G gives overall bad results when trained with L_{nll}.

In summary, we recommend using L_{nll} to ensure learning accurate noise models. If learning the process and sensor model does not work well, L_{nll} can be combined with L_{mse} or the models can be pretrained.

D. Training Sequence Length

Karkus et al. [13] found that using longer training sequences improved results for their dPF. Here we aim to find a sequence length with a good trade off between training speed and model performance. More details can be found in Appendix VII-F.

Summary of Results: The results in Fig. 2 show that longer sequences are beneficial for training DFs, because they demonstrate error accumulation during filtering and allow for convergence of the state estimate when the initial state was noisy. However, performance eventually saturates and increasing l also increased our training times. We therefore chose $l = 10$ for all experiments, which provides a good trade-off between training speed and performance.

E. Learning Noise Models

The following experiments analyse if and how well complex models of the process and observation noise can be learned

	const. $\mathbf{R}$			hetero. $\mathbf{R}$			
	RMSE	nll	D_Q	RMSE	nll	corr	D_Q
dEKF	23.2	32.3	0.08	16.4	**30.1**	-0.63	**0.002**
dUKF	24.0	32.3	0.238	16.6	**30.1**	-0.65	0.01
dMCUKF	22.5	32.4	0.354	16.7	30.3	**-0.67**	0.009
dPF-M	**14.0**	**30.3**	**0.054**	**12.8**	**30.1**	-0.12	0.032

TABLE I: Results for learning constant or heteroscedastic observation noise $\mathbf{R}$ and constant process noise $\mathbf{Q}$ on a dataset with $\sigma_{q_p} = 3$ and 30 distractors. Corr. is the correlation coefficient between $\mathbf{R}$ and visible target pixels. $D_{\mathbf{Q}}$ denotes the Bhattacharyya distance between true and learned process noise.

through the filters.

To isolate the effect of the noise models, we use a fixed, pretrained sensor model and the analytical process model, such that only the noise models are trained. More details on the experiments and results can be found in Appendix VII-G.

1) Heteroscedastic Observation Noise: We compare DFs that learn constant or heteroscedastic observation noise on different datasets to see if learning more complex, heteroscedastic observation noise models improves performance. The process noise models are constant, but we test if the DFs can learn different magnitudes of the positional process noise $\mathbf{q_p}$.

We evaluate the process noise model using the Bhattacharyya distance. To test if higher observation noise is predicted when the red disc is not visible, we compute the correlation coefficient between predicted $\mathbf{R}$ and the number of visible target pixels.

Summary of Results: Table I shows results for one dataset. With constant observation noise models, all DFs except for the dPF-M perform poorly: To avoid wrong updates of the state estimate when the disc is occluded, they learned to ignore all observations by predicting a high $\mathbf{R}$.

Like [7], we find that learning heteroscedastic observation noise solves this problem and increases the tracking performance significantly.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.

Learning the process noise model $\mathbf{Q}$ works better when the tracking performance is good. All DFs have more difficulties when the ground truth noise has a low magnitude.

2) Heteroscedastic Process Noise: The effect of heteroscedastic process noise has not yet been evaluated in related work. We create datasets with heteroscedastic 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.

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

Summary of Results: Table II shows that learning heteroscedastic noise models for the dynamics is more difficult than for the observations. This is not surprising, as the input values for predicting the $\mathbf{Q}$ are the noisy state estimates.

Plotting the predictions for $\mathbf{Q}$ (see Appendix Fig. 4) reveals that all DFs learn to follow the true values for the velocity noise $\mathbf{q_v}$ relatively well, but also predict state dependent values for $\mathbf{q_p}$, which is actually constant. Despite not being completely accurate, learning heteroscedastic process noise models still reliably improves the performance of all DFs.

	const. $\mathbf{Q}$			hetero. $\mathbf{Q}$		
	RMSE	nll	D_Q	RMSE	nll	D_Q
dEKF	**12.2**	29.7	**0.864**	**11.6**	**29.0**	**0.351**
dUKF	12.3	**29.6**	0.867	11.9	29.1	0.4
dMCUKF	12.3	29.7	0.882	11.8	29.1	0.42
dPF-M	12.6	30.4	0.936	11.9	29.9	0.589

TABLE II: Results for learning constant or heteroscedastic process noise $\mathbf{Q}$ on a dataset with heteroscedastic $\mathbf{q_p}$, $\sigma_{q_p} = 3.0$ and 30 distractors. $D_{\mathbf{Q}}$ is the Bhattacharyya distance between true and learned process noise.

