Eirik Ekjord Vesterkjær

Combining grid-based uncertainty propagation and neural networks with uncertainty estimation

Master's thesis in Engineering Cybernetics

Supervisor: Adil Rasheed

June 2020

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of Science and Technology

Eirik Ekjord Vesterkjær

Combining grid-based uncertainty propagation and neural networks with uncertainty estimation

Master's thesis in Engineering Cybernetics
Supervisor: Adil Rasheed
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

We can apply grid-based methods for deterministic uncertainty propagation in combination with uncertainty estimation algorithms for neural networks. This can be used to model dynamical processes and their uncertainties using neural networks. Further, such methods can yield calibrated uncertainty estimates when forecasting with dissipative first order systems.

This master's thesis in engineering cybernetics develops and applies an approach to employing existing grid-based uncertainty propagation methods (e.g. *the unscented transform*) on machine learning algorithms which have a variable (*heteroscedastic*) output uncertainty. Further, the thesis explores a method of jointly learning the process dynamics and process uncertainty in a dynamical system. This method applies recent advances in predictive uncertainty estimation for neural networks in combination with grid-based uncertainty propagation.

The methods are applied on a real world dataset, to model the temperature evolution in the main bearing of wind turbines at a Norwegian wind farm. It's found that the temperature confidence intervals predicted by a neural network-based method when simulating generalizes to other wind turbines. The results indicate that it is feasible to jointly model first order process dynamics and process uncertainty with neural networks. Though a trade-off between predictive accuracy and calibration was observed, this can likely be mitigated by fine-tuning the uncertainty estimation methods and optimization procedures.

The main contributions from this thesis are: *i)* Derivation of equations for applying grid-based uncertainty propagation methods on systems with variable output uncertainty, *ii)* Demonstration of how this can be used to model process dynamics and process noise using neural networks, and *iii)* Application of the methods to model the temperature of a wind turbine main bearing with uncertainty estimates.

Sammendrag

Punktbaserte metoder for deterministisk videresending av usikkerhet kan benyttes i kombinasjon med usikkerhetsestimering for nevrale nettverk. Dette kan brukes for å modellere dynamiske tilstandsromprosesser samt deres prosessusikkerhet ved hjelp av nevrale nettverk. Disse metodene kan gi kalibrerte usikkerhetsestimater når de benyttes for å modellere og simulere dissipative første ordens systemer.

Denne masteroppgaven innen kybernetikk viser hvordan deterministiske punktbaserte algoritmer for videresending av usikkerhet (f.eks. *the unscented transform*) kan benyttes på maskinlæringsalgoritmer med en variabel (*heteroskedastisk*) prediksjonsusikkerhet. Videre utforskes en metode for å lære systemdynamikk og tilknyttet usikkerhet ved hjelp av nevrale nettverk med usikkerhetsestimering. Her benyttes nylig utviklede metoder for usikkerhetsestimering i nevrale nettverk i kombinasjon med punkbaserte metoder for videresending av usikkerhet.

Metodene blir anvendt på et eksperiment med ekte data fra en vindmøllepark i Norge. De benyttes der for modellere temperaturutviklingen i lageret der akslingen til vindturbinene roterer. Resultatene viser at konfidensintervaller for temperatur som forutsies av et nevralt nettverk kan generalisere når nettverket anvendes på nye vindturbiner. Eksperimentet indikerer at det er mulig å modellere førsteordens dynamikk sammen med prosessusikkerhet ved hjelp av nevrale nettverk. Det ble observert en avveining mellom nøyaktighet i prediksjon og nøyaktighet i usikkerhetsestimater, men dette kan trolig reduseres ved å kombinere ulike usikkerhetsestimeringsmetoder og forbedre optimeringsprosedyren.

Hovedbidragene fra denne masteroppgaven er: *i)* Derivasjon av ligninger som lar oss anvende eksisterende punktbaserte algoritmer for videresending av usikkerhet på modeller med variabel prediksjonsusikkerhet, *ii)* Demonstrasjon av hvordan dette kan benyttes for å modellere prosessdynamikk og prosessusikkerhet ved hjelp av nevrale nettverk, og *iii)* Anvendelse av metodene for å modellere temperaturutviklingen i lageret til akslingen på ekte vindturbiner, med usikkerhetsmål.

Preface

This thesis is submitted for the degree of Master of Technology in Engineering Cybernetics (Siv.Ing., Kybernetikk og Robotikk), to the Department of Engineering Cybernetics at the Norwegian University of Science and Technology.

I want to extend my thanks to my advisors: Adil Rasheed (NTNU), Eivind Rosón Eide (Kongsberg Digital), and Gerthory Toussaint (Kongsberg Digital). To Adil, for initial ideas, and suggestions for project direction and experiments. To Eivind, for feedback on the methods, reviewing the report, and his hospitality while I was in Asker. To Gerthory, for assistance with gathering data and information on the main experiment, and his hospitality while I was at Grilstad.

Lastly, I am very grateful for all the moral support from my family during this period.

Code for the experiments in this thesis was primarily written in the `Python v3.6.9` [1] programming language. The `PyTorch` [2] library was used for neural network implementations, and the `AdaBound` optimizer [3] was used for optimizing model parameters. A differentiable unscented Kalman filter variant was implemented by me using `PyTorch`. The `FilterPy` [4] library was used for verification of this implementation. `SciPy` packages [5] were used for data handling and plot creation. Alongside this, `MATLAB R2019b` [6] was used for state space model identification. This report is based on the `COPSCE-NTNU` thesis template [7].

Contents

Figures

Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Advanced artificial intelligence (AI) implementations are rapidly becoming accessible, widespread, and remarkably successful at complex tasks [8] [9]. Neural network-based machine learning (ML) models have garnered significant interest in the past decades, as deep neural networks have challenged and pushed the state of the art within many fields - including time-series forecasting [10]. ML-based AI is enabling impressive technological advancements, but the inner workings of the state-of-the-art implementations is not well understood [11]. The consequence of this is that the integration of AI in safety-critical systems poses significant challenges [12].

Physics-based simulators are widely used for process representation, optimization and simulated safety tests. These enable high-fidelity simulations based on differential equations, without the safety implications of running the process they model. The potential in applying machine learning models in place of a traditional simulator lies in the possibility of speeding up the evaluation by several orders of magnitude. Such a possibility is clearly enticing - but it comes at a price: Substituting a physics-based model for a machine learning-based model arguably decreases the interpretability of the results. When can we then *trust* such a model to be correct? This is a question that necessitates an answer if we are to apply machine learning for predictive forecasting or decision making.

In order to make informed decisions, we need information on the *risk* in the outcomes that may entail from these decisions. As the risk associated with a method is dependent on the inherent uncertainty in said method [13], uncertainty estimation for industrial methods (e.g. [14]) have played an important role in developing methods applicable for safety-critical domains.

Uncertainty estimation can be valuable. Predicting that a prediction is erroneous in advance can make it possible to employ preventative measures before a failure takes place. Recent contributions have proposed several algorithms for estimating the predictive uncertainty in the output of neural networks [15] [16] [17]. These appear very promising, and have been demonstrated to perform well

in diverse applications [18] [19]. However, such methods are not fault proof: they still suffer when faced with data very dissimilar from what they have previously seen [20].

Even in case of a highly accurate model, uncertainty in its input can propagate to its output, yielding uncertain predictions. This becomes increasingly relevant as ML-based systems are becoming widespread: If we interconnect systems that each have an associated uncertainty, the propagation of uncertainty can be crucial to account for. There is extensive theory concerning propagation of input uncertainty through black-box functions [21] [22] [23] [24], much of which has roots in state estimation theory. *But can we apply this on a machine learning model with variable predictive uncertainty?*

This last question is what sparked the work in this thesis. It was posed in the context of dynamical systems modeling, and motivated by a simple idea: Replacing a state space model with a neural network that employs predictive uncertainty estimation. Could this be used to obtain both accurate predictions and accurate uncertainty estimates? The inspiration for this was work done in my specialization pre project [25] and an experiment proposed by Kongsberg Digital: Modeling the temperature evolution of a wind turbine. The idea of a model that fits straight into existing control theory approaches was enticing, and the approach was further explored. The result is what is presented in this thesis.

1.2 Objectives

The theme of this thesis was proposed by Kongsberg Digital. The goal of the project was to explore methods for estimating the uncertainty of predictions from ML models in an industrial setting.
The main objective of this thesis is to combine *model uncertainty estimation* techniques and *input uncertainty propagation* techniques in order to estimate the total uncertainty in the resulting prediction of a dynamical system. This developed into the following research questions,

1. How can we propagate uncertainty through a machine learning model whose output is uncertain? Can we apply existing techniques directly, or do we need to extend them?
2. Are existing uncertainty estimation methods applicable for small-scale neural networks?
3. Can small neural networks with uncertainty estimation be applied to model a dynamical system and it's uncertainties?

These are the questions the work in this thesis builds upon.

1.3 Contributions and Outline

The main contributions of this thesis are,

1. an approach to combining grid-based input uncertainty propagation with model uncertainty estimation for machine learning models.
2. a demonstration of how this can be applied to simulate and identify a model represented by a neural network with predictive uncertainty estimation
3. the application of these methods to model and estimate the uncertainty in the temperature evolution of wind turbines over long time horizons. This is done with real-world data from a wind farm in Norway.

1.3.1 Outline

This thesis shows how we can apply uncertainty estimation methods for neural networks in combination with grid-based uncertainty propagation techniques. Using this, neural networks with predictive uncertainty estimation are employed to develop a model which predicts the temperature in a wind turbine, along with the associated uncertainty. Two separate uncertainty estimation methods for neural networks are considered: *Deep Ensembles* and *Monte Carlo dropout*. The approach is evaluated against alternative methods, including system identification [26] combined with Bayesian model averaging [27].

We begin with a brief overview of relevant literature in Chapter 2. This first covers machine learning, model representation using neural networks and state space modeling. Following this, relevant uncertainty estimation methods are reviewed: Grid-based uncertainty propagation, filtering, and advances in uncertainty estimation for machine learning models.

Following the literature review, the main methods developed this thesis are presented in Chapter 3. We show how common grid-based uncertainty propagation methods can be extended to propagate uncertainty through equations with a variable output uncertainty. We then demonstrate how we can apply this to simulate a model whose dynamics are given by an stochastic difference equation, and how this can be extended for use in a Gaussian filtering framework. Lastly, the approach used in this thesis to model a dynamic system by combining neural networks (with uncertainty estimation) and grid-based uncertainty propagation methods is outlined.

A qualitative synthetic experiment is then considered in Chapter 4. This concerns simulating a dynamical system represented by a stochastic difference equation with variable process uncertainty.

The main experiment is presented in Chapter 5. This concerns a wind turbine temperature modeling task. Several models that estimate the main bearing tem-

perature within wind turbines are developed. These are based on real data from a wind farm in Norway, and motivated by physical energy balance equations. Both long-horizon simulations and one-step-ahead predictions when filtering are considered for evaluating the models.

The results from the experiments are then discussed in Chapter 6, and possible directions for future research are suggested. Lastly, we conclude in Chapter 7. The main results are there summarized, along with some concluding remarks.

1.3.2 A Final Note

It was chosen to present the information necessary to understand the main experiment (Chapter 5) in the same chapter instead of in the literature study, as to not switch context repeatedly in the thesis. This means that the literature study in Chapter 2 presents the foundations for the methods developed in Chapter 3, and Chapter 5 should be accessible in isolation.

Chapter 2

Literature Study

On order to apply uncertainty estimation on a machine learning model, a thorough understanding of the underlying theory is necessary. The methods presented in Chapter 3 build upon traditional techniques for dynamical process modeling, so an overview of these is also required.

The literature study first covers machine learning and neural networks, then dynamical process modeling. Following this, existing approaches to uncertainty estimation are presented. This covers both methods with origins in state estimation, and recent methods developed for neural networks. Lastly, a brief overview of the key observations from the literature study is presented.

2.1 Machine Learning

The field of machine learning (ML) concerns computer programs that adapt to improve their performance [28]. It can be employed for a wide variety of tasks, including *modeling*. System models represented by computer programs with *tuneable parameters* are applied in a wide range of domains, such as power electronics [29] and renewable energies [30], aerospace [31] and language modeling [8].

The perceptron [32] is an important early contribution to machine learning, as it laid the foundation for *deep neural networks* (DNNs). Machine learning models based on DNNs have demonstrated remarkable results for complex tasks including object detection [33] [34], physics modeling [35] [36] [37], and semantic segmentation [38] [39].

2.1.1 Advances in Neural Networks

A neural network [40] is a nonlinear mapping $f : \mathbb{R}^n \to \mathbb{R}^m$ parametrized by weights $\mathbf{W}$ and biases $\mathbf{B}$. Most commonly, it consists of multiple affine transformations (e.g. $\mathbf{W}^i\boldsymbol{x} + \mathbf{B}^i$ for fully connected networks) and nonlinear activations $\boldsymbol{\sigma}(\cdot)$ applied sequentially to an input. An affine transformation followed by an activation is often referred to as a layer - and *deep* neural networks can have hundreds of layers [41].

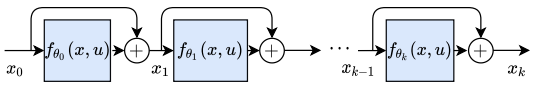


Figure 2.1: Residual Neural Network Architecture: A simple example of how a residual network can look. The residual blocks learn a difference $x_k - x_{k-1}$. This difference is commonly scaled by some factor α, omitted in this illustration.

Neural networks are commonly used for supervised learning, where they can be *trained* to learn patterns through stochastic optimization [42]. This consists of two main steps, *i)* a forward pass and loss computation, and *ii)* a backward pass and parameter optimization. In the forward pass, the network observes a set of *inputs* and makes *predictions*. These predictions are then evaluated using a *loss criterion*, a function which decreases the better the predictions are. For the backward pass, the backpropagation algorithm [42] is used to compute the gradient of the network parameters with respect to the loss function output. An optimization algorithm then updates the network parameters using the gradient.

Deep neural network architectures (in terms of layer count) can compose layers of abstraction to learn both low level features and abstract patterns [40]. This has led profound advances within complex fields such as computer vision [38] and language modeling [8].

A major obstacle to the success of these deep networks has been the difficulty in optimizing their parameters. Parallelization has been instrumental in this regard: Though networks may have thousands or millions of parameters, the computations can be expressed as vectorized operations, which lends to GPU acceleration. Rectifier activation functions [43] and residual architectures [41] (see Figure 2.1) helped combat the longstanding problem of vanishing gradients during optimization. Combined with faster optimization algorithms such as Adam [44] and Adabound [3], the result is that the computation time required for parameter optimization is substantially reduced. Regularization techniques such as dropout [45] can even further improve training speed, and additionally improve the generalization capabilities of the networks. The compounded impact of such developments have enabled models with billions of parameters and unprecedented representational capabilities [8].

The astounding results demonstrated by neural networks are undeniable: They have laid the foundation for important technological advances, including self driving [9]. Still, this success does not come without challenges. The complexity of neural networks can lead to significant obfuscation of their internal mechanics. *It is not well-understood how the most advanced models work* [11]. This has severe safety-implications. How can we trust that a model which we do not understand produces correct results? How can we determine when the model is malfunctioning? These challenges have led to a significant interest in uncertainty estimation for neural networks [46][16][17][20][47][15][48][49].

t	$y_1(t)$	$y_2(t)$
0.0	3.0	−1.5
0.1	2.5	3.2
0.2	4.1	1.1
⋮	⋮	⋮

Table 2.1: Time Series Data: Datasets for dynamical systems are typically given as time series data. These consist of measurements with an associated timestamp t.

2.2 Dynamical Process Modeling

The output of a dynamical process depends on past inputs. This necessitates methods to express how the process evolves over time. State space models are a widely used approach to representing dynamical processes, and these will be covered first. Following this, some methods to model dynamical systems using neural networks will be reviewed. Data sets for dynamical systems are commonly given as *time series*. These are sequences of data points that describe the evolution of some values over time - as shown in Table 2.1.

2.2.1 State Space Models

A state space model [50, p. 4] represents a dynamical system. It consists of a set of states $\boldsymbol{x} = [x_1, \ldots, x_n]^\top \in \mathbb{R}^n$, inputs $\boldsymbol{u} = [u_1, \ldots, u_p]^\top \in \mathbb{R}^p$, and outputs $\boldsymbol{y} = [y_1 \ldots y_m]^\top \in \mathbb{R}^m$, and associated first order differential equations which can be expressed compactly in vector form as

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t) \quad \in \mathbb{R}^n \\ \boldsymbol{y} &= \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}, t) \quad \in \mathbb{R}^m \end{aligned} \tag{2.1}$$

Where t is the time. Theory for system analysis, modeling, simulation, and control with state space models is thoroughly covered in [51], [52], and [50].

Systems represented as Equation (2.1) are *continuous-time* systems. They can be *simulated* using numerical integrators, e.g. Runge-Kutta methods [50, p. 526-567], to yield *discrete time systems* [51, p. 121-125] where the system dynamics are given by *difference equations*,

$$\begin{aligned} \boldsymbol{x}[k+1] &= \boldsymbol{f}_d(\boldsymbol{x}[k], \boldsymbol{u}[k], t) \quad \in \mathbb{R}^n \\ \boldsymbol{y}[k] &= \boldsymbol{h}(\boldsymbol{x}[k], \boldsymbol{u}[k], t) \quad \in \mathbb{R}^m \end{aligned} \tag{2.2}$$

where $t = t_0 + kh$ for a time step (sample time) of length h.

As real-world processes often exhibit stochastic behaviour (due to unmodeled phenomena and sensor noise, among other causes), process noise and measurement noise are commonly included as terms in state space models. For a discrete-time system, this can be expressed as,

$$\begin{aligned} \boldsymbol{x}[k+1] &= \boldsymbol{f}_d(\boldsymbol{x}[k], \boldsymbol{u}[k], \boldsymbol{w}[k], t) && \in \mathbb{R}^n \\ \boldsymbol{y}[k] &= \boldsymbol{h}(\boldsymbol{x}[k], \boldsymbol{u}[k], \boldsymbol{v}[k], t) && \in \mathbb{R}^m \end{aligned} \tag{2.3}$$

where $\boldsymbol{w}[k]$ and $\boldsymbol{v}[k]$ are, respectively, the process noise and sensor noise at step k. They are commonly assumed to be additive (multivariate) Gaussians with zero mean.

System Identification

When modeling physical systems with state space models, Equation (2.3) can often be derived from known quantities (e.g. masses and heat capacities) and the governing physical equations (e.g. mass balances, energy balances, geometrical constraints).

In other cases, the quantities or governing equations might not be entirely known, in which case they can be fully or partially estimated using machine learning. System identification is the process of determining the structure and parameters $\boldsymbol{\theta}$ of such dynamic system models based on observations of the system inputs and outputs, in order to obtain a state space model.

Offline system identification for linear state space models can be done using numerical algorithms for subspace identification (N4SID) such as described in [26] and [53]. On a high level, these obtain the (approximate) system parameters by finding the least squares' solution of a matrix equation expressing prediction error over time.

2.2.2 Time Series Modeling with Neural Networks

Neural networks are powerful function approximators, which makes them suitable for modeling complex temporal relations.

Recurrent neural networks have been widely applied to model sequences, and early examples of system modeling with them date back more than 20 years [54] [55]. Variants of *long short term memory networks* [56], a form of recurrent neural networks, have seen a great deal of success within forecasting [10] and speech recognition [57].

More recently, recently, the authors of [37] demonstrated how differentiable ODE solvers can be applied to *learn* and *simulate* systems governed by ordinary differential equations using neural networks. Even more interesting, their work establishes a connection between deep neural networks and dynamical systems: Residual connections [41], employed by most deep architectures, can be expressed

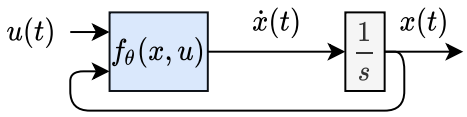


Figure 2.2: Neural ODEs: Ordinary differential equation represented by a neural network (f_θ), as demonstrated in [37]. By integrating the neural network predictions with a differentiable ODE solver ($1/s$), it becomes possible to optimize the neural network parameters using conventional stochastic optimization techniques. Recognize the similarity to the residual network in Figure 2.1: By unwrapping the ODEnet we can recover a similar structure to a residual network.

using differential equations. This lends to an interpretation of deep networks as psuedo-dynamical systems, where their *depth* is analogous to a *simulation length*.

The models explored in [37] are referred to as ODEnets. They were briefly explored in [25]. ODEnets can, very generally, be expressed as a differential vector field

$$\dot{\boldsymbol{x}} = \boldsymbol{f}_\theta(\boldsymbol{x}, t)$$

And the prediction from the network is given by the integral of the vector field over some time horizon $T_0 \dots T$, given an initial condition. Given constant time steps, they can be treated as difference equations. ODEnets are fascinating and highly relevant in context of recent interest in physics informed machine learning [35] [58] [36]. However, applications of neural networks to model difference equations date back to at least the early 2000s [59]. A benefit of this (illustrated well in [59] and [60]) is that such networks can be used to formulate a state space model (see Section 2.2.1). This enables the use of neural networks in traditional filtering and control algorithms.

2.3 Uncertainty Estimation

Uncertainty estimation can broadly be explained as as estimating the uncertainty associated with a prediction in advance, without necessarily knowing the ground truth. Uncertainty estimation is crucial for safe application of a method, as the associated risk is tied to the uncertainty in the possible outcomes [13]. Being able to reject an erroneous prediction before it causes a failure can be invaluable. Similarly, efficient information fusion that combines noisy estimates to yield accurate predictions is crucial for complex tasks like tracking and control.

There is a longstanding history of uncertainty estimation in cybernetics. Advanced control algorithms often require estimation of inner process states. Filtering algorithms have been widely employed for this purpose since the at least the

mid 19th-century [61]. More recently, uncertainty estimation for machine learning algorithms have seen increased interest. For the safe adoption of artificial intelligence algorithms and machine learning-based models, robust uncertainty estimation is a necessity [11]. This can be challenging for dynamical systems, as each time step predicted by a model will inject additional uncertainty into the estimate. However, as approximate inference for state space models is employed in traditional state estimation, this is a reasonable starting point to investigate.

This section will first cover traditional state estimation, grid-based uncertainty propagation, and filtering. Following this, uncertainty in machine learning will be reviewed. This will include a brief overview of the theory that many of these algorithms build upon: Bayesian statistics. Then, common classifications of the uncertainty and methods to estimate them will be covered.

2.3.1 State Estimation and Filtering

Dynamical systems such as ships and airplanes often employ *state estimators* in order to determine their internal (and often unmeasurable) states from noisy measurements.

Gaussian filtering algorithms are a widely used [62] [31] [19] [63] approach to state estimation. These build upon the Kalman filter [61], which has been applied in tracking, control, and sensor fusion tasks since the 1960s. Kalman filtering algorithms consist of a set of recursive equations: *1) A time update step* where the current state estimate is propagated in time to predict the next state, and *2) A measurement update step* where the propagated state estimate is corrected based on (noisy) measurements. As such, propagation of uncertainty is a key task performed by filtering algorithms.

Multivariate Gaussian Random Variables

Uncertainties in Kalman filtering algorithms are often expressed in terms of multivariate Gaussian distributions (*multivariate normal*). These can conveniently be represented using only the first two moments of the distribution: mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$. The probability density function of a multivariate Gaussian distribution is defined when the covariance is positive definite, in which case it is given by Equation (2.4).

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{n/2}\det(\boldsymbol{\Sigma})^{1/2}} \exp\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\}, \quad \mathbf{x} \in \mathbb{R}^n \tag{2.4}$$

A linear transformation applied on a multivariate Gaussian distribution yields a multivariate Gaussian distribution [64, eqs. 1.16.13-14]. Any multivariate Gaussian random variable $\boldsymbol{X} \in \mathbb{R}^n$ can therefore be expressed in terms of the standard multivariate Gaussian distribution $\boldsymbol{Z} \sim \mathcal{N}(\mathbf{0}_n, \mathbf{I}_{n \times n}) \in \mathbb{R}^n$:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \boldsymbol{\mu} + \sqrt{\boldsymbol{\Sigma}}\boldsymbol{Z} \tag{2.5}$$

Where $\boldsymbol{\Sigma} = \sqrt{\boldsymbol{\Sigma}}\sqrt{\boldsymbol{\Sigma}}^\top$ (which can e.g. be obtained by Cholesky factorization [65]).

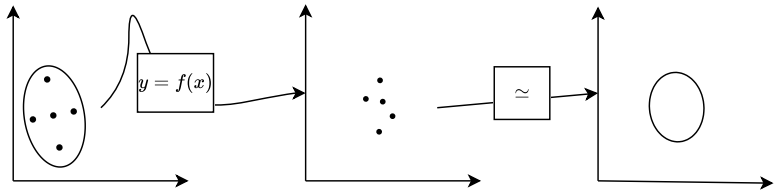


Figure 2.3: Grid-based Uncertainty Propagation: A continuous input distribution is approximated using deterministically chosen point coordinates and weights (*left*). The points are transformed (*center*), and the statistics of the propagated continuous distribution are approximated from the statistics of the weighted points (*right*). Algorithm 1 describes one method for implementing the procedure.

Grid-Based Gaussian Filters

Early extensions of the Kalman filter to nonlinear systems employ linearization [50, Chapter (1.2.3)] to apply the same recursive equations as the original filter [63]. The past decades, grid-based approaches have become more commonplace, in part motivated by the susceptibility of linearization to cause instability [66], which can for instance be seen in [25]. Grid-based filters approximate the propagation of state uncertainty using deterministic derivative-free numerical integration methods. Variants include the unscented Kalman filter [66] [67] [59], the Gauss-Hermite quadrature Kalman filter [68], and sparse grid quadrature filters [69] [70].

Consider a discrete time system on the form of Equations (2.6) and (2.7),

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{w}_k \tag{2.6}$$

$$\boldsymbol{y}_k = \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{v}_k \tag{2.7}$$

Where $\boldsymbol{f}$ is the state transition function, $\boldsymbol{h}$ is the observation function that relates the system state to external sensor measurements, and $\boldsymbol{w}$ and $\boldsymbol{v}$ are white noise terms with covariance $\mathbf{Q}$ and $\mathbf{R}$, respectively. Subscripts denote time step.

The key operation required for applying Kalman filtering equations to nonlinear systems is the propagation of the state estimate through the state transition function $\boldsymbol{f}$ and observation function $\boldsymbol{h}$. To do this, grid-based filters solve the same underlying equations as the original Kalman filter: The Bayesian filtering equations given in (2.8) and (2.9).

Propagation of a multivariate Gaussian state distribution $\boldsymbol{x}_{k-1} \sim \mathcal{N}(\hat{\boldsymbol{x}}_{k-1}, \mathbf{P}_{k-1})$ for a single time step can be expressed analytically as

$$p(\boldsymbol{x}_k) = \int_{\mathbb{R}^n} p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1} \tag{2.8}$$

Where $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ is given by Equation (2.6). Bayes rule can be used to express the conditional propagated state estimate after observing a sensor reading $\boldsymbol{y}_k$,

$$p(\boldsymbol{x}_k|\boldsymbol{y}_k) = \frac{p(\boldsymbol{y}_k|\boldsymbol{x}_k)p(\boldsymbol{x}_k)}{\int_{R^n} p(\boldsymbol{y}_k|\boldsymbol{x}_k)p(\boldsymbol{x}_k)d\boldsymbol{x}_k} \tag{2.9}$$

Where $p(\boldsymbol{y}_k|\boldsymbol{x}_k)$ is given by Equation (2.7). In an ideal world, we would be able to apply these equations directly and obtain the full propagated state distribution. In practice, the integrals over the stochastic variables will not be tractable.

The first two moments of the output distribution $p(\boldsymbol{x}_k)$ from the propagation can be found using Equation (2.8). Only these two first moments are needed to obtain a Gaussian approximation, which is the state distribution Gaussian filters maintain. Using the informal notation $E[\boldsymbol{x}^p]$ to refer to the p-th raw moment, this can be expressed as following for the state transition in Equation (2.6),

$$E[\boldsymbol{x}_k^p] = \int_{\mathbb{R}^n} \boldsymbol{x}_k^p p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})d\boldsymbol{x}_k \tag{2.10}$$

$$= \int_{\mathbb{R}^n} \boldsymbol{x}_k^p \left[\int_{\mathbb{R}^n} p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})p(\boldsymbol{x}_{k-1})d\boldsymbol{x}_{k-1} \right] d\boldsymbol{x}_k \tag{2.11}$$

$$= \int_{\mathbb{R}^n} \left[\int_{\mathbb{R}^n} \boldsymbol{x}_k^p \mathcal{N}(\boldsymbol{x}_k; \boldsymbol{f}(\boldsymbol{x}_{k-1}), \mathbf{Q}_k)d\boldsymbol{x}_k \right] \mathcal{N}(\boldsymbol{x}_{k-1}; \hat{\boldsymbol{x}}_{k-1}, \mathbf{P}_{k-1})d\boldsymbol{x}_{k-1} \tag{2.12}$$

Where Fubini's theorem and the fundamental theorem of calculus are used to change the order of integration. Equation (2.12) can be further rewritten using Equation (2.5). This makes it possible to express the integration in terms of a standard multivariate Gaussian probability density. The grid-based filters all employ approximations of Equation (2.12) expressed in terms of standard multivariate Gaussians,

$$\int_{\mathbb{R}^n} \boldsymbol{g}(\boldsymbol{z})\mathcal{N}(\boldsymbol{z}; \mathbf{0}_n, \mathbf{I}_{n\times n})d\boldsymbol{z} \simeq \sum_{i=1}^{N} w_i \boldsymbol{g}(\boldsymbol{z}_i) \tag{2.13}$$

Where $\boldsymbol{g}(\boldsymbol{z})$ is the inner integral in Equation (2.12) rewritten using Equation (2.5), and will depend on the moment. The same grid-based approximation can use used to express the covariances needed for the measurement update equations when filtering [70] [68]. It should be noted that Gaussian quadrature-based approaches [68] can compute the integrals *exactly* for Gaussian uncertainties. On the contrary, the unscented transform [66] (used in the unscented Kalman filter) only approximates the uncertainty propagation integrals [71, Chapter 4.4].

An practical interpretation is that the grid-based methods propagate a discrete approximation of the state distribution at each time step. The selection of the points $\boldsymbol{z}_i$ and associated weights w_i (along with the number (N) of such points) depends on the specific algorithm being used. Figure 2.3 illustrates how such grid based methods work.

Remark 2.3.1 (Grid-Based Numerical Integration) *In addition to the approximations mentioned here, numerical rules for other integral forms can be derived [72, Chapter 3.6]. Some are outlined in [73, Appendix A] If the probability distribution is not known but its moments can be computed, moment matching can be used to derive similar integration rules [21], or the unscented transform can be used [23].*

The Unscented Transform

The unscented transform is a grid based uncertainty propagation algorithm, used in the unscented Kalman filter. Many variants have been presented [71] [74] [67]. A version outlined in [66] is given in Algorithm 1.

Algorithm 1: The Unscented Transform

▷ Given $p(\boldsymbol{x}) \sim \mathcal{N}(\hat{\boldsymbol{x}}, \mathbf{P}_x) \in \mathbb{R}^n$,
▷ A function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$
▷ And a user-determined scaling κ (often set to $\kappa = 3 - n$, see [66])
The random variable $\boldsymbol{x}$ is approximated as a set $2n+1$ of deterministically chosen points and corresponding weights,

$$\begin{aligned} \boldsymbol{\mathcal{X}}^{(0)} &= \hat{\boldsymbol{x}}, & w^{(0)} &= \kappa/(n+\kappa) \\ \boldsymbol{\mathcal{X}}^{(i)} &= \hat{\boldsymbol{x}} + \big(\sqrt{(n+\kappa)\mathbf{P}_x}\big)_i, & w^{(i)} &= 1/2(n+\kappa) \\ \boldsymbol{\mathcal{X}}^{(i+n)} &= \hat{\boldsymbol{x}} - \big(\sqrt{(n+\kappa)\mathbf{P}_x}\big)_i, & w^{(i+n)} &= 1/2(n+\kappa), \end{aligned} \tag{2.14}$$

Where $\sqrt{(n+\kappa)\mathbf{P}_x}\big)_i$ is the i-th row (or column) of the matrix square root, which can be computed using a Cholesky factorization [65]. The superscripts $^{(i)}$ denote point index, not an exponent.
The points are transformed through the function $\boldsymbol{f}$,

$$\boldsymbol{\mathcal{Y}}^{(i)} = \boldsymbol{f}(\boldsymbol{\mathcal{X}}^{(i)}), \quad i = 0, \ldots, 2n \tag{2.15}$$

And the propagated mean and covariance are,

$$\begin{aligned} \hat{\boldsymbol{y}} &\simeq \sum_{i=0}^{2n} w^{(i)} \boldsymbol{\mathcal{Y}}^{(i)}, \quad i = 0, \ldots, 2n \\ \mathbf{P}_y &\simeq \sum_{i=0}^{2n} w^{(i)} (\boldsymbol{\mathcal{Y}}^{(i)} - \hat{\boldsymbol{y}})(\boldsymbol{\mathcal{Y}}^{(i)} - \hat{\boldsymbol{y}})^\top \end{aligned} \tag{2.16}$$

Accuracy of Widely Used Grid-Based Approximations

Grid-based uncertainty propagation methods have varying accuracies. The accuracy is often expressed in terms of orders, indicating what *order of polynomial* $\boldsymbol{g}(\boldsymbol{z})$ they are accurate to. Quadrature rules (such as the Gauss-Hermite quadrature [68]) with m evaluation points and weights are exact for polynomials of order

$o \leq 2m-1$. Consider,

$$E[f(x)] = \int f(x)\mathcal{N}(\boldsymbol{x}; \mathbf{0}_n, \mathbf{I}_{n\times n})d\boldsymbol{x}$$
$$E[(f(x)-E[f(x)])^2] = \int (f(x))^2\mathcal{N}(\boldsymbol{x}; \mathbf{0}_n, \mathbf{I}_{n\times n})d\boldsymbol{x} - E[f(x)]^2$$

Where $f(x) = x^p$. A 3-point Gauss-Hermite quadrature rule can accurately propagate the distribution mean for $p \leq 5$, but the propagated variance will only be accurate for $p \leq 2.5$.

The accuracy of the unscented transform (which the unscented Kalman filter builds upon) is analyzed in [71, Chapter 4.3]. It is there shown that the propagated mean is accurate to the third order, and the covariance to the second order. The magnitude of the errors depend on the variant of the unscented transform, and the scaling parameters used. Further, as noted in [71, Chapter 4.4], the unscented transform for scalar systems can yield identical weights and evaluations points to the 3-point Gauss-Hermite quadrature rule for scalar systems.

2.3.2 Bayesian Uncertainty Estimation

There is a substantial body of literature concerning Bayesian uncertainty estimation [27] [75] [76] [77] [14] [49]. The Gaussian filtering methods in Section 2.3.1 are derived from Bayesian statistics, and many deep learning uncertainty estimation techniques build upon similar methods. The fundamental idea behind Bayesian uncertainty estimation is that *probabilities* are used to represent the uncertainties. In order to make predictions, we perform marginalization over the uncertainties that are present. In order to learn a model, we first *assume* a probability distribution over the parameters (referred to as a prior), $p(\boldsymbol{\theta})$. We then observe data $\mathcal{D}$ and *adjust* the assumption to obtain $p(\boldsymbol{\theta}|\mathcal{D})$ (referred to as the posterior). In order to perform inference with the resulting model, we marginalize out the uncertainty in the conditional parameter distribution to obtain a prediction,

$$p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}) = \int_{\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} \tag{2.17}$$

Bayesian methods have additionally been considered a great deal for *model averaging* [76] [75], including for forecasting [27]. This is also referred to as soft model selection, as it involves combining multiple models (none of which are necessarily correct) to obtain more accurate predictions and predictive uncertainty. Instead of selecting among competing models, Bayesian model averaging employs a mixture of an *ensemble* of models to make predictions. This can both improve predictive accuracy (i.e. more accurate predictive mean) and yield better uncertainty estimates (as spread-error correlation between predictions is common [27]) [78]. Bayesian model averaging requires estimation of the weighting of each

model in the ensemble. Approaches to this include using Bayes factors [76] and the related approximation, the Bayesian information criterion [75]. An arguably simpler approach was proposed in [27], where the *Expectation Maximization* algorithm was applied to estimate the model average weighting.

2.3.3 Uncertainty in Machine Learning Algorithms: An Overview

Several authors have recently categorized uncertainty sources in machine learning - though with focus on deep learning and neural networks. Notable examples include [46], [79] and [80]. One can, very broadly, summarize uncertainties in machine learning as belonging to the following categories,

1. **Model uncertainty**: Uncertainty associated with the choice of model and the parameters of the chosen model. This uncertainty is reducible given sufficient amounts of high quality data. It is expressed as $p(\boldsymbol{\theta}|\mathcal{D})$ in Equation (2.17).
2. **Inherent noise**: An irreducible uncertainty stemming from noise in the data, stochasticity in the process, and other phenomena that cannot be accounted for by increasing the amount of available data. This puts a lower bound on the uncertainty in the predictions from a model, and is expressed as $p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})$ in Equation (2.17).
3. **Propagated uncertainty**: Uncertainty in the inputs to an algorithm will propagate through the algorithm. This propagation of uncertainty is highly relevant for interconnected machine learning algorithms and regression tasks with uncertainty in the input. Uncertainty propagation could be expressed in Equation (2.17) by including a distribution over $\boldsymbol{x}$.
4. **Concept shift and distributional shift**: Differences between the training environment and the operating environment might occur in an instant, or slowly over time. This will result in added uncertainty that can be challenging to quantify. This is also referred to as model misspecification. It can be interpreted as a mismatch between the function $p(\boldsymbol{y}|\boldsymbol{x},\mathcal{D})$ that has been learned, and the true process $p(\boldsymbol{y}|\boldsymbol{x})$.

Works concerning **(1)** and **(2)** include [17], [16], [47], [46], and [27]. **(3)** is thoroughly reviewed in [21], and relevant works include [24], [22], [67], [70], and [73, Appendix A]. A review of **(4)** is given in [81]. **(4)** is outside the scope of this thesis, and will not be covered further.

Predictive uncertainty - the uncertainty stemming from model uncertainty and inherent noise - is visualized in Figure 2.4. Propagated uncertainty is visualized in Figure 2.5.

2.3.4 Model Uncertainty Estimation for Neural Networks

There has, in recent years, been significant research efforts on representing model uncertainty and inherent noise in neural networks. Many methods build directly

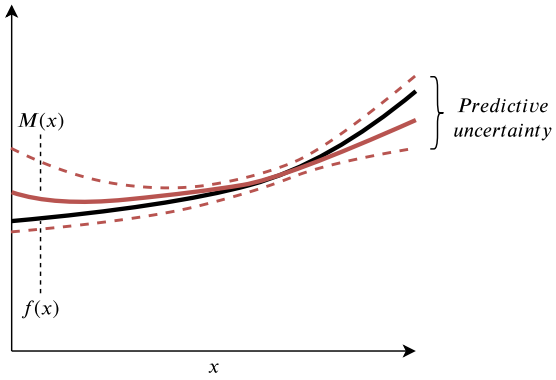


Figure 2.4: Illustration of Predictive Uncertainty: Model uncertainty and inherent noise result in a predictive uncertainty. Even if the model input is deterministic, the model output can have an associated uncertainty. This is illustrated above. The continuous black line indicates the ground truth $f(x)$, and the model predictions $M(x)$ with the associated predictive uncertainty are indicated by the continuous and dashed red lines, respectively. *Note: This figure was also used in my pre project report.*

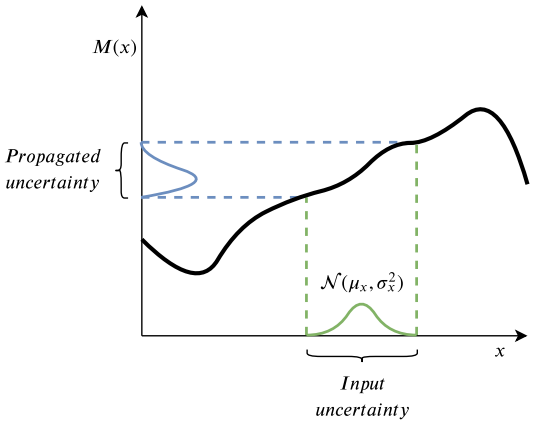
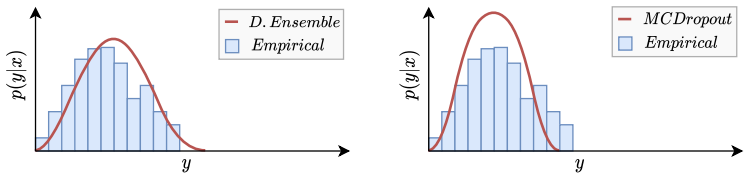


Figure 2.5: Propagated Uncertainty

Figure 2.6: Illustration of Propagated Uncertainty: Uncertainty in the input to to a model can propagate through the model, yielding uncertainty in the output. This is illustrated above, where an uncertain input (along the x-axis) causes an uncertain output (along the y-axis) from the model M. *Note: This figure was also used in my pre project report.*

upon Bayesian statistics, and learn a distribution over the model parameters. However, neural networks can have millions of parameters, making the integral in Equation (2.17) intractable. As such, approximate Bayesian methods are often used. Briefly put, these rewrite Equation (2.17) in terms of a sum over a finite parameter mixture. These use a finite number of realizations $\boldsymbol{\theta}_i$ of the parameter distribution $p(\boldsymbol{\theta}|\mathcal{D})$. The output y is often approximated [16] [17] as a Gaussian using the first two moments of the output mixture distribution,

$$\begin{aligned} y &\sim \sum_{i}^{T} \frac{1}{T} p(y|\boldsymbol{x};\boldsymbol{\theta}_i) \\ &\simeq \mathcal{N}(y;\hat{y},\sigma_y^2) \end{aligned} \tag{2.18}$$



(a) **(b)**

Figure 2.7: A Qualitative Example of Deep Ensemble and MC Dropout Predictions: Deep ensembles **(a)** optimize on the predictive likelihood. We can expect them to predict a distribution with good coverage of the ground truth. Networks using Monte Carlo dropout **(b)** typically optimize on the mean square error in the predictive mean. We can then expect them to predict a distribution with mode close to the expectation of the ground truth.