	$\sigma_{q_p} = 0.1$		$\sigma_{q_p} = 3.0$		$\sigma_{q_p} = 9.0$	
	RMSE	nll	RMSE	nll	RMSE	nll
dEKF	9.0	27.1	10.9	**28.0**	**17.6**	29.5
dUKF	9.1	27.1	12.3	28.7	18.7	29.5
dMCUKF	**8.7**	**27.0**	11.4	28.2	18.2	**27.3**
dPF-M	9.2	28.4	**10.1**	29.0	20.0	34.7
LSTM-1	11.0	27.7	14.2	29.0	22.9	30.4
LSTM-2	9.0	27.2	11.9	28.4	19.5	29.9

TABLE III: Comparison between the DFs and LSTM models with one or two LSTM layers on three different datasets with 30 distractors and process noise with increasing magnitude.

F. Benchmarking

In the final experiment, we compare the performance of the DFs to two LSTM models. We use an LSTM architecture similar to [10], 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 noise models. We compare their performance on datasets with 30 distractors and different levels of constant process noise.

Summary of Results: The results in Table III show that all models track target disc well and make reasonable uncertainty predictions. While there is no significant difference between the different DF variants, the LSTM model, however, needs two layers of LSTM cells to reach the performance of the DFs.

Unstructured models like LSTM can thus learn to perform similar to differentiable filters, but they require a much higher number of trainable parameters than the DFs. They are also harder to analyse since they do not use explicit models of the dynamics, the observation or the process noise.

VI. Conclusions & Future Work

Our experiments have shown that all DFs we evaluated are well suited for learning both, sensor and process model, as well as the associated noise models. Only the dPF behaved differently from the other DFs in some experiments when its belief was represented by a GMM. LSTMs can reach the same performance, but need significantly more trainable weights.

The system we used here has relatively simple dynamics without strong nonlinearities. In future work, we will thus test the DFs on more challenging, real-world problems. We will also try using even more complex, correlated noise models.

The main challenge in working with differentiable filters is keeping the training stable and finding good values for the numerous hyperparameters of the filters. While we hope that our work provides some orientation, we still recommend the dEKF for getting started. It is not only the most simple of our DFs, but also the most numerically stable during training.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Evan Archer, Il Memming Park, Lars Buesing, John Cunningham, and Liam Paninski. Black box variational inference for state space models. *arXiv preprint arXiv:1511.07367*, 2015.

[3] M. Bauza and A. Rodriguez. A probabilistic data-driven model for planar pushing. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3008–3015, May 2017. doi: 10.1109/ICRA.2017.7989345.

[4] Vinay A. Bavdekar, Anjali P. Deshpande, and Sachin C. Patwardhan. Identification of process and measurement noise covariance for state and parameter estimation using extended kalman filter. *Journal of Process Control*, 21(4):585 – 601, 2011. ISSN 0959-1524. doi: https://doi.org/10.1016/j.jprocont.2011.01.001. URL http://www.sciencedirect.com/science/article/pii/S0959152411000023.

[5] 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.

[6] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113. IET, 1993.

[7] 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.

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

[9] Rico Jonschkowski and Oliver Brock. End-to-end learnable histogram filters. 2016.

[10] Rico Jonschkowski, Divyam Rastogi, and Oliver Brock. Differentiable particle filters: End-to-end learning with algorithmic priors. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, USA, 2018.

[11] S. J. Julier. The scaled unscented transformation. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, volume 6, pages 4555–4559 vol.6, 2002.

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

[13] Peter Karkus, David Hsu, and Wee Sun Lee. Particle filter networks with application to visual localization. In *Conference on Robot Learning*, pages 169–178, 2018.

[14] Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *International Conference on Learning Representations (ICLR)*, 2017.

[15] Kristian Kersting, Christian Plagemann, Patrick Pfaff, and Wolfram Burgard. Most likely heteroscedastic gaussian process regression. In *Proceedings of the 24th international conference on Machine learning*, pages 393–400. ACM, 2007.

[16] Jeffrey K. Uhlmann Simon J. Julier. New extension of the kalman filter to nonlinear systems. *Proc.SPIE*, 3068: 3068 – 3068 – 12, 1997. doi: 10.1117/12.280797. URL https://doi.org/10.1117/12.280797.