Variational Bayesian networks such as [82] and [47] represent model parameters as probability distributions parametrized by deterministic values (e.g. a normal distribution given by a mean and a variance), and sample from these when performing inference. This relies on the reparametrization trick [83], which enables optimization on stochastic samples of variational distributions. *Monte Carlo dropout* [16] [84] is among the most widely used approaches for variational Bayesian inference, and represents model parameters as a bimodal distribution. *Deep ensembles* were proposed in [17] as an alternative to Bayesian methods. This approach involves learning a parametrization of a distribution with several independently trained networks. A qualitative illustration of the predictions from *Monte Carlo dropout* and *Deep Ensembles* is given in Figure 2.7

Recent research indicates that methods based on deep ensembles yield better uncertainty estimates than many competing methods in empirical experiments [20] [49]. However, it has been shown that all existing methods suffer significantly when faced with out of distribution samples (e.g. due to model misspecification) [85] [20].

Method	Deep Ensembles [17]	Monte Carlo dropout [16]
Model uncertainty	An ensemble of function representations combined as a finite mixture distribution	Bimodal distribution over the parameters, which is sampled from when performing inference, and combined as a finite mixture distribution
Inherent noise	Learned as a function of inputs	Constant variance given by a hyperparameter

Table 2.2: Deep Ensembles and Monte Carlo Dropout: Overview. These methods for estimating the predictive uncertainty in a neural network both represent the model uncertainty and inherent noise, but do so in different ways.

Deep Ensembles

Deep ensembles [17] are a neural network architecture consisting of multiple identical networks. They rely on random initialization of neural network parameters to obtain several models which reproduce the training data with similar accuracy. The resulting model consists of M neural networks, each of which has parameters corresponding to a local optimum of the training cost function.

Training a deep ensemble requires

1. Deciding on a distribution class to parametrize the output as, and
2. Choosing a proper scoring rule [77] as optimization target.

In [17], the networks output Gaussians $y \sim \mathcal{N}(\mu_{\theta_m}(\boldsymbol{x}), \sigma^2_{\theta_m}(\boldsymbol{x}))$ where m is the network number. Performing inference is straightforward, as the ensemble output is given by a mixture of the M network outputs. [17] further approximates this as a Gaussian given by the first two moments of the mixture distribution, as seen in Equation (2.19). It has been argued that Deep ensembles can be interpreted in terms of a Bayesian model average [49].

$$
\begin{aligned}
p(y|\boldsymbol{x};\theta_{1\ldots M}) &= \frac{1}{M}\sum_{m=1}^{M}\mathcal{N}(\mu_{\theta_m}(\boldsymbol{x}), \sigma^2_{\theta_m}(\boldsymbol{x})) \\
&\simeq \mathcal{N}(\mu_*(\boldsymbol{x}), \sigma^2_*(\boldsymbol{x})), \\
\mu_*(\boldsymbol{x}) &= \frac{1}{M}\sum_{m=1}^{M}\mu_{\theta_m}(\boldsymbol{x}) \\
\sigma^2_* &= \frac{1}{M}\sum_{m=1}^{M}\{\,(\mu_{\theta_m}(\boldsymbol{x}) - \mu_*(\boldsymbol{x}))^2 + \sigma^2_{\theta_m}(\boldsymbol{x})\,\}
\end{aligned}
\tag{2.19}
$$

Monte Carlo Dropout

Monte Carlo dropout [16] is an approximate variational Bayesian inference method for neural networks. The network parameter distribution is given by bimodal Gaussian mixture,

$$q(\boldsymbol{\theta}) = (1-d)\mathcal{N}(\boldsymbol{\theta}, \sigma^2\mathbf{I}) + d\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}) \tag{2.20}$$

Where $0 < d < 1$ is a constant probability. The network employs variational inference to sample from the parameters. This means that it samples from a Bernoulli distribution, and transforms the realizations using the *variational parameters* $\boldsymbol{\theta}$. This can be performed in a single step by applying dropout with probability d before every affine transformation in the network [84], which samples from the mixture modes. The network output has an uncertainty given by a constant variance τ^{-1}, which represents the inherent noise. Extensions which learn the inherent noise have been presented as well [46]. For inference, T forward passes with dropout enabled are performed and the outputs are combined as a mixture. Equation (2.21) shows how we can approximate the integral in Equation (2.17) for a univariate output. This involves sampling realizations $q(\boldsymbol{\theta})_t$ of $q(\boldsymbol{\theta})$ using dropout.

$$\begin{aligned} p(y|\boldsymbol{x}; q(\boldsymbol{\theta})) &= \int_{q(\boldsymbol{\theta})} \mathcal{N}(f_{q(\boldsymbol{\theta})}(\boldsymbol{x}), \tau^{-1}) p(q(\boldsymbol{\theta})) dq(\boldsymbol{\theta}) \\ &\simeq \mathcal{N}(\mu_{mc}(\boldsymbol{x}), \sigma^2_{mc}(\boldsymbol{x})), \\ \mu_{mc}(\boldsymbol{x}) &= \frac{1}{T}\sum_{t=1}^{T} f_{q(\boldsymbol{\theta})_t}(\boldsymbol{x}) \\ \sigma^2_{mc} &= \frac{1}{T}\sum_{m=1}^{T} \{ (f_{q(\boldsymbol{\theta})_t}(\boldsymbol{x}) - \mu_{mc}(\boldsymbol{x}))^2 + \tau^{-1} \} \end{aligned} \tag{2.21}$$

It has been noted [17] that Monte Carlo dropout tends to give uncalibrated uncertainty estimates when compared to alternatives (such as Deep Ensembles). The lack of calibration is also noted in the appendix of the original paper [84] on Monte Carlo dropout.

2.3.5 Input Uncertainty Propagation for Neural Networks

Most uncertainty propagation methods do not impose constraints on the function that they are applied on. As such, the methods considered in Section 2.3.1 can be applied for neural networks.

More generally, we can distinguish between *i) layer-wise* uncertainty propagation, and *ii) entire-network* uncertainty propagation for neural networks. The authors of [24] compared some approaches, and found that Monte Carlo simulation [22] followed by entire-network unscented transform yielded the best results

among them. Assumed density filtering [86] has been applied [48][87] for layer-wise propagation of uncertainty. As it assumed zero correlation between the activation distributions in a layer it is likely not optimal for low-dimensional regression tasks.

Regardless of the method, there is a trade-off between computational complexity and accuracy [21]. Low-order numerical integration methods (e.g. 3-node Gauss-Hermite quadrature [68] or the unscented transform [67]) will yield less accurate results for higher order moments compared to higher-order methods (e.g $m > 3$-node quadrature formulas [21]). Stochastic sampling methods such as Monte Carlo simulation [22] can more accurately capture the effects of nonlinearities and uncertainty in the model - at the cost of increased computational expense.

For neural networks that model a state transition function, the network output will tend towards the identity function as the time step length tends towards zero. For such models, entire-network propagation methods seem like a sensible approach - as the network output might exhibit less nonlinearities than the individual layers do.

2.3.6 Uncertainty Estimation for Dynamical System Models

Monte Carlo dropout has been applied successfully to estimate predictive uncertainty for time series modeled by neural networks. Notable implementations include [18] and [88]. These methods rely on stochastic sampling: re-running the same simulation repeatedly, whilst resampling the model parameters using dropout each time step.

Deep ensembles have also been applied for recurrent neural networks. This is demonstrated in [19], where they employ an ensemble of long short term memory networks. These simulate individually, and the individual predictions at a time step are combined as a mixture. [19] uses the ensemble as a virtual sensor for sideslip angle estimation to guide an unscented Kalman filter - an interesting hybrid model. Approaches that combine neural networks and traditional state space modeling were in fact explored in some early works on the unscented Kalman filter, including [59]. Though not entirely equivalent to [19], as [59] model the process dynamics (not a virtual measurement), it is nonetheless interesting to see that combination approaches have been studied.

ODEnets, as covered in Section 2.2.2, are well suited to modeling dynamical systems. The impact of including stochastic regularization techniques (e.g. dropout, additive noise and multiplicative noise) to ODEnets is explored in [89]. The approach can be informaly be expressed as Equation (2.22), where $\boldsymbol{G}$ depends on the exact regularization technique used.

$$\begin{aligned} \boldsymbol{h}_{k+1} &= \boldsymbol{f}(\boldsymbol{h}_k, t; \boldsymbol{\theta}) + \boldsymbol{G}(\boldsymbol{h}_k, t; \boldsymbol{\theta}_v)v, \\ v &\sim \mathcal{N}(0,1) \end{aligned} \tag{2.22}$$

Their research indicates that such methods can provide significant predictive per-

formance benefits. The authors in [89] do *not* consider uncertainty estimation. However, Monte Carlo dropout is a well established method of estimating model uncertainty. As mentioned, it has additionally been employed to estimate uncertainty in dynamical systems - and for residual architectures [88]. In light of this, the results in [89] indicate that dropout should be well suited for uncertainty estimation with neural difference equations. Aside from this, it is interesting to see that they report additive and multiplicative noise is beneficial for improving generalization. The model structure resulting from incorporating the noise is, in effect, nearly indistinguishable from the state transition structure employed in Gaussian filtering. Perhaps equally interesting: It is similar to the *output* one can obtain from predictive uncertainty estimation for neural networks, as can be seen in Equation (2.18).

2.3.7 Evaluating Probabilistic Predictions

Calibration and sharpness are two very important characteristics of probabilistic predictions. *Calibration* is a measure of how well the predicted distributions correspond to the empirical observations [90] [77]. This means that if a calibrated model predicts some 95% confidence interval, empirical observations should lie within that interval 95% of the time. *Sharpness* refers to the density of the individual prediction distributions [77]. As such, a sharper prediction will yield narrower confidence intervals. A very accurate model that is calibrated produces sharp predictions.

Useful metrics for evaluating probabilistic predictions are presented in [77], [90], and [91] [92]. The *expected calibration error* (ECE) and *reliability diagrams* [90] are two closely related metrics which measure calibration.

Expected Calibration Error and Reliability diagrams

Expected calibration error [90] measures the average discrepancy between the predicted and empirical coverage of a confidence interval.

The approach in [90] can be adapted for regression models that predict continuous output distributions. The expected calibration error can then be expressed as,

$$\begin{aligned} \mathrm{ECE} &= \sum_{i=0}^{K-1} \frac{1}{K} |c_i - \alpha_i| \\ \alpha_i &\triangleq i/(K-1) \end{aligned} \tag{2.23}$$

Where K confidence intervals with increasing coverage are evaluated. α_i is the predicted coverage of confidence interval of confidence interval i. The fraction of empirical observations that fall within the confidence interval is given by c_i. For a perfectly calibrated model, we expect that exactly $\alpha_i \cdot 100\%$ of the empirical observations fall into a α_i confidence interval. If this is the case for all the confidence intervals evaluated, we obtain an expected calibration error of 0%. Conversely, a

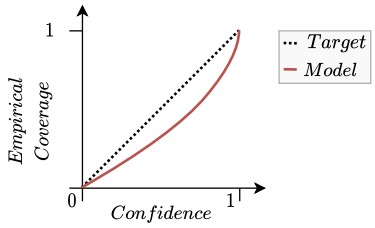


Figure 2.8: A Reliability Diagram: A reliability diagram shows the calibration curve of a model: A comparison of the predicted coverage (confidence) and the empirical coverage. This illustration shows an overconfident model, meaning that the confidence intervals have lower empirical coverage than predicted. A perfectly calibrated model will have a confidence that consistently matches the empirical coverage (indicated by the dashed line). The expected calibration error in Equation (2.23) is the integral of the absolute vertical difference between the dashed diagonal line and the model's calibration curve for confidences between 0 and 1. This is often converted to a percentage.

model that is entirely uncalibrated will yield an expected calibration error of 50%.

This method can visualized well using reliability diagrams [90], which plot α_i against c_i. This is illustrated in Figure 2.8.

2.4 Literature Study Conclusion

There have been significant developments for neural networks within the last decade. This has sparked an interest in uncertainty estimation for neural networks, as their predictive capabilities can offer utility for industrial applications. Approaches to estimate uncertainty in dynamical systems often rely on deep networks or recurrent structures.

Some neural network approaches - such as ODEnets - are drawing inspiration from traditional system representations and physical modeling. Such models are interpretable in terms of traditional state space representations, which lends to them being applied in conjunction with state estimation techniques. The effect of noise injection in ODEnets has been explored previously. This results in a system structure with the same components as the traditional *process dynamics - process noise* models employed in Kalman filters. Further, it closely resembles the output of existing predictive uncertainty estimation methods.

The reviewed research hints at an enticing possibility: Representing a process state transfer function *and* the process noise of a system using a neural network with predictive uncertainty estimation. This would be possible to employ in a state space representation. Further, the use of neural networks in filtering algorithms has been studied previously. Employing a grid-based uncertainty propagation algorithm for propagating a state belief through the neural network would be quite

different from most existing approaches, which rely on stochastic sampling. However, this would have some benefits:

1. The model would consist of difference equations, which are often highly interpretable
2. The model could be used for *non-stochastic* uncertainty estimation, e.g. with Deep Ensembles. From a reliability perspective, this could be desirable
3. The model would be applicable in a filtering problem (which may not be as trivial with e.g. a recurrent neural network)

The research questions for the thesis tie directly into the observations from the literature study - and the research questions will be answered in Sections 3.1 and 3.2.

Chapter 3

Method

This chapter develops a simple approach to combine variable predictive uncertainty with input uncertainty propagation. Following this, an approach to modeling dynamical systems using neural networks with predictive uncertainty estimation is outlined.

3.1 Grid-Based Uncertainty Propagation Through Models with Uncertainty Estimates

Grid-based uncertainty propagation methods such as the *unscented transform* and *sparse grid quadrature* are numerical approximations to solving weighted integrals.

Propagating uncertainty through a model with a variable (*heteroscedastic*) output uncertainty requires marginalization over the model's uncertainty, in addition to marginalization over the input uncertainty. This can be expressed as an iterated integral. These integrals can be approximated using existing grid-based uncertainty propagation methods. This means that *model uncertainty estimation* and *grid-based input uncertainty propagation* can be seamlessly combined.

The approach presented here has not been identified in existing literature. However, as it is only a small extension of existing methods, it is not unlikely that it has been applied previously by another author.

This section proceeds as follows: *i)* we first derive equations for applying grid-based uncertainty propagation to models with variable output uncertainties *ii)* relate the approach to existing methods, *iii)* show how the approach can be applied in a Gaussian filtering framework, and lastly, *iv)* briefly cover necessary assumptions, limitations, and possible extensions.

3.1.1 Propagating Uncertainty Through Models With Variable Uncertainty

Consider the Bayesian integral for uncertainty propagation in a dynamical system,

$$p(\boldsymbol{x}_k) = \int_{\mathbb{R}^n} p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})p(\boldsymbol{x}_{k-1})d\boldsymbol{x}_{k-1} \tag{3.1}$$

Using the fundamental theorem of calculus and Fubini's theorem we can express the resulting first and second raw moments as follows,

$$\begin{aligned} E[\boldsymbol{x}_k] &= \int_{\mathbb{R}^n} \boldsymbol{x}_k p(\boldsymbol{x}_k) d\boldsymbol{x}_k \\ &= \int_{\mathbb{R}^n} \int_{\mathbb{R}^n} \boldsymbol{x}_k p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) d\boldsymbol{x}_k p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1} \\ &= \int_{\mathbb{R}^n} E[\boldsymbol{x}_k|\boldsymbol{x}_{k-1}] p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1} \end{aligned} \tag{3.2}$$

$$\begin{aligned} E[\boldsymbol{x}_k \boldsymbol{x}_k^\top] &= \int_{\mathbb{R}^n} \boldsymbol{x}_k \boldsymbol{x}_k^\top p(\boldsymbol{x}_k) d\boldsymbol{x}_k \\ &= \int_{\mathbb{R}^n} \int_{\mathbb{R}^n} \boldsymbol{x}_k \boldsymbol{x}_k^\top p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) d\boldsymbol{x}_k p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1} \\ &= \int_{\mathbb{R}^n} E[\boldsymbol{x}_k \boldsymbol{x}_k^\top|\boldsymbol{x}_{k-1}] p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1} \end{aligned} \tag{3.3}$$

Where the notation $E[\boldsymbol{x}_k|\boldsymbol{x}_{k-1}]$ indicates the conditional expectation of $\boldsymbol{x}_k$ given that $\boldsymbol{x}_{k-1}$ is known.

We remark that we can express the first n raw moments of the output distribution in terms of the first n raw moments of the model's (*conditional*) output distribution $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$. As a consequence, we can express the n first moments of the output distribution without imposing any assumptions on the distribution $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ aside from being able to compute these n moments. This means that we can approximate Equations (3.2) and (3.3) using a grid-based uncertainty propagation method (introduced in Section 2.3.1),

$$E[\boldsymbol{x}_k] \simeq \sum_i w_{k-1}^{(i)} E[\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^{(i)}] \tag{3.4}$$

$$E[\boldsymbol{x}_k \boldsymbol{x}_k^\top] \simeq \sum_i w_{k-1}^{(i)} E[\boldsymbol{x}_k \boldsymbol{x}_k^\top|\boldsymbol{x}_{k-1}^{(i)}] \tag{3.5}$$

Equations (3.4) and (3.5) describe the most important result from this section. It is the result of a short derivation, but it describes how we can propagate an

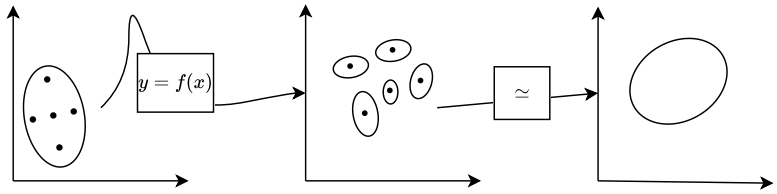


Figure 3.1: Grid-based Uncertainty Propagation with Variable Uncertainty: A continuous input distribution is approximated using deterministically chosen point coordinates and weights (*left*). The points are transformed, yielding a set of probability distributions (*center*), and the statistics of the propagated continuous distribution are approximated from the statistics of the mixture of these distributions (*right*). Compared to Figure 2.3, the difference is the variable uncertainties associated with the propagated points.

input uncertainty through an uncertain model. The output distribution can be expressed as a mixture of the distributions predicted at the input grid evaluation points. This can be extended to estimate higher order moments, but that will not be considered in this thesis.

3.1.2 Relations to Existing Methods

The presented grid-based propagation approach describes how we can *directly* apply existing methods for propagating uncertainty through a model which has an uncertain output. In practice, it decouples uncertainty in the input $\boldsymbol{x}_{k-1}$ from uncertainty in the output $\boldsymbol{x}_{k|\boldsymbol{x}_{k-1}}$. This makes it simple to express conditional uncertainties - which arise in functions with variable uncertainties such as machine learning models with uncertainty estimates.

Conceptually, it is very similar to the approaches employed in existing literature on filtering, including [68], [66], and [69]. The main difference is the slight restructuring of the computations. Since the uncertainty in the system state transition is expressed by the model (and not some pre defined, or time varying covariance), the process noise covariance cannot be moved outside of the summation in Equation (3.5) - which is what is commonly done. This means that only the *reconstruction* of the state uncertainty following the model evaluation needs to change compared to [68], [66], and [69]. In practice, existing methods are applied to perform the propagation of the input uncertainty - the presented approach only describes how we can use these methods *and* additionally account for variable uncertainty in the state transition.

Aside from that, a similar method is required for Gaussian process state space models: They require propagation of Gaussian uncertainty through a Gaussian process (which has variable state transition uncertainty). The authors of [93] suggest moment matching or linearization for this purpose. The extended version [94] of their paper outlines how these approaches can be applied for Gaussian

processes. One of these is based on moment matching. Their starting point is the same as is considered here (Equation (3.2), Equation (3.3)), but as they know the functions exactly they can compute the moments analytically. Consequently, their approach appears an ideal choice in cases where Gaussian processes with Gaussian inputs are considered.