[17] H.W. Sorenson. *Kalman Filtering: Theory and Application*. IEEE Press selected reprint series. IEEE Press, 1985. ISBN 9780879421915. URL https://books.google.de/books?id=2pgeAQAAIAAJ.

[18] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[19] Jaleel Valappil and Christos Georgakis. Systematic estimation of state noise statistics for extended kalman filters. *AIChE Journal*, 46(2):292–308, 2000.

[20] Rudolph Van Der Merwe. Sigma-point kalman filters for probabilistic inference in dynamic state-space models. 2004.

[21] 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.

[22] M. Wüthrich, C. Garcia Cifuentes, S. Trimpe, F. Meier, J. Bohg, J. Issac, and S. Schaal. Robust gaussian filtering using a pseudo measurement. In *Proceedings of the American Control Conference*, Boston, MA, USA, July 2016. URL http://arxiv.org/abs/1509.04072.

[23] Michael Zhu, Kevin Murphy, and Rico Jonschkowski. Towards differentiable resampling, 2020.

VII. Appendix

A. Bayesian Filter Algorithms

1) Kalman Filter: The Kalman Filter [12] is a closed-form solution to the filtering problem for systems with a linear process and observation model and Gaussian additive noise. The belief about $\mathbf{x}$ is represented by the mean μ and covariance matrix $\mathbf{\Sigma}$. At each timestep, the filter predicts $\hat{\mu}$ and $\hat{\mathbf{\Sigma}}$ using the process model. The innovation $\mathbf{i}_t$ is the difference between the predicted and actual observation and is used to correct the prediction. The Kalman Gain K trades-off the process noise Q and the observation noise R to determine the magnitude of the update.

Prediction Step:

$$\hat{\boldsymbol{\mu}}_t = \mathbf{A}\boldsymbol{\mu}_{t-1} + \mathbf{B}\mathbf{u}_t \quad (4) \qquad \hat{\mathbf{\Sigma}}_t = \mathbf{A}\mathbf{\Sigma}_{t-1}\mathbf{A}^T + \mathbf{Q}_{t-1} \quad (5)$$

Update Step:

$$\mathbf{S}_t = \mathbf{H}\hat{\mathbf{\Sigma}}_t\mathbf{H}^T + \mathbf{R}_t \quad (6) \qquad \mathbf{i}_t = \mathbf{z}_t - \mathbf{H}\hat{\boldsymbol{\mu}}_t \quad (8)$$

$$\mathbf{K}_t = \hat{\mathbf{\Sigma}}_t\mathbf{H}^T\mathbf{S}_t^{-1} \quad (7) \qquad \boldsymbol{\mu}_t = \hat{\boldsymbol{\mu}}_t + \mathbf{K}_t\mathbf{i}_t \quad (9)$$

$$\mathbf{\Sigma}_t = (\mathbf{I}_n - \mathbf{K}_t\mathbf{H})\hat{\mathbf{\Sigma}}_t \quad (10)$$

2) Extended Kalman Filter (EKF): The EKF [17] extends the Kalman Filter to systems with non-linear process and observation models. It replaces the linear models for predicting $\hat{\mu}$ in Eq. 4 and the corresponding observations $\hat{\mathbf{z}}$ in Eq. 8 with non-linear models f and h . For predicting the state covariance Σ and computing the Kalman Gain $\mathbf{K}$, these non-linear models are linearized around the current mean of the belief. The Jacobians $\mathbf{F}_{|\mu_t}$ and $\mathbf{H}_{|\mu_t}$ replace $\mathbf{A}$ and $\mathbf{H}$ in Equations 5 - 7 and 10. This first-order approximation can be problematic for systems with strong non-linearity, as it does not take the uncertainty about the mean into account [20].

3) Uncentered Kalman Filter (UKF): The UKF [16, 20] was proposed to address the aforementioned problem of the EKF. Its core idea, the *Unscented Transform* [16], is to represent a Gaussian random variable that undergoes a nonlinear transformation by a set of specifically chosen points in state space, the so called *sigma points* $\boldsymbol{\chi} \in \mathbf{X}$. The statistics of the transformed random variable can be then be calculated from the transformed sigma points. For example, in the prediction step of the UKF, the non-linear transform is the process model (Equation 13) and the new mean and covariance of the belief are computed in Equations 14 and 15.