3.1.3 Gaussian Filtering

We can apply grid-based propagation methods to propagate Gaussian state beliefs through a state transformation with variable uncertainty. Consider the discrete dynamic state space model given by Equations (3.6) to (3.9). As shown in Equations (3.8) and (3.9), the process noise is given by variable Gaussian uncertainty, but the measurement noise is well-specified by some known sensor characteristic. Note that it is here assumed that the measurement function (relating the system state $\boldsymbol{x}$ to the output) is given by a matrix multiplication, similarly to in [64, Chapter 5.5].

$$\begin{aligned} \boldsymbol{x}_k &= \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}) + \boldsymbol{w}_k \quad \in \mathbb{R}^n && (3.6)\\ \boldsymbol{y}_k &= \mathbf{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k \quad \in \mathbb{R}^m && (3.7)\\ \boldsymbol{w}_k &\sim \mathcal{N}(\mathbf{0}_n, \mathbf{Q}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1})) && (3.8)\\ \boldsymbol{v}_k &\sim \mathcal{N}(\mathbf{0}_m, \mathbf{R}) && (3.9) \end{aligned}$$

The propagation of uncertainty will in this thesis be approximated using the unscented transform. Other grid-based methods could also be used.

Substituting Equation (3.6) into Equations (3.4) and (3.5) we note that it is possible to express the uncertainty propagation through the state transition as,

$$E[\boldsymbol{x}_k] \simeq \sum_i w_{k-1}^{(i)} \boldsymbol{f}(\boldsymbol{x}_{k-1}^{(i)}, \boldsymbol{u}_{k-1}) \tag{3.10}$$

$$E[\boldsymbol{x}_k \boldsymbol{x}_k^\top] \simeq \sum_i w_{k-1}^{(i)} \Big(\boldsymbol{f}(\boldsymbol{x}_{k-1}^{(i)}, \boldsymbol{u}_{k-1}) \boldsymbol{f}(\boldsymbol{x}_{k-1}^{(i)}, \boldsymbol{u}_{k-1})^\top + \mathbf{Q}(\boldsymbol{x}_{k-1}^{(i)}, \boldsymbol{u}_{k-1}) \Big) \tag{3.11}$$

Which means that for a system with a non-variable process noise $\mathbf{Q}$ we recover the same equations as in conventional Gaussian grid-based filtering (see [68]).

We can modify the Unscented Kalman Filtering algorithm [71, Algorithm 8] to enable simulation and filtering using a model with a variable output uncertainty. This is outlined in Algorithm 2, with the sigma point set from Algorithm 1. The process state transition and the process noise in Algorithm 2 are expressed using

the functions $\boldsymbol{f}$ and $\mathbf{Q}$ in Equations (3.6) and (3.8).

Algorithm 2: Unscented Kalman Filter with Variable Model Uncertainty

---

Initialize

$$\hat{\boldsymbol{x}}_0 = E[\boldsymbol{x}_0], \quad \mathbf{P}_0 = E[(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0)(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0)^\top]$$
$$N = 2n+1, \text{ where } n = \dim \boldsymbol{x}, \quad \kappa \geq 0$$

for $k = 1 \ldots \infty$

1. **Compute sigma points**: According to Algorithm 1

$$\boldsymbol{\mathcal{X}}_{k-1} = \underbrace{\left[\hat{\boldsymbol{x}}_{k-1} \quad \hat{\boldsymbol{x}}_{k-1} \pm \sqrt{(n+\kappa)\mathbf{P}_{k-1}}\right]^\top}_{[N,\ n]} \tag{3.12}$$

2.1. **Time update**: Evaluate process dynamics and process noise

$$\boldsymbol{\mathcal{X}}_{k|k-1}, \quad \boldsymbol{\mathcal{Q}}_k = \underbrace{\boldsymbol{f}(\boldsymbol{\mathcal{X}}_{k-1}, \boldsymbol{u}_{k-1})}_{[N,\ n]}, \quad \underbrace{\mathbf{Q}(\boldsymbol{\mathcal{X}}_{k-1}, \boldsymbol{u}_{k-1})}_{[N,\ n,\ n]} \tag{3.13}$$

$$\hat{\boldsymbol{x}}_{k|k-1} = \sum_{i=0}^{N-1} w^{(i)} \boldsymbol{\mathcal{X}}^{(i)}_{k|k-1} \tag{3.14}$$

$$\mathbf{P}_{k|k-1} = \sum_{i=0}^{N-1} w^{(i)} \{(\boldsymbol{\mathcal{X}}^{(i)}_{k|k-1} - \hat{\boldsymbol{x}}_{k|k-1})(\boldsymbol{\mathcal{X}}^{(i)}_{k|k-1} - \hat{\boldsymbol{x}}_{k|k-1})^\top + \boldsymbol{\mathcal{Q}}^{(i)}_k\} \tag{3.15}$$

2.2. **Time update**: Predict next output

$$\hat{\boldsymbol{y}}_{k|k-1} = \mathbf{H}_k \hat{\boldsymbol{x}}_{k|k-1} \tag{3.16}$$
$$\mathbf{P}_{yy,k} = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \tag{3.17}$$
$$\mathbf{P}_{x,k} = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \tag{3.18}$$

3. **Measurement update**:

$$\mathbf{K}_k = \mathbf{P}_{xy,k} (\mathbf{P}_{yy,k} + \mathbf{R}_k)^{-1} \tag{3.19}$$
$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k|k-1} + \mathbf{K}_k (\boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1}) \tag{3.20}$$
$$\mathbf{P}_k = \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{P}_{xy,k}^\top \tag{3.21}$$

end **for**

3.1.4 Accuracy, Limitations, Necessary Assumptions, and Extensions

Accuracy

A grid-based uncertainty propagation method will be more accurate than a Taylor expansion-based method. This is thoroughly covered by [71], and also outlined in [67].

Intuitively, one might expect that a variable process uncertainty would incur a loss in accuracy for the propagation of uncertainty. However, this will not necessarily be the case. Considering Equation (3.11), we see that the propagated raw second moment is expressed as a summation over

$$\boldsymbol{f}(\boldsymbol{x}_{k-1}^{(i)}, \boldsymbol{u}_{k-1})\boldsymbol{f}(\boldsymbol{x}_{k-1}^{(i)}, \boldsymbol{u}_{k-1})^\top + \mathbf{Q}(\boldsymbol{x}_{k-1}^{(i)}, \boldsymbol{u}_{k-1}) \tag{3.22}$$

Remark that the state transition is squared (outer product), whereas the process noise covariance is not.

Assume the entries in $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ can all be expressed as a p-th order polynomial of $\boldsymbol{x}$. Then as long as all the entries of $\mathbf{Q}(\boldsymbol{x}, \boldsymbol{u})$ can be expressed as a p_q-th order polynomial of $\boldsymbol{x}$ with $p_q \leq 2p$, we can expect a similarly accurate result as we would get if $\mathbf{Q}$ was constant. More importantly: For many common uncertainty estimation methods for neural networks (Monte Carlo dropout and Deep Ensembles for instance), the term $\mathbf{Q}(\boldsymbol{x}, \boldsymbol{u})$ can reasonably be expected to be exactly twice the order of $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ (in terms of polynomial complexity). This can be seen e.g. in Equation (2.19), where $\boldsymbol{f}$ would be given by μ_* and $\mathbf{Q}$ would be given by σ_*^2.

These observations indicate that *if* a variable process uncertainty were to cause inaccuracies, it would likely be due to inaccuracies in the predicted process uncertainty - not the propagation of uncertainty.

More generally, the accuracy of the result will be dependent on the grid-based propagation method and the complexity of the transformation. This is outlined in Section 2.3.1. For non-variable uncertainty, the polynomial complexity of the propagation will generally grow exponentially with the number of moments that are propagated: Requiring accurate integration over an $\mathcal{O}(p^k)$ polynomial for propagating k moments through a polynomial of order p.

Computational Cost

The two operations which are likely to incur the highest computational cost for this approach, are *1)* evaluating the model, and *2)* using a grid-based approximation for the input distributions.

The unscented transform in Algorithm 1 is used in this thesis. For a state space model of dimension n, it requires $2n + 1$ model evaluations to propagate uncertainty for a single time step, and a single Cholesky decomposition. This will therefore be the case for Algorithm 2. This also means that the complexity can be

expected to be similar to as in existing variants of the unscented Kalman filter: $\mathcal{O}(n^3)$ complexity [71].

The caveat of the above statement is that evaluation of the machine learning model can incur a significant computational cost. As $2n+1$ model evaluations are performed each time step, the complexity of Algorithm 2 in terms of the model complexity $\mathcal{O}(M)$ is $\mathcal{O}(nM)$. As such we can expect $\mathcal{O}(nM+n^3)$ complexity. However, as will be seen in Chapter 5, the networks that are used for the main experiments are very small. It should further be noted that many optimization libraries (e.g. `PyTorch`) enable high degrees of parallelism. Although $2n+1$ model evaluations are required each time step, these can be expressed as a single batch of inputs to the model.

Assumptions Required for the Grid-Based Propagation Methods

Certain grid-based propagation methods (notably, some variants of the unscented transform) can have negative weights. A grid-based method with negative weights should not be used when the uncertainty in the model is variable. The reason for this, is that negative weights will yield negative terms in the summations of Equation (3.11). That can in turn result in negative definite covariance matrices. This can also be explained by considering that the propagation can be expressed as a mixture of the model output distributions. The weighting of the mixture is given by the weights of the input distribution grid approximation, and they must as such be positive semidefinite.

As one may note, the unscented transform variant in Algorithm 1 can have negative weights depending on the choice of κ. This means that κ should be chosen such that all weights are ≥ 0 if Algorithm 1 is used. This is also expressed in Algorithm 2.

Downsides of Grid-Based Propagation Methods

Grid-based propagation methods generally rely on the assumption of a specific distribution class. Deviations from these assumptions will incur loss in accuracy. Some variants, such as the unscented transform, do not strictly impose this assumption [23]. However, the same limitations will in practice be in place, and the common grid-based propagation methods won't yield accurate results for multimodal distributions or distributions with significant skew. This is a notable downside of the approach presented. A challenge associated with this comes in the form of nonlinear systems that are simulated for long time horizons: We cannot expect them to maintain a Gaussian state distribution. For multivariate state spaces, this can be a significant challenge (which will be briefly explored in Section 4.1). However, for dissipative univariate systems (as will primarily be considered in Chapter 5), the assumption of a Gaussian state distribution over time will not be as unreasonable.

Numerical accuracy and positive semidefinite covariances can be problematic when applying grid-based uncertainty propagation. When first extending Algorithm 2 to multivariate systems, several challenges associated with numerical accuracy were encountered. These were caused by quantization errors, and were resolved by changing to 64-bit floating point numbers instead of 32-bit. Further, a somewhat impractical issue is that some (at least `PyTorch`'s) Cholesky decomposition implementations may raise exceptions when the state covariance is zero-valued. This makes sense from a mathematical point of view, but can be impractical - e.g. if initializating with zero state uncertainty. For the implementation of Algorithm 2, this was solved by wrapping the Cholesky decomposition with some logic to exclude zero-valued rows and columns from the decomposition computation. This is of no consequence for Chapter 5 (as the neural network models used there ended up being univariate), but can be impractical for multivariate systems.

Nonlinear Measurement Functions

The presented approach can be extended to systems with a nonlinear measurement function $\boldsymbol{h}$. This is covered more in-depth in Appendix A.

Accurate estimation of the output distribution of $\hat{\boldsymbol{y}}$ after a nonlinear $\boldsymbol{h}$ can be achieved by *discretizing* the conditional output distributions from the model (the predicted means $\boldsymbol{\mathcal{X}}_{k|k-1}^{(i)}$ and associated covariances $\boldsymbol{\mathcal{Q}}_k^{(i)}$) before transforming them through $\boldsymbol{h}$. This preserves information on higher order moments in the distributions of the propagated sigma points, which can impact the output mean and covariance if $\boldsymbol{h}$ is not affine. In practice, this can be achieved by applying the unscented transform on the outputs from the models at the evaluation points. This is very similar to what is done in the unscented Kalman filter in [71, Algorithm 8], there referred to as *augmentation* of sigma points.

The approach presented in Appendix A is more general than what is covered in this section, but also more computationally expensive. Algorithm 6 in Appendix A can be used as a substitute for Algorithm 2 also for affine measurement functions. In fact, the method described in Appendix A was the first approach that was derived, and this section is a simpler variant of that. As the experiments in the thesis don't require nonlinear measurement functions, the method described in Appendix A ended up not being necessary. However, as it can be useful for filtering with an uncertain model and a nonlinear measurement function (e.g. another neural network), it was decided to keep it as an appendix.

3.1.5 Implementation of the method

`PyTorch` [2] was used to implement a filtering algorithm based on the approach presented here. It employs the unscented transform from Algorithm 1 to propag-

ate uncertainties. As all of the computations are performed using `PyTorch`, they are differentiable using `autograd`, which is `PyTorch`'s automatic differentiation package. The primary benefit of this is that optimization on the simulation trajectories (including the state estimate covariances) becomes possible. The `FilterPy` [4], library was used as a reference for initial testing (without variable output uncertainty).

For the sake of transparency, it should be noted that the implementation is actually based on Algorithm 6 in Appendix A, and not Algorithm 2. However, the resulting computations end up being the same.

3.2 Neural Networks as Stochastic Difference Vector Fields

This section presents an approach to modeling dynamic systems and their uncertainties using stochastic differential equations represented by neural networks. It is similar to what is explored in [89], though the approach presented here is with the intent of uncertainty estimation.

We proceed with *i)* presenting the state-space representation the method is based on, *ii)* show how this can be formulated using a model with variable output uncertainty, *iii)* demonstrate how neural networks with uncertainty estimation can be used to model this, and *iv)* how such models can be simulated in time. Lastly, *v)* we present optimization procedures and *vi)* relate the approach to existing methods.

3.2.1 The Foundation: State Space Models with Process Noise

System dynamics in discrete-time state space models are commonly expressed using ordinary difference equations. These are often accompanied by an additive process noise: a Gaussian diffusion term that represents the uncertainty in the dynamics,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{3.23}$$

Where $\boldsymbol{w}_k \sim \mathcal{N}(\mathbf{0}_n, \mathbf{Q}_k)$ for $\boldsymbol{x} \in \mathbb{R}^n$.

A challenge associated with such a formulation is that the process noise can be expected to exhibit heteroscedasticity: It will likely be correlated with (some of) the inputs or states in the system.

3.2.2 Process Uncertainty as a Model-Specified Diffusion Term

Instead of using a constant (or possibly time varying) covariance as process noise, it is proposed to employ a neural network with uncertainty estimation. The neural network is used to jointly model the system dynamics $\boldsymbol{f}$ and the process noise $\mathbf{Q}$. This can, similarly to in [89] be expressed as a difference equation with a diffusion

term. As the neural network models considered in Chapter 5 ended up having one-dimensional state spaces ($x \in \mathbb{R}^1$), the notation used here will reflect that. The proposed system model is on the form of Equations (3.24) to (3.27).

$$x_{k+1} = x_k + f_\theta(x_k, \boldsymbol{u}_k) + w_{\theta,k}, \tag{3.24}$$
$$y_k = x_k + v_k, \tag{3.25}$$
$$w_{\theta,k} \sim \mathcal{N}(0, Q_\theta(x_k, \boldsymbol{u}_k)) \tag{3.26}$$
$$v_k \sim \mathcal{N}(0, R_k) \tag{3.27}$$

Where the process dynamics f_θ are given by the *predictive mean* of a neural network with uncertainty estimation, and the process noise $w_{\theta,k} \sim \mathcal{N}(0, Q_\theta(x_k, \boldsymbol{u}_k))$ has variance given by the *predictive variance* of the same neural network.

3.2.3 Representing Uncertain Dynamics With Neural Networks

Existing approaches to predictive uncertainty estimation can be applied directly to model Equations (3.24) and (3.26). The only practical requirement is that the method must be capable of estimating a predictive mean and predictive (co)variance. *Monte Carlo dropout* and *Deep Ensembles* are two widely used approaches that can be applied for this purpose. Inference algorithms for both of these methods are outlined in Section 2.3.4. Figures 3.2 and 3.3 illustrate how we can model Equations (3.24) and (3.26) system using these two methods.

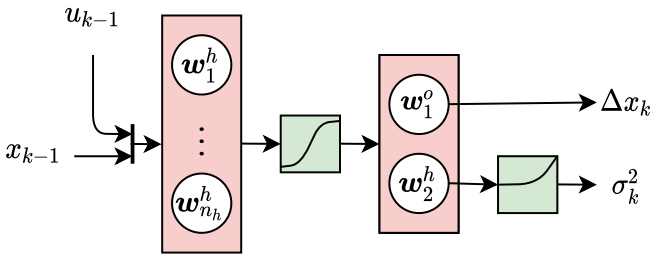


Figure 3.2: Deep Ensemble - Difference Equation Architecture: Shown for a first order difference equation. The individual networks predict a step Δx and an uncertainty given by a variance σ^2. The *ensemble* consists of several of these networks, as described in Section 2.3.4, and the output distributions of the networks in the ensemble are combined as a mixture. The network neurons are given by the white nodes (showing the weights only, bias omitted for clarity). The hidden layer outputs are passed through an activation function (indicated by the centre square block) individually. The predicted variance is passed through a *rectifier* to ensure it is positive (indicated by the rightmost square block).

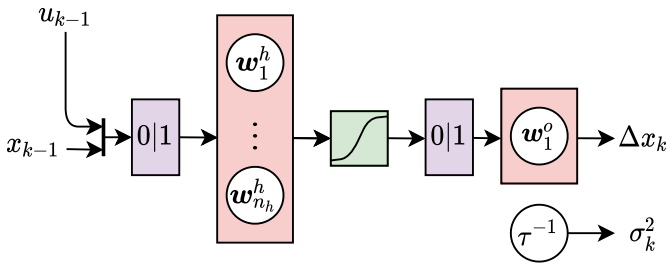


Figure 3.3: MC Dropout - Difference Equation Architecture: Shown for a first order difference equation. A forward pass through the network predicts a step Δx, whose uncertainty is given by a constant variance τ^{-1}. As described in Section 2.3.4, the parameters of the Monte Carlo dropout networks are given by a bimodal distribution. The network neurons are given by the white nodes. The hidden layer outputs are passed through an activation function (indicated by the centre square block) individually. For performing inference, multiple forward passes are done, and the output is given by the mixture of the output distributions from these. Each forward pass, *dropout* is applied before each weight layer. This *randomly zeros out* vector elements (indicated by 1|0 in the illustration), which *randomly samples* from the bimodal weights and biases [84].

3.2.4 Simulating the System Model

A model on the form of Equations (3.24) to (3.27) can be simulated using the grid-based uncertainty propagation method presented in Section 3.1. More precisely, the *time update* of Algorithm 2 can be applied directly for simulating a Gaussian approximation of the system state over time. Further, for filtering, Algorithm 2 in its entirety can be applied.

The simulation of the neural network models can be expressed concisely by adding a layer of abstraction over Algorithm 2. This is shown in Algorithm 3.

Algorithm 3: Approximate Simulation of Stochastic Vector Fields

$\hat{x}_0 \leftarrow$ Initialization
$P_0 \leftarrow$ Initialization
for $k = 1, \ldots, \infty$ **do**

$\boldsymbol{\mathcal{X}}_{k-1} \leftarrow SigmaPoints(\hat{x}_{k-1}, P_{k-1})$
$\boldsymbol{u}_{k-1} \leftarrow GetInput()$
`// Same input for all the sigma points`
$\boldsymbol{\mathcal{U}}_{k-1} \leftarrow Stack(\boldsymbol{u}_{k-1}, 2n+1)$
`// Assuming not filtering, then` $\hat{x}_k = \hat{x}_{k|k-1}, P_k = P_{k|k-1}$
$\hat{x}_k, P_k, \hat{y}_k, P_{yy,k} \leftarrow TimeUpdate(\boldsymbol{\mathcal{X}}_{k-1}, \boldsymbol{\mathcal{U}}_{k-1})$

3.2.5 Parameter Identification

Backpropagation is used to optimize the model parameters. As the filter used to simulate is implemented in `PyTorch`, the propagation of uncertainty through the neural difference equations is differentiable using `PyTorch`'s automatic differentiation [2]. This makes it possible to optimize on the state estimates that result from simulation, using backpropagation.

The parameter optimization differs somewhat between the Monte Carlo dropout-based neural network and the Deep Ensemble-based neural network. The explanation for this can be seen in Figures 3.2 and 3.3: The Monte Carlo dropout implementation *doesn't learn a variance*. The predictive uncertainty of the Monte Carlo dropout algorithm used in this thesis is given by the model parameter uncertainty, and a constant variance: the inverse of its precision τ. The precision is a hyperparameter that in practice represents the inherent noise (see Section 2.3.3). On the contrary, the networks in the Deep Ensemble learns this inherent noise as a function of the inputs. That necessitates slightly differing training methodologies.

Optimization Procedure for the Deep Ensemble network

The output estimates ($\hat{y}$) in Algorithm 2 are given in terms of Gaussians. We can optimize directly on the likelihood of these Gaussian state predictions conditional to the initial state x_0 and the model parameters $\boldsymbol{\theta}$. However, it is more convenient to express the optimization using the negative log likelihood (NLL). This enables rewriting the optimization as a summation, which is minimized. Minimizing the NLL is equivalent to maximizing the likelihood. Letting the past inputs be given by $\boldsymbol{p}_{0\dots k} = [\boldsymbol{u}_0, \boldsymbol{u}_1, \dots, \boldsymbol{u}_{k-1}]^\top$, we can express the negative log likelihood of the simulation over the whole dataset as,

$$\ell(y_1, y_2, y_3, \dots, y_{N_D} | x_0, \boldsymbol{p}_{0\dots k}; \boldsymbol{\theta}) = \sum_{k=1}^{N_D} \ell(T_{B,k} | x_0, \boldsymbol{p}_{0\dots k}; \boldsymbol{\theta}) \tag{3.28}$$

It is convenient to decouple the simulation length from the loss function magnitude, as this will make the gradients invariant of the simulation length when training. This doesn't change the optimum, and can be achieved by normalizing the NLL by the number of simulation steps.

$$\ell(y_1, y_2, y_3, \dots, y_{N_D} | x_0, \boldsymbol{p}_{0\dots k}; \boldsymbol{\theta}) \propto \frac{1}{N_D} \sum_{k=1}^{N_D} \ell(y_k | x_0, \boldsymbol{p}_{0\dots k}; \boldsymbol{\theta}) \tag{3.29}$$

The right hand side of Equation (3.29) is approximated by a minibatch of B simulations of length N_{sim},

$$\ell(\boldsymbol{y}; \boldsymbol{\theta}) \simeq \frac{1}{BN_{sim}} \sum_{i=1}^{B} \sum_{k=i}^{i+N_{sim}} \ell(y_k | x_{0_i}, \boldsymbol{p}_{0_i \dots k}; \boldsymbol{\theta}) \tag{3.30}$$

Where each of the negative log likelihood values on the right hand side of Equation (3.30) are given by Equation (3.31). This can be derived by taking the logarithm of the Gaussian probability distribution function in Equation (2.4), and dropping the constant term.

$$\ell(y_k|x_{0_i}, \boldsymbol{p}_{0_i\ldots k}; \boldsymbol{\theta}) = \frac{1}{2}\ln(P_{y,k}) + \frac{1}{2P_{y,k}}(y_k - \hat{y}_k)^2 \tag{3.31}$$

y_k and $P_{y,k}$ in Equation (3.31) are the output mean and variance predicted by the model at that specific time step, given the initial conditions.

In practice, this means that the system is repeatedly initialized, and then simulated for a time horizon using Algorithm 3 (in dead-reckoning). The mean NLL of the true measurements for the sequence is then computed based on the predicted distributions. The gradient of the NLL with respect to the parameter vector $\boldsymbol{\theta}$ is computed using backpropagation. An optimization algorithm can then update the parameters with the goal of minimizing the NLL.

Similarly to in [17], the networks in the ensemble are trained individually. Contrary to [17] adversarial examples [95] are not used as it was not found to yield improvements.

A summary of this method is given in Algorithm 4. Initialization can be performed by setting an uncertain state (high state (co)variance) and filtering for a few steps. The computational graph can then be detached (meaning that the resulting state is treated as a constant), before beginning the simulation which is optimized on. In Algorithm 4 this is simply denoted as $x_0 \leftarrow y_0$. For the experiment in Chapter 5 where the networks have one-dimensional state spaces, this 'fuzzy' initialization is not a necessity.

Algorithm 4: Stochastic Neural Difference Equation Optimization: Deep Ensemble

Initialize the parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots \boldsymbol{\theta}_M$ of the networks randomly
while *training* **do**
  for $m = 1 \ldots M$ **do**
    $\mathcal{L} \leftarrow 0$
    for $i = 1, \ldots, B$ **do**
      Sample sequence $(\boldsymbol{y}_{0\ldots N_{sim}}, \boldsymbol{u}_{0\ldots N_{sim}})$
      Initialize $x_0 \leftarrow y_0$
      Simulate $\boldsymbol{\theta}_m$ with Algorithm 3 to obtain $\hat{\boldsymbol{y}}_{1\ldots N_{sim}}, \boldsymbol{P}_{y,1\ldots N_{sim}}$
      $\mathcal{L} = \mathcal{L} + \sum_{k=1}^{N_{sim}} \ell(y_k|x_0, \boldsymbol{p}_{0\ldots k}; \boldsymbol{\theta}_m)/(N_{sim}B)$
    Minimize $\mathcal{L}$ wrt. $\boldsymbol{\theta}_m$

Optimization Procedure for the Monte Carlo dropout network

As the Monte Carlo dropout network doesn't learn the inherent noise, it is simulated without uncertainty propagation when training. This means the optimization

procedure resembles what is done in existing literature that applies Monte Carlo dropout for dynamic system modeling [18].

Similarly to for the Deep Ensemble model, optimization is performed by simulating a minibatch of sequences, computing a loss, and updating the parameters. During these simulations, dropout is active - which means that for each prediction step, the model parameters are randomly sampled. An L_2 (Mean square error) loss criterion is used. As specified in [16], *maximum a posteriori* optimization is performed, yielding an additional L_2 loss for each entry in the parameter vector.

Relating this to Equation (3.30), the optimization loss can be expressed as,

$$\ell(\boldsymbol{y};\boldsymbol{\theta}) \simeq k_{reg}(\sum_j \theta_j^2) + \frac{1}{BN_{sim}} \sum_{i=1}^{B} \sum_{k=i}^{i+N_{sim}} \frac{1}{2}(y_k - \hat{y}_k)^2 \tag{3.32}$$

Where $k_{reg}(\sum_j \theta_j^2)$ is the joint negative log likelihood of the model parameters. The regularization constant k_{reg}, is a function of the hyperparameters used for Monte Carlo dropout.

Following the optimization, the model precision (inverse of the output variance) is found by maximizing the likelihood of the training data when simulating with uncertainty propagation using Algorithm 3. A grid search is used. This is somewhat similar to what is applied in [18] - the main difference being that they express the inherent noise *outside* of the dynamics (so as an additive noise term to $\hat{y}$), whereas it is here expressed *in* the dynamics (so as an additive noise term to Δx).

Algorithm 5: Stochastic Neural Difference Equation Optimization: Monte Carlo Dropout

Initialize the parameters $\boldsymbol{\theta}$ of the network randomly
while *training* **do**
  $\mathcal{L} \leftarrow 0$
  for $i = 1, \ldots, B$ **do**
    Sample sequence $(\boldsymbol{y}_{0\ldots N_{sim}}, \boldsymbol{u}_{0\ldots N_{sim}})$
    Initialize $x_0 \leftarrow y_0$
    for $k = 1, \ldots N_{sim}$ **do**
      $x_k \leftarrow ForwardWithDropout(x_{k-1}, \boldsymbol{u}_{k-1}; \boldsymbol{\theta})$
      $\hat{y}_k \leftarrow x_k$
    $\mathcal{L} = \mathcal{L} + \sum_{k=1}^{N_{sim}} \frac{1}{2} ||y_k - \hat{y}_k|| / (N_{sim} B)$
  $\mathcal{L} = \mathcal{L} + k_{reg}(\sum_j \theta_j^2)$
  Minimize $\mathcal{L}$ wrt. $\boldsymbol{\theta}_m$
// Find inherent noise through a grid search
Minimize NLL wrt. τ^{-1}

3.2.6 Relations to Existing Methods

The approach presented here builds upon the work presented in [89]. This is combined with existing work on the unscented Kalman filter [59] [66], and uncertainty estimation methods for neural networks [17] [16].

Existing approaches to modeling dynamical systems *and* their uncertainties using neural networks often employ stochastic simulations with *Monte Carlo dropout*. This involves repeatedly simulating the entire trajectory with dropout enabled [10] [18] [88] (though similar methods that employ deep ensembles have also been applied [19]).

The distinguishing difference between these approaches and the simulation method applied in this thesis, is in which order the marginalization over the present uncertainties is performed when performing inference. Both when applying Monte Carlo dropout and Deep Ensembles, inference is performed for a single step. Grid-based uncertainty propagation is used to propagate the Gaussian state belief through the state transition and process noise (both if which are given by the model).

Many existing approaches can informally be expressed as approximations of

$$p(x_t|x_0) = \int_\theta \int_{\tau=0}^{t} \int_{x_{\tau 1}} p(x_\tau|x_{\tau-d\tau};\theta)p(x_{\tau-d\tau})dx_{\tau-d\tau}d\tau p(\theta)d\theta \tag{3.33}$$

Meaning that they perform marginalization over the uncertainty in the model predictions *after* the simulation. They simulate a set of particles for the entirety of the time horizon, without interaction between the particles. In comparison, the approach considered here can be expressed as an approximation of

$$p(x_t|x_0) = \int_{\tau=0}^{t} \int_\theta \int_{x_{\tau-1}} p(x_\tau|x_{\tau-d\tau};\theta)p(x_{\tau-d\tau})dx_{\tau-d\tau}p(\theta)d\theta d\tau \tag{3.34}$$

Where the marginalization over the model's predictive uncertainty is performed at each time step. This means that every time step, the state distribution is approximated as a Gaussian, and sigma points for propagating the distribution the next time step are recomputed using that Gaussian.

A major benefit of existing approaches is that they are capable of predicting rich state distributions over time, as it is expressed as a mixture of several estimates. On the contrary, the approach presented here is more closely related to existing approaches employed in filtering. As a result, it can be applied directly in a Kalman filter, only requiring small modifications to the uncertainty propagation method - as outlined in Section 3.1.

Chapter 4

Synthetic Experiment

This chapter considers a qualitative synthetic experiment: Simulation of a dynamical state space model with a variable (heteroscedastic) process uncertainty.

4.1 Stochastic Nonlinear Orbiter

The experiment shows how we can apply uncertainty propagation on a stochastic system to predict the system state over time. The uncertainty propagation method described in Section 3.1 will be compared against a stochastic Monte Carlo Simulation with 10000 samples. This is a simulation with no feedback (i.e. not filtering). This is primarily a qualitative experiment. We will consider the effect of applying grid-based uncertainty propagation when a Gaussian approximation is reasonable, and also when it *isn't* reasonable.

The stochastic dynamicalk process we will consider is represented by the following difference equation,

$$\begin{aligned} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} &= h \begin{bmatrix} -d & -1 \\ 1 & -d \end{bmatrix} \begin{bmatrix} x_1 |\sin(x_1)| \\ x_2 |\sin(x_2)| \end{bmatrix} + h\gamma \boldsymbol{w}, \\ \boldsymbol{w} &\sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} |x_1|^{1/2} & |x_1 x_2|^{1/4}/8 \\ |x_1 x_2|^{1/4}/8 & |x_2|^{1/2} \end{bmatrix} \right) \end{aligned} \tag{4.1}$$

This is a nonlinear orbiter where the steps are approximated using Euler integration. Its trajectory is visualized in Figures 4.2 and 4.4. The system has an additive variable process noise $\boldsymbol{w}$ (white noise), whose magnitude is scaled by γ. The covariance of $\boldsymbol{w}$ is a function of the system's states.

For this experiment, the step length h will be fixed to $1/100$ and the damping d will be fixed to $1/12$. The simulation length is set to 30s. We will consider two separate values of the uncertainty scaling factor γ: $1/10$, and $3/10$. The first yields a near-Gaussian empirical state distribution, whereas the second results in a heavy-tailed, warped empirical state distribution.

4.2 Evaluation Overview

The evaluation will focus on the disparity between the predicted state distribution and the empirical state distribution (approximated by the Monte Carlo simulation). This is illustrated using graphs of predicted state distribution in comparison the the empirical state distribution. Additionally, two metrics will be considered:

1. **Eucludian distance**: Distance between the predicted state distribution mean and the empirical state distribution mean. Lower is better.
2. **Wasserstein$_2$ distance** [96]: A distance metric indicating how much work it would take to morph one probability distribution into another. This is computed between the predicted (Gaussian) state distribution and a Gaussian given by the empirical state distribution's mean and covariance. Lower is better.

4.3 Simulation Methods

4.3.1 Unscented Transform

The time update step of Algorithm 2 is used to simulate a Gaussian approximation of the system states over time.

4.3.2 Monte Carlo Simulation

A stochastic Monte Carlo simulation is used to simulate 10 000 particles. This procedure consists of two steps,

1. **Deterministic State Transition:** The deterministic state update term is computed, and the covariance matrix of $h\gamma\boldsymbol{q}$ is computed.
2. **Stochastic Sampling:** A random sample is drawn from the diffusion term specified by the process noise, and added to the state update.

The 10 000 particles are simulated individually for the time horizon. This results in a point cloud for each time step.

4.4 Results

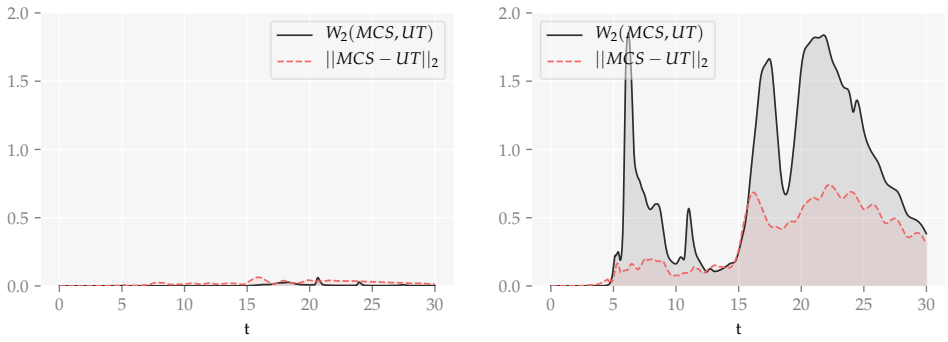
This section presents the results for the two separate values of γ.

The overall results are as one can expect: When the empirical state distribution is reasonably well approximated by a Gaussian, the unscented transform yields accurate results. When the empirical state distribution is not well-approximated by a Gaussian, the accuracy of the predictions is degraded. However, even if the empirical state distribution is highly non-Gaussian, the predictions from grid-based uncertainty propagation *can* still be reasonable.

4.4.1 Near-Gaussian Empirical State Distribution: $\gamma = 1/10$

The marginal distributions of the states over time are given in Figure 4.2, and their joint distribution over time are given in Figure 4.3. The Euclidian mean and Wasserstein distance over time are given in Figure 4.1a.

We can observe from Figure 4.2 that the marginal distributions of the states over time match closely between the unscented transform and the Monte Carlo simulation. The joint distributions of the Monte Carlo simulation in Figure 4.3 are somewhat heavy-tailed. Though a Gaussian state distribution isn't a perfect approximation of this, it is reasonable. The predictive mean in Figures 4.2 and 4.3 tracks the empirical mean closely. This is also illustrated well in Figure 4.1a.



(a) $\gamma = 1/10$ **(b)** $\gamma = 3/10$

Figure 4.1: Wasserstein Distance and Predictive Mean Euclidian Distance Over Time: W_2 denotes Wasserstein$_2$ distance, and $||\cdot||_2$ denotes Euclidian distance. When a Gaussian is a good approximation of the empirirical state distribution, grid-based uncertainty propagation can yield accurate predictions. This is shown in **(a)**. When a Gaussian is a poor approximation, the quality of the predictions will degrade. This is shown in **(b)**.

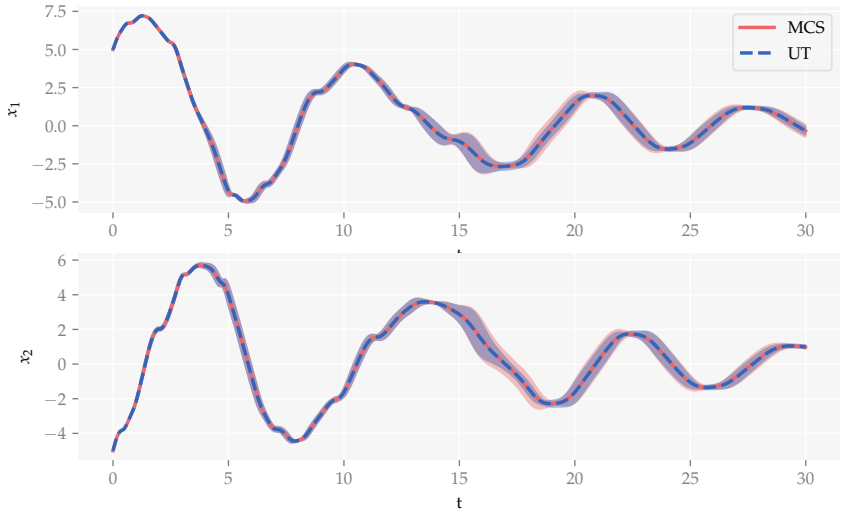


Figure 4.2: Marginal Distribution of Orbiter States for $\gamma = 1/10$: The marginal distributions of the Unscented Transform (UT) prediction overlaid a Gaussian approximation of the Monte Carlo Simulation (MCS) marginal distributions. Predictive means are given by the lines, and $\pm 3STD$ confidence intervals are given by the shaded area. Similarly to as seen in Figure 4.3, the predictions from the UT for $\gamma = 1/10$ follow the MCS distribution closely.

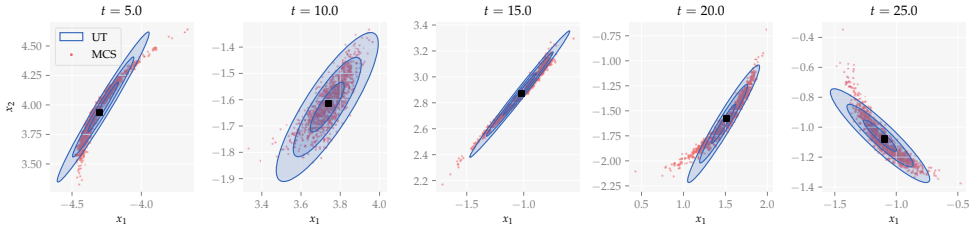


Figure 4.3: Joint Distribution of Orbiter States for $\gamma = 1/10$: The joint state distribution of the orbiter for 5 different time steps are shown. These are *snapshots* of the simulation in Figure 4.4. The Unscented Transform (UT) prediction is given by the shaded area (in blue), which marks 68.27%, 95.45%, and 99.73% confidence ellipses for the predictive mean. The Monte Carlo Simulation (MCS) is given by the scattered point predictions. Only 1000 of the 10 000 particles are shown. The mean of the Monte Carlo simulation is given by the square marker.

4.4.2 Heavy-Tailed Empirical State Distribution: $\gamma = 3/10$

The marginal distributions of the states over time are given in Figure 4.4, and their joint distribution over time are given in Figure 4.5. The Euclidian mean and Wasserstein distance over time are given in Figure 4.1b.

In comparison to Section 4.4.1, we can observe in Figure 4.1b that the approximation yields less accurate predictions in this case. The highly skewed and warped distribution of the Monte Carlo simulation can be seen well in Figure 4.5. A single Gaussian cannot approximate this well - so the accuracy will be lower. Still, though the overall predictions decline in quality, the empirical means of the Monte Carlo simulation in Figure 4.5 are all within the 99.73% confidence ellipses predicted by the unscented transform.

We can note that both the Wasserstein distance and Euclidian distance in Figure 4.1b decrease towards the end of the simulation. This is an important observation. As the particles in the Monte Carlo simulation converge towards the equilibrium in $[0, 0]^\top$, their distribution will be better approximated by a unimodal, non-skewed distribution such as a Gaussian.

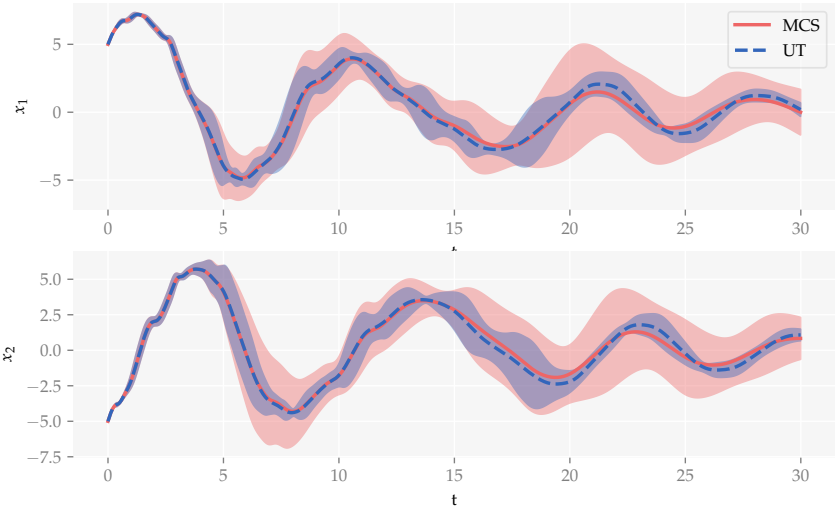


Figure 4.4: Marginal Distribution of Orbiter States for $\gamma = 3/10$: The marginal distributions of the Unscented Transform (UT) prediction overlaid a Gaussian approximation of the Monte Carlo Simulation (MCS) marginal distributions. Predictive means are given by the lines, and $\pm 3STD$ confidence intervals are given by the shaded area. Just as in Figure 4.5, we can recognize that the predicted variances do not follow very closely. However, the empirical mean of the MCS is reasonably close to the prediction of the UT for the entirety of the simulation.