$$\boldsymbol{\chi}^0 = \boldsymbol{\mu} \qquad \boldsymbol{\chi}^i = \boldsymbol{\mu} \pm (\sqrt{(n+\lambda)\mathbf{\Sigma}})_i \ \forall i \in \{1...n\} \quad (11)$$

$$w^0 = \frac{\lambda}{\lambda + n} \quad w^i = \frac{0.5}{\lambda + n} \qquad \forall i \in \{1...2n\} \quad (12)$$

$$\hat{\mathbf{X}}_t = f(\mathbf{X}_{t-1}, \mathbf{u}_t) \quad (13) \qquad \hat{\boldsymbol{\mu}}_t = \sum_i w^i \hat{\chi}_t^i \quad (14)$$

$$\hat{\mathbf{\Sigma}}_t = \sum_i w^i(\hat{\chi}_t^i - \hat{\boldsymbol{\mu}}_t)(\hat{\chi}_t^i - \hat{\boldsymbol{\mu}}_t)^T + \mathbf{Q}_t \quad (15)$$

The parameter λ controls the spread of the sigma points and how strongly the original mean $\boldsymbol{\chi}^0$ is weighted in comparison to the other sigma points.

In theory, the UKF conveys the nonlinear transformation of the covariance more faithfully than the EKF and is thus better suited for strongly non-linear problems [18]. In contrast to the EKF, it also dos not require computing the Jacobian of the process and observation models, which can be advantageous when those models are learned.

In practice, tuning λ can be difficult since placing the sigma points too far from the mean increases prediction uncertainty and can even destabilize the filter. [16] suggested to chose λ such that $\lambda + n = 3$. This however results in negative values of λ if $n > 3$, for which the estimated covariance matrix is not guaranteed to be positive semidefinite any more. This problem can be solved by changing the way in which $\mathbf{\Sigma}$ is computed [11]. However, for $-n < \lambda < 0$, the sigma point $\boldsymbol{\chi}^0$, which represents the original mean, is also weighted negatively. This not only seems counter-intuitive but strongly negative w^0 can also cause divergence of the estimated mean.

Besides from the parametrization we use here, there also exists a scaled variant of the UKF that was introduced in [11] and is frequently used, e.g. in [20, 18]. It sets $\lambda = \alpha^2(\kappa + n) - n$ and suggests using $\kappa = 0$ and small positive values for α, like $1e - 3$. In addition, this formulation uses a different w^0 when calculating $\mathbf{\Sigma}$: $w^0 = \frac{\lambda}{\lambda+n} + (1 - \alpha^2 + \beta)$, where $\beta = 2$ is recommended if the true distribution of the system is Gaussian. The suggested parameters however again result in large negative weights for $\boldsymbol{\chi}^0$ and thus destabilized the filter in our experiments. While the filter becomes stable for larger values of α, we chose to keep the simpler parametrization introduced above.

4) Monte Carlo Unscented Kalman Filter (MCUKF): The UKF represents the belief over the state with as few sigma points as possible. However, finding the correct scaling parameter λ can be difficult, especially if the state is high dimensional. Instead of relying on the unscented transform to calculate the mean and covariance of the next belief, we can also resort to Monte Carlo methods, as proposed by [22]. In practice, this means replacing the carefully constructed sigma points and their weights in Equations 11, 12 with uniformly weighted samples from the current belief. The rest of the UKF algorithm stays the same, but more samples are necessary to represent the distribution of the belief accurately.

5) Particle Filter (PF): In contrast to the different variants of the Kalman Filter explained before, the Particle Filter [6] does not assume a parametric representation of the state distribution. Instead, it represents the belief with a set of *particles*. This allows the filter to track multiple hypotheses about the state at the same time and makes it a popular choice for tasks like localization or visual object tracking [18].

An initial set of particles $\boldsymbol{\chi} \in \mathbf{X}_0$ is drawn from some prior belief and initialized with uniform weights π. At each timestep, new particles are generated by applying the process model to the previous particles and sampling additive process noise:

$$\mathbf{X}_t = f(\mathbf{X}_{t-1}, \mathbf{u}_t, \mathbf{q}_t) \quad (16)$$

Given an observation $\mathbf{z}_t$, the weight of each particle $\boldsymbol{\chi}_t^i$

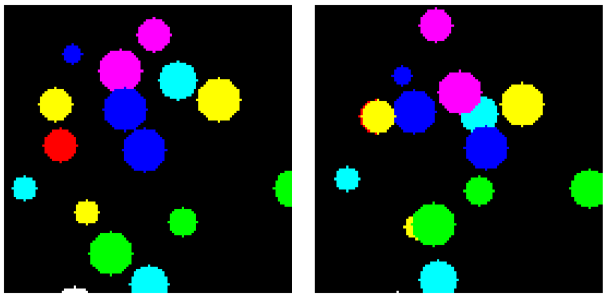

Fig. 3: Two sequential observations from our simulated task. The filters need to track the red disc, which can be occluded by the other discs or leave the image temporarily.

is updated based on the likelihood $p(\mathbf{z}_t|\boldsymbol{\chi}^i_t)$ by $\boldsymbol{\chi}^i$: $\pi^i_t = \pi^i_{t-1} p(\mathbf{z}_t|\boldsymbol{\chi}^i_t)$.

A potential problem of the PF is particle deprivation: Over time, many particles will receive a very low likelihood $p(\mathbf{z}_t|\boldsymbol{\chi}^i_t)$, and eventually the state would be represented by too few particles with high weights. To prevent this, a new set of particles with uniform weights can be drawn (with replacement) from the old set according to the weights. This resampling step focuses the particle set on regions of high likelihood and is usually applied after each timestep.

B. Simulated Environment

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

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

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.

Figure 3 shows two example images from the recorded data. We create multiple datasets with varying numbers of distractors, different levels of process noise for the disc position and constant or heteroscedastic process noise. All datasets contain 2400 sequences for training, 300 validation sequences and 303 sequences for testing. The sequences have 50 steps and the colours and sizes of the distractors are drawn randomly for each sequence.

C. Training

Unless stated otherwise, we train the DFs end-to-end on subsequences of length 10 for 15 epochs. Testing is done on the full sequences (50 steps). During training, the initial state is perturbed with noise sampled from a Gaussian with $\boldsymbol{\sigma} = 5$. For testing, we evaluate all DFs with five different, fixed perturbations (sampled from the same distribution) and average the results. The initial covariance for dEKF, dUKF and dMCUKF are set accordingly. For the dPF variants, we sample the initial particles around the perturbed state using the same Gaussian distribution. When training from scratch, we initialize $\mathbf{Q}$ and $\mathbf{R}$ with $\mathbf{Q} = 100 * \mathbf{I}_4$ and $\mathbf{R} = 900 * \mathbf{I}_2$, reflecting the high uncertainty of the untrained models.

D. Implementation and Hyperparameters

In the first set of experiments, we evaluate different implementation choices and hyper parameters for the four filters. We seek settings for the DFs that not only perform well but also increase the stability of the filter during training.

Experiments: We evaluate the DFS on a dataset with 15 distractor discs and constant process noise with $\boldsymbol{\sigma}_{q_p} = 0.1$ and $\boldsymbol{\sigma}_{q_v} = 2$. All DFs are trained end-to-end and learn both the observation and the process model as well as heteroscedastic observation noise and constant process noise. We train on sequences of length 10 for 15 epochs using the likelihood loss (Eq. 2 or 3). Testing is done on the full sequences (50 timesteps).

dEKF: Of all DFs discussed here, the dEKF is the only one without hyperparameters or relevant implementation choices. It however requires the Jacobian of the process model $\mathbf{F}$. Tensorflow implements auto differentiation, but has (as of now) no native support for computing Jacobians in graph mode. While it can be done, it requires looping over the dimensions of the differentiated variable one by one, which we found to be relatively slow especially during graph construction. We therefore recommend to manually derive the Jacobians where applicable.

dUKF: The only hyperparameter of the dUKF, λ, determines 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.

We tested different values in $[-10, 10]$ for λ, but found no significant differences between results. The choice of λ is presumably more important for problems with strongly non-linear dynamics. We generally recommend using small λ but avoiding values for which the sigma point at the mean is weighted negatively ($-n < \lambda < 0$), since large negative weights can destabilize the filter (see Sec. VII-A3).

dMCUKF: In contrast to the dUKF, the dMCUKF simply samples pseudo sigma points around the current belief. Its only hyperparameter thus is the number of sampled points during training and testing. We train and evaluate the dMCUKF with different numbers of pseudo sigma points. The results show that as few as five points are enough for training the filter successfully and that using more than 100 points during training does not reliably improve the performance of the trained filters. The test performance saturates at around 500 evaluated points. These numbers can of course change when the state has fewer or more dimensions. In the following, we use 100 points during training and 1000 for testing.

dPF: The differentiable Particle Filter has the highest number of different implementation choices. The likelihood for the observation update can be implemented with a trained neural network as in [13, 10] or with an analytical Gaussian likelihood function. Learning the likelihood did not improve results in our setting and even decreased performance on more difficult datasets. We thus use the analytical likelihood function in all following experiments.