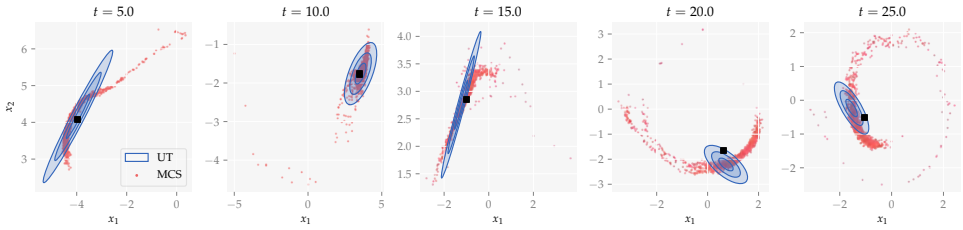


Figure 4.5: Joint Distribution of Orbiter States for $\gamma = 3/10$: The joint state distribution of the orbiter for 5 different time steps are shown. These are *snapshots* of the simulation in Figure 4.4. The Unscented Transform (UT) prediction is given by the shaded area (in blue), which marks 68.27%, 95.45%, and 99.73% confidence ellipses for the predictive mean. The Monte Carlo Simulation (MCS) is given by the scattered point predictions. Only 1000 of the 10 000 particles are shown. The mean of the Monte Carlo simulation is given by the square marker. The UT yields a poor approximation of the MCS distribution over time in this case. Due to the significant increase in process uncertainty compared to in Figure 4.3, the state distribution of the MCS becomes far more spread out. This heavy-tailed and warped distribution cannot be approximated well by a multivariate Gaussian, meaning that grid-based approximations like UT will perform poorly. *However*, despite significant differences in the predicted distributions, the empirical mean of the Monte Carlo simulation is reasonably close to the predictive mean of the UT, and is within the 99.73% confidence ellipse for all of the time steps shown in the plot.

4.5 Summary

The simulations in this experiment show that grid-based uncertainty propagation methods *can* yield accurate state distribution estimates, but also that they are *not guaranteed* to do so. The accuracy of the simulation considered in this experiment can be limited by the Gaussian approximation. However, in cases where the Gaussian assumption holds, a grid-based method can offer similar results to a Monte Carlo simulation at a fraction of the cost.

Aside from this, the observation concerning convergence of the Monte Carlo simulation is important. It can give us some intuition on what systems can be expected to be well suited to a Gaussian approximation. If the uncertainty in the state distribution *dissipates*, it will tend towards a single coordinate. If this occurs at a similar or faster rate than the state distribution is spread (e.g. due to uncertainty injected from process noise or warping due to nonlinearity in the dynamics), the assumption of a Gaussian state distribution can be expected to be reasonably accurate. Importantly, physical processes are dissipative - and Chapter 5 concerns a physical process.

Chapter 5

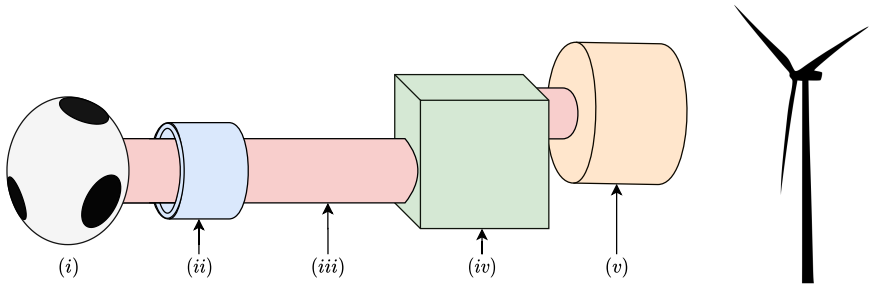
Wind Turbine Bearing Temperature Modeling

This experiment concerns the development of a wind turbine main bearing temperature model. The goal of this model is to produce predictions with uncertainty estimates for the main bearing temperature of a specific type of wind turbines located in a wind farm in Norway. This experiment will demonstrate how the methods in Sections 3.1 and 3.2 can be applied to solve a real-world problem. It will also serve as a basis for a discussion on the benefits, drawbacks, and practicalities of the methods presented in this thesis.

The chapter is divided into seven main parts.

1. Overview of the experiment. This introduces the process being modeled, the utility the model serves, and the evaluations which will be performed
2. Data overview and information on the pre-processing that has been performed
3. Overview of methods that have previously been applied to solve similar modeling tasks
4. Overview of the physics that govern the system in question
5. Information on the models developed for the experiment
6. More detailed information on the evaluation criteria and the cases considered for evaluation
7. Presentation of results from the evaluation

The *main bearing* of a wind turbine is a bearing which enables the turbine blades to rotate. The sheer size of larger wind turbines and the wind forces involved can put significant stress on this component. Figure 5.1a illustrates the main components of a vertical axis wind turbine, including the main bearing. Figure 5.1b illustrates a vertical axis wind turbine. The components shown in Figure 5.1a are located at the very top of Figure 5.1b.



(a) Wind Turbine Component Diagram **(b)** Wind Turbine

Figure 5.1: Illustrations: **(a)** Diagram showing some important components of a wind turbine: *i)* The hub where the rotor blades are mounted, *ii)* The main bearing in which the shaft rotates, *iii)* The main shaft, *iv)* The gearbox, and *v)* the generator. These are mounted inside the nacelle, at the top of the turbine. **(b)** Illustration of a vertical axis wind turbine from the Wikimedia Commons, licensed under the CC BY-SA 3.0 license.

5.1 Overview

The wind turbines in question are fitted with sensors which yield periodic measurements of several values (covered in Section 5.2). Among these is the temperature of the turbine main bearing, which the intent is to model. An important question to consider is *why should we model something which we already measure*? In this case, there are multiple moments which motivate developing a model - and these are summarized below.

1. **Bearing temperature estimates in presence of missing measurements**. As a significant portion of readings are missing, it is desirable to have estimates of the bearing temperature even when the measurements are not transmitted. More generally speaking, applying a suitable model in conjunction with a filter (e.g. a Kalman filter) can yield *both* accurate predictions when measurements are available, *and* estimates when measurements are unavailable (dead-reckoning).
2. **Data stream verification**. The labels of the data from the wind turbines are assigned manually, and discrepancies between a model and measurements can assist in determining if turbines have mislabeled measurements.
3. **Wind turbine health monitoring**. Significant deviations between model predictions and measurements can indicate a turbine with deteriorating health.

The above motivates the development of a model which is accurate the case of dead reckoning (lack of output feedback). A filtering algorithm can be applied in conjunction with this in order to incorporate sensor measurements in the case where they are available.

These use-cases are very well suited towards uncertainty estimation. An al-

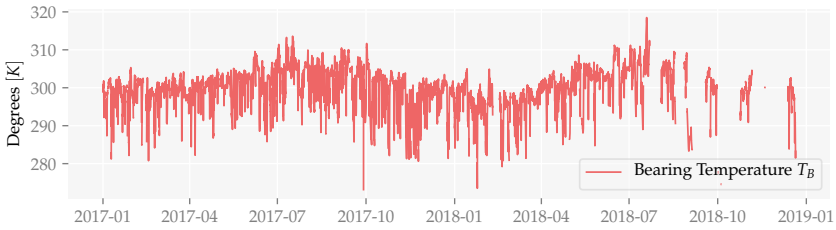


Figure 5.2: Bearing Temperature Measurements: Bearing temperature measurements from Wind Turbine 1 (*WTUR1*) over the course of two years. This is the variable that we are interested in modeling. Note the seasonality and non-negligible number of missing measurements. For an illustration of where the turbine bearing is located, see Figure 5.1.

gorithm capable of predicting the current bearing temperature along with a *calibrated* confidence interval would be ideal: This could for instance provide the means to approximate the current state of the turbines when temperature measurements are missing. Further, it could enable sensible weighting between the data reported by the turbines and the model prediction: A confident prediction that does not correspond to the measurements could be an indication of deteriorating turbine health or malfunctioning sensors. Likewise, a highly *un*confident prediction might indicate unexpected model inputs - hinting at mislabeled data.

To evaluate the models, two separate quantitative evaluations will be considered,

1. **Case A: Simulation**. Predicting the bearing temperature evolution in the wind turbines without any knowledge of the bearing temperature sensor measurements (*dead-reckoning*). The target is to forecast the temperature evolution over time.
2. **Case B: Filtering**. The bearing temperature measurements are assumed to arrive sequentially, and are used as measurements in a filtering algorithm. The target is to predict the temperature evolution *one step ahead*.

Following these cases, the interpretability of the resulting models will be briefly looked into. This will focus on the neural network-based models, since the method presented in Section 3.2 has not been identified in existing literature.

More in-depth information on the evaluation is presented in Section 5.6, and the results in Section 5.7. Before that, an overview of the data, previous work, system, and models will be presented.

5.2 Data Sourcing and Dataset Overview

A time series dataset for wind turbines located in Norway was obtained through Kongsberg Digital. The data, spanning just under 2 years, consists of the measurements shown in Table 5.1.

t [UTC]	T_B [°C]	T_E [°C]	P [kW]	ω [RPM]

Table 5.1: Wind Turbine Data Columns: Each row in the wind turbine data contains a timestamp t, main bearing temperature T_B, external temperature T_E, electrical power output P, and generator angular speed ω. The measurement units are shown in the table (RPM denotes rotations per minute).

The data is from four separate turbines: Wind Turbines 1, 2, 3, and 6 - from now on referred to as *WTUR1*, *WTUR2*, *WTUR3*, and *WTUR6*. The data spans two years per turbine (January 1, 2017 - December 21, 2018), and measurements are equally spaced with a 10 minute period. There are 103 620 rows of measurements in total for each of the four turbines. A significant number of the values are missing: Between 26.8% (for WTUR1) and 34.2% (for WTUR6) per turbine. Tables B.1 to B.4 in the appendix give a more more detailed overview of the statistics of the data, including how the missing measurements are distributed for each turbine. The bearing temperature measurements from WTUR1 are shown in Figure 5.2.

Training, validation, and test data

Since WTUR1 has the lowest number of missing values, it is used for training and validation. The data from WTUR1 spanning the first 12 months is used for training models. The following 6 months of data from the same turbine is used in the evaluation. The remaining data from the turbine is not used, as it is missing a majority of the values.

The data from the other three turbines is used as test data to examine generalization of the models when evaluating. Similarly as for WTUR1, only the first 18 months of the data is used. Figure 5.3 shows the sensor readings from WTUR1 from the first week of the data. As can be seen there, the temperature measurements vary smoothly, but the generator speed and power output rise and fall more abruptly.

Unmeasured Variables and Associated Uncertainties

There are several unmeasured variables that can have an impact on the temperature change in the wind turbines. A prime example of this is the cooling system, which will certainly affect the temperature in the turbine bearing. We can expect these unmeasured variables to limit the predictive accuracy and result in added process uncertainty. However, it is likely that there are correlations between some of the unmeasured variables and the variables that are measured - which can be possible for a model to learn.

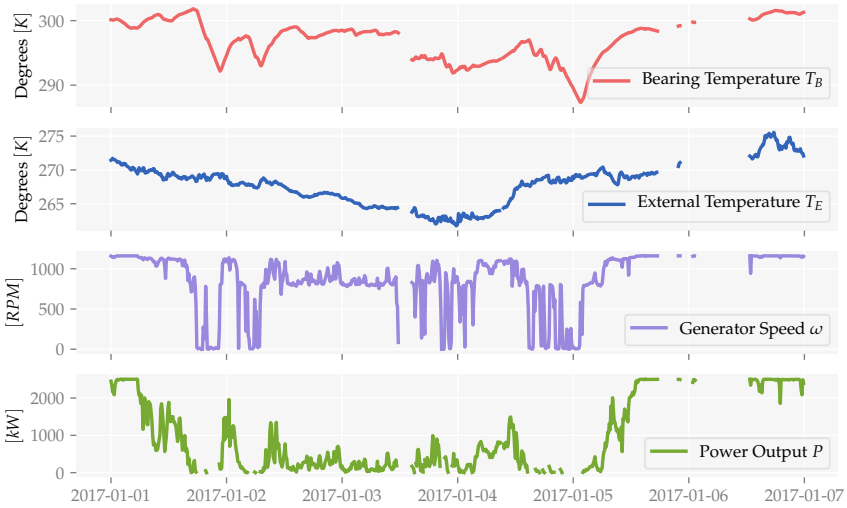


Figure 5.3: WTUR1 measurements: The first week of sensor measurements from Wind Turbine 1 (*WTUR1*). Remark the missing measurements. The temperatures vary smoothly, whereas the generator speed and power output vary more abruptly.

5.2.1 Data Pre Processing

The data has a significant amount of missing measurement values. This results in two challenges: *i)* How to clean the data, and *ii)* How to make predictions when input measurements are missing. Additionally, anomalies in the data need to be tackled.

Unit Changes

Temperature measurements were first converted to Kelvin, as it is more convenient for expressing the physics which govern energy balance.

Anomalies and Outliers

An initial exploratory data analysis revealed that there were some anomalies in the data. Most notably the external temperature measurements from Wind Turbine 3, which can be seen in Figures 5.4 and 5.5. The measurements are are likely *i)* mislableled, or *ii)* the result of a faulty sensor. It was decided to still use the data for testing, as it can give insight into the effect of erroneous data on the model predictions.

In addition to this very notable anomaly, some outliers were present, such as a small amount of negative generator speeds, and a small number of negative

power output values. Some of these are caused by the turbines grid-sourcing electricity for starting the blade rotation [97], but some have greater magnitude and are likely due to temporary sensor malfunction. It was initially attempted to remove outliers using a quantile-based method (Tukey's method [98]). This yielded improved one-step-ahead predictor accuracy, but caused a degradation in longer horizon simulation accuracy for the models tested at that point. It was therefore decided to use a simple method: Clipping measurements to be ≥ 0, an example of which is shown in Equation (5.1). Figure 5.5 shows the marginal distributions of the turbine measurements after the clipping.

$$T_B(t) \leftarrow \max(T_B(t), 0) \tag{5.1}$$

Replacing Missing Measurements

For simulation, missing input measurements are replaced by the previous available measurement ('forward filling'), shown in Equation (5.2). When there are large measurement gaps, this results in significant degradation in the prediction quality. The reason forward filling is used in place of more advanced gap-filling methods, is that simulation is performed as if running on-line with measurements arriving in real time (so future measurements are assumed to be unknown). Forward filling is a simple method to handle missing data in cases where there is no opportunity to fill gaps by interpolation.

$$T_B(t) \leftarrow \begin{cases} T_B(t), & \text{if } T_B(t) \neq NaN \\ T_B(t - \Delta t), & \text{else} \end{cases} \tag{5.2}$$

Normalization

For training and simulating neural networks, the data was normalized column-wise to have a mean of 0 and a variance of 1. This was done using the sample mean and sample standard deviation of the respective data column from Wind Turbine 1. Equation (5.3) shows how this is done for a single measurement. This same mean and standard deviation is also used for normalizing the data from other turbines, as the statistics of their data are not assumed to be known.

$$T_B(t) \leftarrow (T_B(t) - \mu_{T_B})/\sigma_{T_B} \tag{5.3}$$

5.3 Previous Work

Models for the wind turbine bearing temperature have previously been developed by Kongsberg Digital. The models include linear state space models and (shallow) autoregressive neural network models. They are currently running in production. These models run in *dead reckoning*, predicting the current bearing temperature

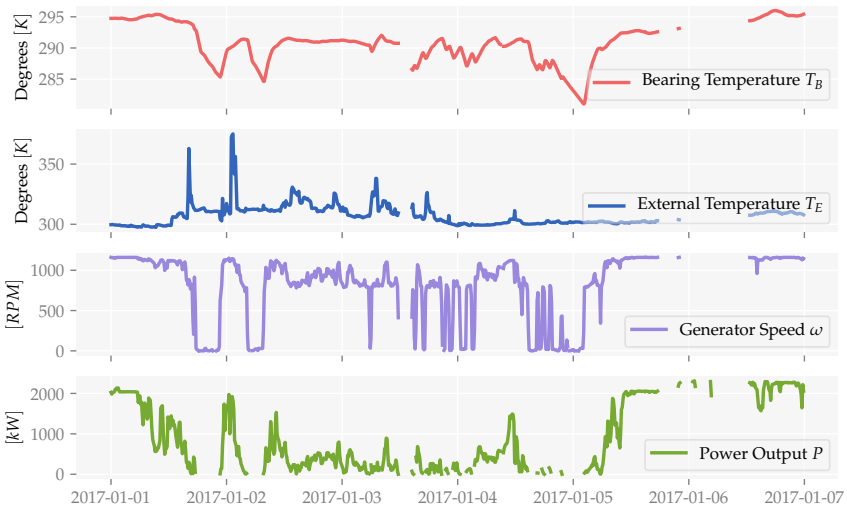


Figure 5.4: WTUR3 measurements: The first week of sensor measurements from Wind Turbine 3 (*WTUR3*). The external temperature measurements are distinctly different from what is measured by WTUR1 during the same period (see Figure 5.3).

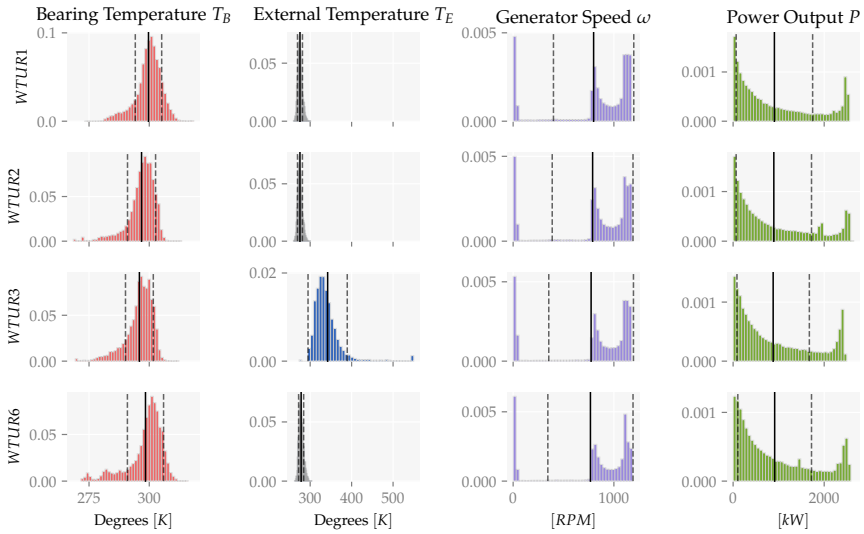


Figure 5.5: Histograms of All Measurements: A comparison of the measurement marginal distributions for all the wind turbines. Each row corresponds to a wind turbine, and each column corresponds to a sensor. Means $\pm 1STD$ are indicated by the vertical lines. Note the discrepancy between the external temperature (T_E) measurements: WTUR3 likely has mislabeled data or a malfunctioning sensor. More detailed statistics are in the appendix: Tables B.1 to B.4.

using only the other sensor readings. In addition to this, a Kalman Filter has been applied for filtering the sensor measurements, but not in combination with the models. These models have been developed in C#, but the source code is unavailable so they cannot be directly used as baselines, nor are exact performance metrics available. The general approaches used are however a suitable starting point for developing a model with uncertainty estimates.

A related approach was identified in [30]. The authors apply a physics-based model for health monitoring of wind turbines. Their model is a one-step-ahead predictor, which was verified in an exchange with the primary author. Aside from that, there is a difference in the sensors that are used: Their model uses the turbine nacelle temperature - not the external temperature.

The parameters in the linear models developed by Kongsberg Digital and [30] were identified using a least square's fit. The parameters in the autoregressive neural networks developed by Kongsberg Digital were identified by optimizing on single-step predictions using the *limited-memory BFGS* algorithm [99, p. 177].

5.4 Physical Considerations

Before formulating models for the wind turbine bearing temperature, it can be useful to reason about what to expect. In order to get an understanding of how the measured variables interact with the turbine bearing temperature, we will briefly review some of the underlying physics.

Similarly to as in [30], this will yield a set of differential equations. These equations will then be used to formulate state space models and neural network-based models.

5.4.1 Energy Balance Equations

We begin by considering the physical equations that govern the system in question. These can be formulated using energy balance [50, Chapter 11.4].

Let V be a fixed material volume [50, Chapter 11.4.1] which encloses the particles in the main bearing, and is assumed to have constant density ρ. The rate of change in the total energy in the system can be expressed using [50, Eq. (11.172)] the rate of change in the specific internal energy $u[\mathrm{J\,kg^{-1}}]$, specific kinetic energy $e_k[\mathrm{J\,kg^{-1}}]$, and specific potential energy $\phi[\mathrm{J\,kg^{-1}}]$,

$$\underbrace{\frac{d}{dt}\iiint_V \rho(u+e_k+\phi)dV}_{\text{rate of change in the total bearing energy}} = \underbrace{-\iint_{\partial V} \rho(u+\frac{p}{\rho}+e_k+\phi)\vec{v}\cdot\vec{n}dA}_{\text{rate of energy change due to pressure work and convection}\simeq 0} \underbrace{-\iint_{\partial V} \vec{j}_Q\cdot\vec{n}dA}_{\text{rate of energy change due to conduction}} \tag{5.4}$$

The body forces acting on the bearing (e.g. by the turbine shaft) are expressed as the gradient $\vec{\nabla}\phi[\mathrm{N\,kg^{-1}}]$ of the potential field. The conduction term $-\vec{j}_Q[\mathrm{W\,m^{-2}}]$ expresses the flux of heat in and out of the bearing volume surface. The material volume is fixed in the nacelle of the wind turbine. We assume there is no flux of particles, so we will neglect the convection terms and pressure work in Equation (5.4).