We also have to chose how to represent the belief of the filter for computing likelihoods (see Section IV-D). 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 is an additional hyperparameter. We evaluated values between $\mathbf{\Sigma} = \mathbf{I}$ and $\mathbf{\Sigma} = 100 * \mathbf{I}$ and found that $\mathbf{\Sigma} = \mathbf{I}$ gave the best results. Overall, dPF-M performed better than dPF-G, as can be seen in the next experiment (Sec. VII-E).

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[13, 10, 23]. Karkus et al. [13] 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 tradeoff is controlled by a parameter $\alpha \in [0, 1]$. The higher α, the more weight is put on the uniform distribution. An alternative to soft resampling is not resampling at every timestep. We test different values for α with resampling every 1, 2 or 5 steps or not resampling at all.

Our results show that resampling frequently improves the filter performance, especially for both dPF-G and dPF-M. This is in contrast to the results in [13], where resampling every second step improved performance in comparison to resampling every step. Soft-resampling also did not have a positive effect in our experiments, since higher values of α decrease the effectiveness of the resampling step. We still chose to use soft resampling, but with a very small value of $\alpha = 0.05$

Finally, the user also has to decide how many particles to use during training and testing. We did the same experiment as for the dMCUKF with the dPF-M and got similar results. In the following, we thus use 100 particles during training and 1000 for testing.

E. Loss Function

In this experiment we compare the different loss functions introduced in Sec. IV-D, as well as a combination of the two $L_{mix} = 0.5(L_{mse} + L_{nll})$. 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.

We also compare the two alternative ways for representing the belief in a dPF (dPF-G and dPF-M). As our test scenario does not require tracking multiple hypotheses, the representation by a single Gaussian in dPF-G should be accurate.

Experiment: Here, we only evaluate the dPF variants and the dEKF, since dUKF and dMCUKF behave very similar to the dEKF. We use a dataset with 15 distractors and constant process noise ($\boldsymbol{\sigma}_{q_p} = 0.1$, $\boldsymbol{\sigma}_{q_v} = 2$). 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 better likelihoods scores than training on L_{mse}, see Fig. 1. The best tracking errors on the other hand are reached with L_{mse}, as well as more precise sensor models.

The only exception is the dPF-G, where training on L_{nll} gives overall bad results. 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 hypothesis.

For judging the quality of a DF, both likelihood 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.

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 opposite gradients: Both dEKF and dPF overestimate the process noise when trained on L_{mse}. This lowers the tracking error by giving more weight to the observations in the dEKF and allowing more exploration in the dPFs. But it also results in a higher uncertainty about the state, which is undesirable when optimizing the likelihood.

We generally found training the dPFs on the L_{nll} more difficult, since the likelihood computation can become numerically unstable when the weights are small or the particle set is not diverse. However, we still recommend using L_{nll} to ensure learning accurate noise models. If learning the process and sensor model does not work well , L_{nll} can be combined with L_{mse} or the models can be pretrained.

F. Training Sequence Length

Karkus et al. [13] tested training their dPF on sequences of length $l \in [1, 2, 4]$ and found that using more steps improved results. 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 l with a good trade off between training speed and model performance.

Experiment: We evaluate only dPF-M and dEKF on a dataset with 15 distractors and constant process noise ($\boldsymbol{\sigma}_{q_p} = 0.1$, $\boldsymbol{\sigma}_{q_v} = 2$). 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 $l \in [1, 2, 5, 10, 25, 50]$ while keeping the total number of examples per batch (steps $\times$ batch size) constant.

Results: Our results in Figure 2 show that both filters benefit from longer training sequences much more than the results in [13] 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 corrupting the initial state, but the results with $l \in [1, 2]$ 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 when the initial state was noisy. However, performance eventually saturates and increasing l also increased our training times. We therefore chose $l = 10$ for all experiments, which provides a good trade-off between training speed and performance.

G. Learning Noise Models

The following experiments analyse if and how well complex models of the process and observation noise can be learned through the filters.

To isolate the effect of the noise models, we use a fixed, pretrained sensor model and the 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 with L_{nll} on different datasets with increasing numbers of distractors as well as increasing positional process noise.

1) Heteroscedastic Observation Noise: We start with datasets with constant process noise and increasing numbers of distractors as well as the positional process noise $\mathbf{q_p}$. We want to see if the DFs can adapt to the different $\mathbf{Q}$ and if learning more complex, heteroscedastic observation noise models improves the performance of the filters. For this, we compare DFs that learn constant or heteroscedastic observation noise, while the process noise is restricted to be constant.

We evaluate the process noise model using the Bhattacharyya distance. 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.

Results: Full results are shown in Table IV. When learning constant observation noise, the dPF-M is the only filter that performs well in terms of tracking error: All other filters, (including the dPF-G) learn a very high $\mathbf{R}$ and thus mostly rely on the process model for their prediction. This is expected, since trusting the observations would result in wrong updates to the mean state estimate when the target disc is occluded. The PF-M does not use the mean of the particles in the likelihood computation, which makes it less sensitive to wrong observations and allows it to learn a lower $\mathbf{R}$.

Like [7], 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. Only the dPF-M sometimes fails to learn this correlation well: Since it already performs well with constant observation noise, it has less incentive to use state-dependent observation noise.

Finally, all DFs learn values of $\mathbf{q_v}$ that are close to the ground truth. For the position noise $\mathbf{q_p}$, however, we see a difference between learning constant or heteroscedastic

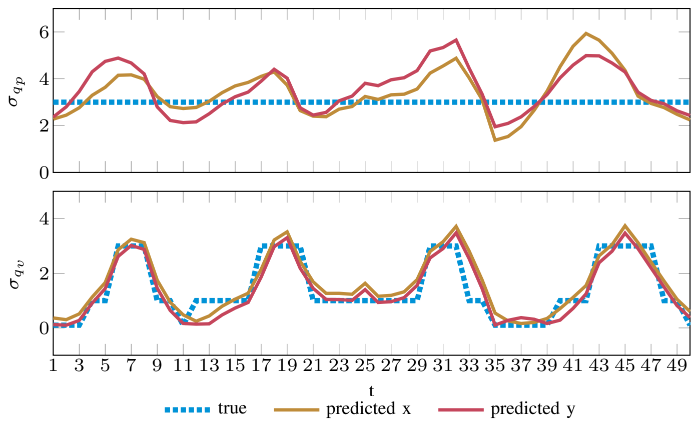

Fig. 4: Predicted and true process noise from the dEKF over one test sequence. The ground truth process noise has the same σ for both coordinates.

observation noise: On the datasets with lower ground truth process noise, $\mathbf{q_p}$ is overestimated by all DFs. Results are especially bad when the learned observation noise model is constant. This could be because the bad tracking performance with constant observation noise prevents learning an accurate process model. The results for learning $\mathbf{q_p}$ indeed improve if we enable learning a better process model by using the true initial state instead of a noisy one.

2) Heteroscedastic Process Noise: The effect of heteroscedastic process noise has not yet been evaluated in related work. We create datasets with heteroscedastic 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.

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

Results: As shown in Table V, 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 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 can see the same behaviour also on dataset with constant $\mathbf{q_v}$. We thus assume that the models rather pick up an unintentional pattern in our data: The probability of the disc being occluded indeed 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, the DFs were able to improve their likelihood scores slightly by learning “wrongly” heteroscedastic noise models.