Approximations of the Energy Balance Equations

We approximate the bearing as a point mass, with mass $m = \rho V[\mathrm{kg}]$, specific heat capacity $c_p[\mathrm{J\,K^{-1}\,kg^{-1}}]$, and uniform temperature $T_B[\mathrm{K}]$. The internal energy can then be expressed as $U = c_p m T_B[\mathrm{J}] = C_p T_B[\mathrm{J}]$. The heat conduction is approximated by a finite number of heat fluxes $q_i[\mathrm{W}]$. The power exerted by the body forces (the integral over $\vec{\nabla}\phi[\mathrm{N\,kg^{-1}}]$) will be expressed as $P_b[\mathrm{W}]$. The change in the bearing kinetic energy will be expressed as $dE_k/dt[\mathrm{W}]$.

We can expect a heat flux proportional to the difference in temperature between the surrounding material and the temperature in the main bearing, which can likely be approximated well in terms of the external temperature $T_E[\mathrm{K}]$ and the thermal resistance $R[\mathrm{K\,W^{-1}}]$. The shaft rotates in the bearing, and this will result in a heat flux caused by dissipation due to the friction force $F_f[\mathrm{N}]$. Further, there is likely to be a heat flux due to electrical dissipation in the generator, which can be approximated using the power output $P[\mathrm{W}]$ of the turbine. Lastly, there are unmodeled phenomena, which will be represented by a unknown heat flux $w[\mathrm{W}]$. We can then rewrite Equation (5.4) to obtain a similar expression as considered in [30],

$$\frac{d}{dt}C_p T_B \simeq P_b - \frac{d}{dt}E_k + \sum_i q_i + w \tag{5.5}$$

$$\simeq \underbrace{P_b}_{\text{power exerted by body forces}} - \underbrace{\frac{d}{dt}E_k}_{\text{change in kinetic energy}} + \underbrace{R^{-1}(T_E - T_B) + k_p P + k_f \omega F_f}_{\text{heat fluxes}} + \underbrace{w}_{\text{unmodeled phenomena}} \tag{5.6}$$

Where k_p is a dimensionless constant. $k_f[\mathrm{RPM/(m/s)}]$ is a conversion factor from the wind turbine generator speed to the magnitude of the sliding velocity for the friction contact surface between the shaft and main bearing.

These equations are only an approximation, but they do give an indication to what we can expect from the models. Having obtained some intuition on the system in question, we proceed to identifying models for the bearing temperature.

5.5 Wind Turbine Model Identification

Several methods were iteratively developed to serve as baseline models for the system in question. These can roughly be divided into three categories,

1. **Least squares' models**: Linear state space models, parameters found from the least squares' solution of a single prediction step (similar to [30] and the linear models developed by Kongsberg Digital)
2. **System identification models**: Linear state space models, parameters found using the *Canonical Variate Analysis* [26] system identification algorithm
3. **Neural network models**: Neural network-based state space models, parameters found from optimization on simulation trajectories

The models which will be considered in more detail are the following:

1. `LS`: Least-squares'-based first order linear state space model
2. `LS-B`: Least-squares'-based first order linear state space model, with a bias term
3. `LS-H`: Least-squares'-based first order linear state space model, with a static (*Hammerstein* [100]) nonlinearity
4. `LS-HB`: Least-squares'-based first order linear state space model, with both a static (*Hammerstein* [100]) nonlinearity and a bias term
5. `CVA-1`: System-identification-based first order linear state space model with bias term and Hammerstein nonlinearity
6. `CVA-2`: System-identification-based second order linear state space model with bias term and Hammerstein nonlinearity
7. `BMA`: Bayesian model average of the two `CVA` models
8. `MCD`: First order probabilistic neural network state space model, based on *Monte Carlo Dropout* [16]
9. `DE`: First order probabilistic neural network state space model, based on *Deep Ensembles* [17]

The following subsections describe how these models were identified and optimized. The evaluation and results are described in Sections 5.6 and 5.7.

5.5.1 Least Squares' Models

Four kinds of least squares'-based models were identified. These will be described briefly here.

LS

A simple linear model can be expressed using Equation (5.6). We consider only the heat fluxes of the right hand side, assume Coloumb friction, and discretize the equation, to obtain the `LS`-model in Equation (5.7),

$$\Delta T_{B,k+1} = \begin{bmatrix} \beta_1 & \beta_2 & \beta_3 & \beta_4 \end{bmatrix} \begin{bmatrix} T_{B,k} \\ T_{E,k} \\ P_k \\ \omega_k \end{bmatrix} \tag{5.7}$$

Where k is the time step, $\Delta T_{B,k+1}$ is the change in bearing temperature from step k to $k+1$, and β_i are the system parameters. This is among the models considered by Kongsberg digital.

LS-B

When approximating Equation (5.6) as Equation (5.7), we are implicitly linearizing the system equations with respect to the variables used, about their origin. By adding a bias term to Equation (5.7) we can shift the equilibrium point of the linearized system, which might improve predictive performance. This can also account for steady-state offsets (e.g. due to unmodeled heat fluxes). The `LS-B`-model is obtained by including a bias term to the right hand side of Equation (5.7).

LS-H

A model similar to the one considered in [30] can be found by using a viscous friction model, which results in a Hammerstein state space model [100] (with a static input nonlinearity). The resulting quadratic term can possibly also encode some information about the kinetic energy of the system. It may be beneficial to use a combination of the friction models, as viscous friction forces generally exhibit nonlinear behaviour [50]. The `LS-H` model is found by including the nonlinear friction term $\beta_5\omega^2$ to the `LS` model.

LS-HB

`LS-HB` combines `LS-H` and `LS-B`, yielding Equation (5.8). As will be seen in Section 5.7, this yields improvements over the other least squares' models for simulation tasks.

$$\Delta T_{B,k+1} = \begin{bmatrix} \beta_1 & \beta_2 & \beta_3 & \beta_4 & \beta_5 & \beta_6 \end{bmatrix} \begin{bmatrix} T_{B,k} \\ T_{E,k} \\ P_k \\ \omega_k \\ \omega_k^2 \\ 1 \end{bmatrix} \tag{5.8}$$

We recognize that all of the least squares' models can be expressed discrete first order state space models. For instance, Equation (5.8) can be written as Equation (5.9).

$$\Delta T_{B,k+1} = \beta_1 T_{B,k} + \begin{bmatrix} \beta_2 & \beta_3 & \beta_4 & \beta_5 & \beta_6 \end{bmatrix} \boldsymbol{u}_k \tag{5.9}$$

Where $\boldsymbol{u}_k = [T_{E,k}\ P_k\ \omega_k\ \omega_k^2\ 1]^\top$ are the sensor measurements (and bias constant) at step k, and $\Delta T_{B,k+1}$ is the change in bearing temperature from step k to $k+1$. For simulation, the temperature steps can then approximated using the previous bearing temperature predictions.

Parameter Identification

The model parameters were found using a least squares' fit. This is among the approaches previously used by Kongsberg Digital, and was also done in [30].
All of the least squares' models above can be expressed as Equation (5.7) can be rewritten as,

$$\Delta T_{B,k+1} = \boldsymbol{\beta}^\top \boldsymbol{f}_k \tag{5.10}$$

Where $\boldsymbol{\beta}$ is the vector of N_{param} parameters, and $\boldsymbol{f}_k$ is a vector of features (bearing temperature and the other measurements) at time step k. Using this we can obtain a system of N_D equations, with N_D being the number of usable data points (that do not contain NaN values):

$$\underbrace{\boldsymbol{\Delta T_B}}_{[N_D]} = \underbrace{\boldsymbol{F\beta}}_{[N_D, N_{param}][N_{param}]} \tag{5.11}$$

The solution to the least squares problem $\min_{\boldsymbol{\beta}} ||\boldsymbol{\Delta T_B} - \boldsymbol{F\beta}||_2$ can be in general found using the Moore-Penrose generalized inverse ('pseudoinverse'). The `PyTorch` `lstsq` implementation was used to solve Equation (5.11) for all of the least squares models.

5.5.2 Canonical Variate Analysis Models

For improving simulation performance (performance in *dead-reckoning* when no output feedback is available), it is sensible to use parameters that are optimized minimize the predictive error in longer simulation trajectories. One approach to this is to use system identification algorithms for state space models in order to identify the structure and parameters of a state space model. An added benefit of this approach is that system identification algorithms estimate the error covariances in the models - a step towards estimating the uncertainty in the temperature predictions (though it can also be done for a least squares' regression).

CVA-1 and CVA-2

A first order and second order state space model are considered: `CVA-1` and `CVA-2`. The motivation behind the second order model is that the nacelle material might act like a heat buffer between the main bearing and the external surroundings,

resulting in second order dynamics. The CVA models use the same input features as LS-HB. Both of these models are discrete state space models, on the form given in Equation (5.12),

$$\begin{aligned} \Delta \boldsymbol{x}_{k+1} &= \mathbf{A}\boldsymbol{x}_k + \mathbf{B}\boldsymbol{u}_k + \mathbf{K}v_k \\ T_{B,k} &= \mathbf{C}\boldsymbol{x}_k + \mathbf{D}\boldsymbol{u}_k + v_k \\ v_k &\sim \mathcal{N}(0, R) \end{aligned} \tag{5.12}$$

Parameter Identification

The *canonical variate analysis* [26] algorithm is used for system identification, with the MATLAB [6] n4sid implementation. Details on the settings used are given in Table B.5 in the appendix. Forward filled data from WTUR1 was used for optimization, which is suboptimal. As the models benefit from having data from both summer and winter (the dynamics are impacted by seasonality) and n4sid requires a continuous data sequence, a trade-off is required. Despite this, the CVA models outperform the least squares' models, as will be seen in Sections 5.7.1 and 5.7.2.

The sensor noise in the temperature measurement and the process noise in the dynamics are expressed by v_k and $\mathbf{K}v_k$ in Equation (5.12), respectively. Note the single noise source ; n4sid yields models with noise terms given in the innovations form (instead of separate noise sources for the process and measurement).

5.5.3 Bayesian Model Averaging

Neither of the CVA models explain the bearing temperature evolution perfectly. As outlined in Section 2.3.2, Bayesian model averaging can improve predictive accuracy and calibration when multiple models are available but none of them fully explain the underlying process.

The CVA-models can be simulated using the prediction step of a Gaussian filtering algorithm to obtain the bearing temperature estimates and associated uncertainties over time. A simplified variant of the approach presented in [27] for Bayesian model averaging is applied to combine the model predictions.

The model average is given by the mixture of the individual model predictions, where the mixture weights are chosen to maximize the likelihood of the resulting predictions. This is not a true *Bayesian* model average as there is no prior distribution on the models or their parameters - but it is a simple and effective method. Similarly to in [27], independence of all forecast errors will be assumed. As is noted in [27], this is not likely to hold, but is unlikely to significantly impact the resulting mixture weighting. Similarly to as in [17], the mixture distribution will be further approximated by a Gaussian with the mixture mean and variance.

Denoting the model yielding the following constrained optimization problem,

$$\begin{aligned} \underset{w_1,w_2}{\operatorname{argmax}}\, \ell(\mathcal{D}|w_1,w_2) \simeq \underset{w_1,w_2}{\operatorname{argmax}} \sum_{\forall k} \log \mathcal{N}(T_{B,k};\mu_k,\sigma_k) \\ s.t. \\ \mu_k = \sum_{i\in(1,2)} w_i y_{i,k} \\ \sigma_k = \sum_{i\in(1,2)} w_i (y_{i,k}^2 + P_{y_i,k} - \mu_k^2) \\ w_1 + w_2 = 1, \quad 0 \leq w_1, \quad 0 \leq w_2 \end{aligned} \tag{5.13}$$

Due to the constraints on w_i and small number of mixture entries, this can be solved quite easily. A grid search (with steps spaced with 0.01) over w_1 was used to do so. It was found that the predictions were still overconfident, so they were then post-processed by including an additional inherent noise given by a constant standard deviation - similarly to in [18]. Both the grid search and computation of the additional inherent noise were performed over the training data. As maximum likelihood estimation is prone to over-fitting, using a hold-out validation set would in general be preferable. However, since the predictive errors were similarly distributed over time for a given turbine, doing so would not have made much of a practical difference in this case. For larger mixture sizes (or if a Gaussian output approximation is not suitable), the iterative approach described in [27] is more suitable.

5.5.4 Neural Network Models

The inclusion of a nonlinearity and bias for the linear models improved predictive performance. To further iterate on this, a black-box nonlinearity such as a neural network can be a suitable approach. This can enable us to model Equation (5.6) more accurately, as nonlinear relations between the inputs can be learned.
The process of developing neural networks for this purpose spanned a large portion of the semester. Part of the reason for this is that *a)* the Hammerstein nonlinearity and bias addition, and *b)* system identification with CVA yielded surprisingly good results. Obtaining results comparable or better than the `CVA` models proved surprisingly challenging. An in-depth documentation of all the approaches that were attempted would yield a far too broad scope. In order to motivate the methods used in the end, some approached that were attempted but were not used in the end will first be briefly covered.

Static feedforward models were briefly tested. As the wind turbine temperature is dissipative, the current temperature should be possible to approximate as a function of a sliding window of previous input measurements. These were found to yield too noisy predictions in practice - likely a result of large variations in the turbine generator speed and power output between time steps. This incudes variants based on convolutional networks.

It was found that single-step-optimized autoregressive neural networks (which have been used by Kongsberg Digital) did not improve upon the system identification-based models. This can be explained by the improvement in performance from the Hammerstein nonlinearity and bias. The same was observed for extreme learning machines, which is a neural network variant that is optimized using a kernel regression algorithm [101].

Recurrent neural networks such as LSTMs have been applied successfully to model complex time series [10] [18]. They were not used due to previous observations: More specifically that `CVA-2` did not significantly outperform `CVA-1` for other turbines. This indicated that a recurrent neural network with multiple hidden internal states could likely yield an unnecessarily complex model for the given purpose.

Optimization on Prediction Horizons

The distinguishing difference between the initial least squares' approach and the system identification algorithms is *optimization on prediction horizons* rather than single-step optimization. As this improves simulation performance for linear systems, applying similar methods for neural networks is reasonable. Instead of optimizing the network on the output from a single step, one can recursively apply the model to predict a trajectory, and optimize on the entire trajectory.

As such, it was chosen to *i)* keep the vector field representation used by the linear models, but *ii)* optimize it on long horizon trajectories rather than single step predictions. In practice, this yields a model similar to the neural ordinary differential equations considered in [37] (though given in terms of *difference* equations).

Incorporating uncertainty estimates into such models spawned the idea of *learning process noise* in addition to the process dynamics, using *Deep Ensembles* [17]. However, simulating these stochastic vector fields requires propagating uncertainty through them (since even if the first input is deterministic, the output that the next prediction will be based on is stochastic). As a bearing temperature model will be dissipative, the assumption of a Gaussian probability distribution for the temperature seems reasonable: A similar assumption is e.g. imposed in [27].

The method in Section 3.1 was then developed. The approach resulting from this is outlined in Section 3.2.

Bearing Temperature Stochastic Neural Difference Equation

As described in Section 3.2, the neural networks model a state transition with a diffusion term. This yields networks which jointly model the bearing temperature change, and the uncertainty in the predicted change. Rewriting Equations (3.24) to (3.27) in terms of the bearing temperature, this can be expressed as,

$$x_{k+1} = x_k + f_\theta(x_k, \boldsymbol{u}_k) + w_{\theta,k}, \tag{5.14}$$
$$T_{B,k} = x_k + v_k, \tag{5.15}$$
$$w_{\theta,k} \sim \mathcal{N}(0, Q_\theta(x_k, \boldsymbol{u}_k)) \tag{5.16}$$
$$v_k \sim \mathcal{N}(0, R_k) \tag{5.17}$$

Just as in Section 3.2, the process dynamics f_θ are given by the *predictive mean* of a neural network with uncertainty estimation, and the process noise $w_{\theta,k} \sim \mathcal{N}(0, Q_\theta(x_k, \boldsymbol{u}_k))$ has variance given by the *predictive variance* of the same neural network.

Optimizing on the simulation output estimates also requires estimating or learning the noise in the bearing temperature sensor. To limit complexity it was chosen to estimate it, based on the `IEC 60751:2008` standard [102] for temperature sensors. Standard B class platinum resistance thermometers (see e.g. the WIKA `IN 00.17` data sheet [103]) have a tolerance of $\pm(0.30 + 0.005|t|)$K, where $|t|$ is the absolute value of the temperature in °C and K is degrees Kelvin. Assuming an operating temperature of 50°C this gives ±0.55K. This is approximated as an additive white noise term v with zero mean and standard deviation 0.18K.

DE

The ensemble consists of 4 networks, each with a single hidden layer of 16 nodes. The output layer has two nodes (predictive mean and variance). This is the same structure as shown in Figure 3.2. It is optimized using the AdaBound optimzer [3]. Full details are given in Table 5.2. The structure of the network is as shown in Figure 3.2. It has the same input vector as `LS-H`. The motivation behind including the quadratic term is that it is a complex function to learn for a small network with a single hidden layer.

MCD

The Monte Carlo dropout network has a single hidden layer with 12 hidden nodes. The output layer has a single hidden node (as the inherent noise is a constant). This is the same structure as shown in Figure 3.3. It is optimized using the AdaBound optimzer [3]. Full details are given in Table 5.3. The structure of the network is as shown in Figure 3.3. Like `DE`, the `MCD` network has the same input vector as `LS-H`.

Hyperparameter	**Value**
Loss	Simulation mean NLL
Training Sequence Length	20
Minibatch Size	8
Minibatch Iterations	11500
AdaBound Learning rate	5×10^{-4}
AdaBound Final Learning rate	1×10^{-7}
AdaBound Gamma	1×10^{-8}
Ensemble size	4
Hidden nodes	16
Activation Function	$2\tanh(x)$
Variance Rectifier	Softplus: $\ln(1 + e^x)$
Dropout	0.05
Training simulation	Algorithm 3

Table 5.2: Deep Ensemble Neural Network Hyperparameters: The settings used for training the Deep Ensemble network wind turbine model. Note that dropout was only active when training, not during evaluation.

Hyperparameter	**Value**
Loss	Simulation MSE + Parameter L_2 reg.
Training Sequence Length	32
Minibatch Size	8
Minibatch Iterations	16000
AdaBound Learning rate	5×10^{-4}
AdaBound Final Learning rate	1×10^{-7}
AdaBound Gamma	1×10^{-8}
Hidden nodes	12
Activation Function	$2\tanh(x)$
Dropout	0.1
τ (inverse normalized variance)	1000
l	1×10^{-2}
T	50
Training simulation	Forward pass with Dropout

Table 5.3: Monte Carlo Dropout Neural Network Hyperparameters: The settings used for training the Monte Carlo dropout network wind turbine model.

Optimizer Selection, Architecture Selection and Parameter Identification

The L-BFGS optimizer in `PyTorch` was tested, but it was found that it failed to converge. It was chosen to use AdaBound instead, due to previous experience with it.

Determining the architecture for the neural networks was somewhat challenging. The networks are rather small (especially the Monte Carlo dropout-based network). Determining their sizes, activation functions and other parameters was an iterative process which was performed manually.

Hidden layer sizes were chosen to be as small as possible without degrading predictive performance. As the `DE`-networks have two outputs, its networks required a larger hidden layer than the `MCD`-network. The activation function was chosen due to it being symmetric and centered about 0. A smooth rectifier (*softplus*) was tested as well, but this did not yield good results.

The neural network models were optimized as outlined in Section 3.2, using Algorithm 4 and Algorithm 5. Checkpointing was used when optimizing, rather than early stopping. The parameters used for evaluation in Section 5.6 are from the checkpoint where the model's MSE was the lowest on the validation data - and the *Minibatch Iterations* noted in Tables 5.2 and 5.3 are the number of optimization steps that had been performed at this point.

The motivation for checkpointing was previous observations from training dynamic neural network models: That the validation performance may plateau temporarily. This likely ties into that optimization is performed on relatively short time sequences, whereas validation is performed on a long sequence. A small localized model error might yield significantly decreased validation results, but encountering this situation during training may be rare.

5.6 Model Evaluation Overview and Criteria

All the models are first quantitatively evaluated on two distinct cases: Simulation, and filtering. Following these two cases, we consider a qualitative comparison of what the models have learned. This focuses on `DE`, `MCD` and `CVA-1`, where we look into the vector field representations of these models and the uncertainties learned by the neural networks.