			$\sigma_{q_p} = 0.1$				$\sigma_{q_p} = 3.0$				$\sigma_{q_p} = 9.0$			
		R	RMSE	nll	corr.	$D_{\mathbf{Q}}$	RMSE	nll	corr.	$D_{\mathbf{Q}}$	RMSE	nll	corr.	$D_{\mathbf{Q}}$
5 distractors	dEKF	const.	22.0	32.1	-	2.776	24.6	32.6	-	0.088	44.3	34.1	-	0.016
		hetero.	15.5	30.2	-0.71	0.66	14.9	29.7	-0.75	0.002	30.5	33.0	-0.61	0.003
	dUKF	const.	22.7	32.1	-	2.892	25.1	32.7	-	0.085	44.7	34.2	-	0.035
		hetero.	15.7	30.3	-0.7	1.216	15.3	29.8	-0.75	0.005	29.9	31.9	-0.66	0.005
	dMCUKF	const.	22.2	32.1	-	3.138	25.6	32.6	-	0.152	45.3	34.2	-	0.03
		hetero.	15.6	30.4	-0.71	0.031	15.1	29.8	-0.75	0.006	30.0	31.9	-0.68	0.005
	dPF-G	const.	23.1	32.1	-	3.167	26.1	32.6	-	0.161	43.1	34.2	-	0.033
		hetero.	17.7	30.9	-0.61	3.007	19.0	31.2	-0.52	0.179	33.5	33.1	-0.63	0.088
	dPF-M	const.	13.8	30.4	-	2.694	15.1	30.7	-	0.045	33.0	37.9	-	0.001
		hetero.	14.7	30.8	-0.42	2.48	14.0	30.5	-0.29	0.029	29.4	37.3	-0.78	0.031
30 distractors	dEKF	const.	21.0	32.2	-	2.89	25.6	32.4	-	0.067	44.4	34.1	-	0.024
		hetero.	13.4	29.5	-0.67	0.96	18.7	31.2	-0.89	0.003	33.6	34.8	-0.75	0.006
	dUKF	const.	21.6	32.4	0	2.812	26.0	32.5	-	0.138	44.7	34.2	-	0.04
		hetero.	12.1	28.7	-0.57	0.756	19.0	31.3	-0.9	0.003	34.3	35.0	-0.82	0.016
	dMCUKF	const.	22.0	32.2	-	3.253	26.1	32.5	-	0.155	45.3	34.2	-	0.03
		hetero.	11.3	28.5	-0.6	1.423	18.8	31.1	-0.89	0.004	36.0	34.0	-0.78	0.02
	dPF-G	const.	22.7	32.2	-	3.151	26.6	32.4	-	0.148	43.5	34.1	-	0.041
		hetero.	16.2	30.6	-0.54	3.106	19.6	31.5	-0.59	0.13	39.3	33.8	-0.52	0.038
	dPF-M	const.	14.1	30.6	-	2.684	18.1	32.5	-	0.045	35.9	38.9	-	0.005
		hetero.	15.2	30.8	-0.53	2.666	18.5	33.2	-0.65	0.08	33.6	39.5	-0.68	0.202

TABLE IV: Results for learning only the noise models through the DFs. 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 the learned process noise model ($D_{\mathbf{Q}}$).

		$\sigma_{q_p} = 0.1$			$\sigma_{q_p} = 3.0$			$\sigma_{q_p} = 3.0$, $\sigma_{q_v} = 2.0$			$\sigma_{q_p} = 9.0$		
	Q	RMSE	nll	$D_{\mathbf{Q}}$	RMSE	nll	$D_{\mathbf{Q}}$	RMSE	nll	$D_{\mathbf{Q}}$	RMSE	nll	$D_{\mathbf{Q}}$
dEKF	const.	9.5	28.4	2.268	12.2	29.7	0.864	18.7	31.2	0.003	29.1	32.5	1.119
	hetero.	8.4	27.4	1.705	11.6	29.0	0.351	17.9	31.1	0.097	27.6	32.1	0.892
dUKF	const.	9.4	28.2	2.819	12.3	29.6	0.867	19.0	31.3	0.003	30.5	32.8	1.056
	hetero.	8.6	27.4	1.679	11.9	29.1	0.4	18.3	31.2	0.075	30.2	32.3	0.968
dMCUKF	const.	9.5	28.2	2.972	12.3	29.7	0.882	18.8	31.1	0.004	30.7	32.6	1.186
	hetero.	8.6	27.5	1.915	11.8	29.1	0.42	18.5	31.0	0.053	29.9	32.4	1.043
dPF-G	const.	12.4	29.7	3.984	14.1	30.3	1.126	19.6	31.3	0.130	31.5	32.6	1.216
	hetero.	12.1	29.4	3.695	13.8	29.8	0.755	20.0	31.2	0.216	31.0	32.4	1.055
dPF-M	const.	11.3	29.6	3.744	12.6	30.4	0.936	18.5	33.2	0.08	30.1	37.6	0.995
	hetero.	9.6	28.7	3.327	11.9	29.9	0.589	18.8	32.6	0.118	28.7	37.4	0.722

TABLE V: Results for learning constant or heteroscedastic process noise $\mathbf{Q}$ on datasets with 30 distractors and different heteroscedastic or constant ($\sigma_{q_p} = 3.0$, $\sigma_{q_v} = 2.0$) process noise. $D_{\mathbf{Q}}$ is the Bhattacharyya distance between true and learned process noise.