5.6.1 Case A: Simulation

For simulation, the true temperature of the main bearing is not assumed to be known by the models. This means that they are predicting in *dead-reckoning*, for long time horizons. The evaluation data for WTUR1 spans January 1, 2018 - June 1, 2018 (inclusive), so it is chronologically after the training data. The evaluation data for the other turbines spans January 1, 2017 - June 1, 2018 (inclusive).

When applying the models for simulation on other turbines, it was observed that the predictive performance declined significantly. A bias correction is therefore performed, which compensates some for this. This is a post-processing step that uses the first 28 days of measurements in 2017.

5.6.2 Case B: Filtering

When filtering, the models obtain measurements sequentially. They predict the turbine bearing temperature one step ahead. The data spans the same time ranges as given in Section 5.6.1. The least squares' models use the previous temperature measurement directly to predict the next, as in [30]. The neural network models use the entirety of Algorithm 2. The system identification-based models use an unscented Kalman filter (as it had been implemented - a conventional Kalman filter could be used). Additionally, a naive model which simply predicts the previously measured temperature is included in the metric tables when filtering. The `BMA` model is not considered when filtering. As its predictions are given by a mixture of the output from `CVA-1` and `CVA-2`, it could have been computed when filtering as well. However, as the filtering algorithms already add an additional layer of complexity, it was chosen to not consider `BMA` for Case B.

Part of the evaluation when filtering is based on predicted temperature change (steps). The steps from the Kalman filtered models are computed as $\hat{T}_{B,k|k-1} - \hat{T}_{B,k-1|k-1}$. This is the difference between their predicted bearing temperature at step k *before* the measurement update, and their predicted bearing temperature at step $k-1$ *after* the measurement update. The steps from the least squares' models are computed as $\hat{T}_{B,k} - T_{B,k-1}$. This is the difference between their predicted bearing temperature at step k, and the previous bearing temperature measurement.

No bias correction is performed when filtering.

5.6.3 Evaluation Criteria for Simulation and Filtering

The same evaluation criteria will be used for both of the two cases in this experiment. They have been chosen in an attempt to evaluate both the predictive accuracy (in terms of predicted mean), and the uncertainty estimate calibration (in terms of overlap between the predicted uncertainty and empirical error).

The metrics in the evaluations are computed using the available bearing temperature measurements (so if missing, they are *not* replaced with forward-filled values). There isn't a perfect overlap between missing bearing temperatures and the missing inputs, which means that the models are using forward-filled inputs for parts of the time steps they are evaluated on. This can be expected to degrade accuracy - especially for the turbines with more missing measurements (notably WTUR6). It was decided to evaluate the models in a setting that is reasonable to implement and deploy.

The criteria used are,

1. **Mean absolute error (MAE)** [104]: Evaluates predictive accuracy. This gives an indication of how inaccurate the predictions on average are. This metric was chosen as it is easily interpretable and not overly sensitive to outliers. Lower is better.
2. **Root mean square error (RMSE)** [104]: Evaluates predictive accuracy. This gives an indication of how inaccurate the predictions are, indicating

the standard deviation of the errors under a Gaussian assumption. Lower is better.

3. R^2 **score** [92]: This evaluates the predictive accuracy. The coefficient of determination is a measure of how much of the target variable variation is explained by the model. Closer to 1 is better. A score at or below 0 indicates the model predictions provide no information. The baseline is a naive model, which has the score of 0. This model predicts the mean temperature for the simulation experiments, and the previous measured temperature for the filtering experiments. For simulation, we consider the R^2 score of the predictions. For filtering, we consider the R^2 score of the predicted steps (as the previous temperature is known, so the R^2 of the predictions would be near 1 regardless of model accuracy).
4. **Empirical coverage of predicted** 95% **confidence intervals**: Evaluates calibration. This evaluates the predictive uncertainty by computing the percentage of bearing temperature measurements that fall within the 95% confidence intervals a model predicts. Closer to 95% is better.
5. **Expected calibration error (ECE)** [90]: Covered in Section 2.3.7. Evaluates calibration. This measures the average discrepancy between the predicted confidence intervals (*confidence*) and the empirical coverage of these confidence intervals (*accuracy*). This means it is closely related to the 95% coverage metric. Closer to 0% is better, with 50% being the worst possible score.

In addition to these metrics, figures are used to assist in the evaluation. This includes graphs of predictions (and confidence intervals) versus measurements over time. Additionally: Histograms of prediction errors, correlation plots of predictions versus measurements, and reliability diagrams [90] (covered in Section 2.3.7).

5.6.4 Model interpretation

The models developed for predicting the wind turbine temperature are given in terms of difference vector fields with 4 free inputs. Visualizing these vector fields can help us interpret the models and their differences. The model interpretation consists of a qualitative comparison of the vector fields learned by the models. It is interesting to examine the difference between a linear model and a neural network-based model. For this reason, we consider the bearing temperature steps learned by `CVA-1`, `DE` and `MCD`. Since `CVA-1` has a process noise that is constant, it is not considered when evaluating the uncertainties. We instead look at the uncertainties learned by `DE` and `MCD`, along with the distribution of the training data. The bearing temperature steps correspond to f_θ in Equation (5.14), and the uncertainties correspond to the standard deviation $\sqrt{Q}$ of $w_{\theta,k}$ in Equation (5.14).

To visualize the vector fields, we project them onto two of the input dimensions while keeping the keeping the other inputs fixed to their mean value (computed from the training data). This gives us contours of the *predicted temperature change* and *predictive uncertainty* as a function of two of the inputs. As the bearing tem-

perature is the quantity of interest, we will compare the model predictions as a function of the bearing temperature and each of the other model inputs.

5.7 Results

This section describes the results from the evaluation presented in in Section 5.6. Case A (Simulation) is covered in Section 5.7.1, Case B (Filtering) in Section 5.7.2, and the model interpretation in Section 5.7.3. An overall summary is given in Section 5.7.4.

For both Sections 5.7.1 to 5.7.3, a brief overview of the most important results are first presented. Following this, the results are detailed more thoroughly.

5.7.1 Case A: Simulation

Overview

No single model consistently outperforms all others on all metrics. Though the models with second order dynamics (`BMA` and `CVA-2`) yield the most accurate predictive means for WTUR1, they suffer more than other models when generalizing. The neural network-based models yield more comparable results to `BMA` for the other turbines. However, the calibration results are more one-sided. Overall, `DE` yields more calibrated predictions than the other models.

The steady-stade offset between the temperature in WTUR2 and WTUR1 (see Figure 5.5) is compensated for reasonably well by the 28 day bias correction (described in Section 5.6.1). The large number of missing measurements for WTUR6 results in the models having lower predictive accuracy for it, compared to for WTUR1.

As expected, the anomaly in the measurements from WTUR3 (see Figure 5.5) has a severe impact on the predictions. This means that none of the models yield acceptable predictive means for this turbine. However, there is a single model which yields reasonable estimates for the uncertainty in the prediction: `DE`.

Predictive Accuracy

The mean absolute error, root mean square error, and R^2 of the predictive means are given in Table 5.4a, Table 5.4b, and Table 5.5a.

As a first note, the models in the result tables are ordered similarly to the order they were presented in Section 5.5. The least squares' models are at the top, and the neural network models at the bottom. Consequently, the general trend is that the models on the lower half of the tables yield the best results. Tables 5.4a and 5.4b show the impact the bias term and Hammerstein nonlinearity have on predictive accuracy for the least square models. The inclusions are theoretically sensible, and improve the empirical results. This is visualized well

in Figure 5.13, where we can see that LS-HB has significantly tighter correlation diagrams between measurements and predictions when compared to LS.

Further, we can observe from Table 5.5a that none of the least squares' models outperform *any* of the other models on MAE, RMSE, or R^2 for turbines 1, 2, and 6. Although the differences between LS-HB and CVA-1 are not substantial, they are still consistent. This is not surprising - we would expect a model that is optimized on longer simulation trajectories to yield better results when evaluated on longer simulation trajectories. The distributions of predictive errors shown in Figure 5.12 illustrate this well.

The BMA model yields improved results when compared to both CVA-1 and CVA-2. This supports the well-established view that model averaging can be beneficial for predictive accuracy [27]. The only turbine where this is not consistently the case is WTUR3.

MCD and DE do not outperform CVA-2 and BMA in predictive accuracy for WTUR1. However, the neural-network based models appear to generalize well when compared to other models. The consequence of this is that the neural network models yield predictions of more similar accuracy to BMA for WTUR2 and WTUR6 in terms of RMSE and MAE, as can be seen in Table 5.4. CVA-2 suffers a greater reduction in MAE and RMSE than CVA-1 when applied to WTUR2. This can be seen in Tables 5.4a and 5.4b, and hints that CVA-2 has overfitted on WTUR1 to some degree. However, the effect is not as significant when these models are applied to WTUR6. This indicates that inclusion of higher order dynamics could possibly be beneficial for predictive accuracy for the neural network-based models - though generalization might pose a challenge.

From comparing Tables 5.4a and 5.4b we can observe that the predictions for WTUR6 have significantly higher RMSE than those of WTUR1 and even WTUR2. However, the difference in MAE isn't as substantial. This is an indication that there are more outliers (predictions with significant error) for WTUR6. This can at least partially be explained by the high amount of missing input measurements for WTUR6. It can be seen from the figures that the models tend to predict the temperature in WTUR6 well, but that they at times deviate significantly. These occurrences will often correspond to times where one or more inputs are forward-filled, an example of which can be seen in Figure 5.11.

It can be seen in Table 5.5a that there is still significant temperature variations that are not explained by the models. It is not a significant surprise that none of the models provide useful information for WTUR3: Their negative R^2 values indicate they would be (significantly) outperformed by a model that predicts the mean temperature of WTUR3. However, this is not necessarily a bad result: Considering that the external temperature data in WTUR3 is anomalous (see Figure 5.5), this is a scenario we would be interested in detecting.

Predictive Calibration

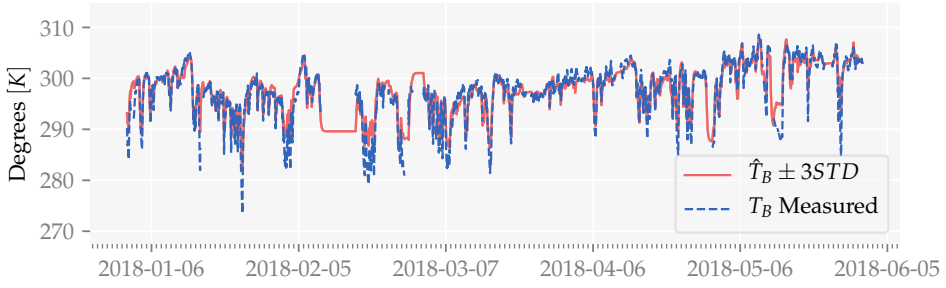
Tables 5.6a and 5.6b show the empirical coverage of 95% confidence intervals and expected calibration error of the model predictions when simulating. The least square's models don't predict confidence intervals, as can be seen in Figures 5.6a and 5.8a. As such, they will not be mentioned further in this subsection.

`DE` yields significantly more calibrated uncertainty estimates than any other model when generalizing to other turbines. The reliability diagrams in Figure 5.14 visualize this very well.

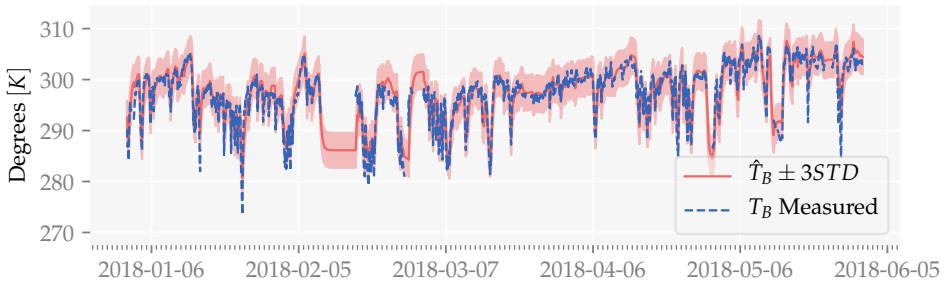
Both `BMA` and `MCD` yield reasonably calibrated predictive uncertainty estimates for WTUR1 - in fact, they yield better estimates than `DE` for WTUR1. This is not unsurprising. The data from WTUR1 used for evaluation occurs chronologically later than the training data, but can be expected to be reasonably similar. Both `BMA` and `MCD` have parameters which are chosen to maximize predictive calibration on the WTUR1 training data under a Gaussian assumption.

Though `BMA` and `MCD` offer decent predictive uncertainty estimates for WTUR1, they do not generalize well to other turbines. They are consistently outperformed by `DE` on WTUR2, WTUR3, and WTUR6. The width of their predicted confidence intervals do not vary much over time or between turbines. On the contrary, `DE` predicts confidence intervals whose width varies significantly - as can be seen in Figures 5.7, 5.9 and 5.10.

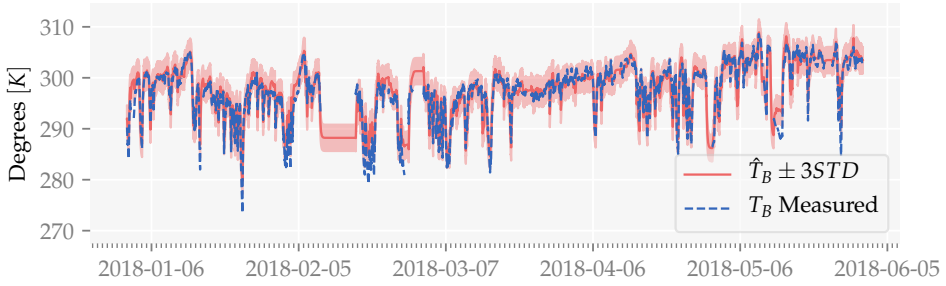
An observation that should be taken into account, is that `DE` is underconfident on WTUR1: It predicts too wide confidence intervals. This can be seen in Table 5.6a. This largely explains the improvement in calibration when `DE` is applied on WTUR2: Similarly to `BMA`, its 95% confidence interval coverage is reduced by approximately 4 percentage points. However, this does not explain the extreme discrepancy in calibration on WTUR3 and WTUR6 between `DE` and *any* other model. The results in Tables 5.6a and 5.6b are a strong indication that `DE` yields predictive uncertainty estimates that generalize better than the comparable approaches used in this thesis. It is noteworthy that this observation is not unique to this experiment [20]. Perhaps equally important, it should be stressed that the uncertainty estimates for `DE` have been observed to be good also for the previous iterations that have been developed during the work for this thesis.



(a) LS-HB

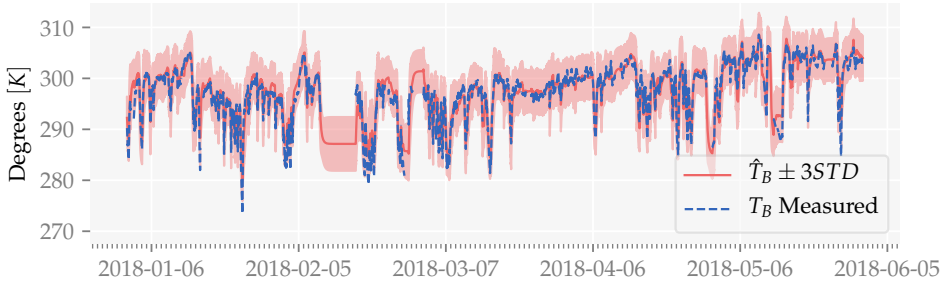


(b) CVA-1

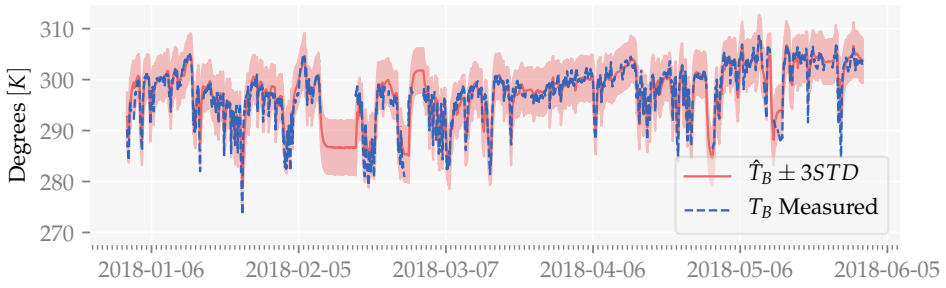


(c) CVA-2

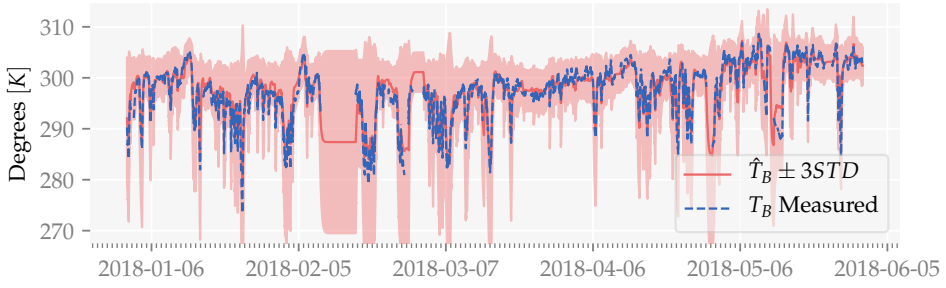
Figure 5.6: Long-Horizon Simulation of Bearing Temperature on WTUR1: LS-HB, CVA-1 and CVA-2: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. This simulation uses data from the turbine the models were trained on, but chronologically later. As in Figure 5.7, the models capture the large variations in bearing temperature but not all the smaller variations (e.g. around $2018-04-06$). Compared to `BMA` in Figure 5.7a, the confidence intervals from `CVA-1` and `CVA-2` are tighter. This is especially evident in areas where input measurements are being forward filled and the prediction converges to an equilibrium (e.g. just after $2018-02-05$).



(a) BMA

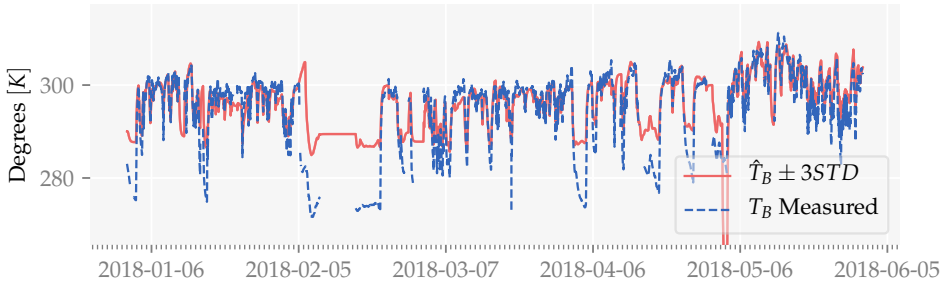


(b) MCD

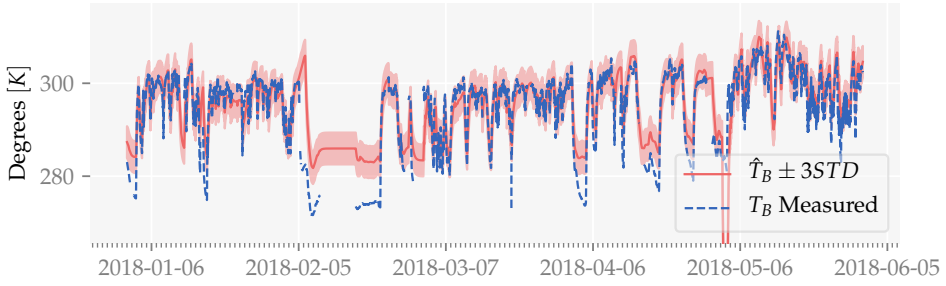


(c) DE

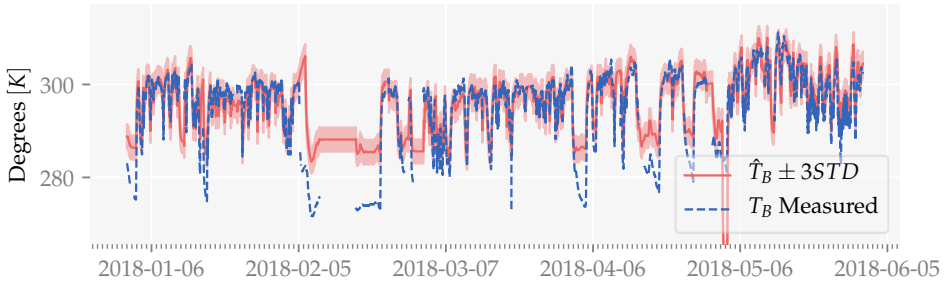
Figure 5.7: Long-Horizon Simulation of Bearing Temperature on WTUR1: BMA, MCD, and DE: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. This simulation uses data from the turbine the models were trained on, but chronologically later. We can recognize the models all capture the large variations in bearing temperature, but not all the smaller variations (e.g. around $2018-04-06$). The confidence intervals from `DE` exhibit more variation over time than those of `BMA` and `MCD`. This is very visible when input measurements are being forward filled and the prediction converges to an equilibrium (e.g. just after $2018-02-05$).



(a) LS-HB

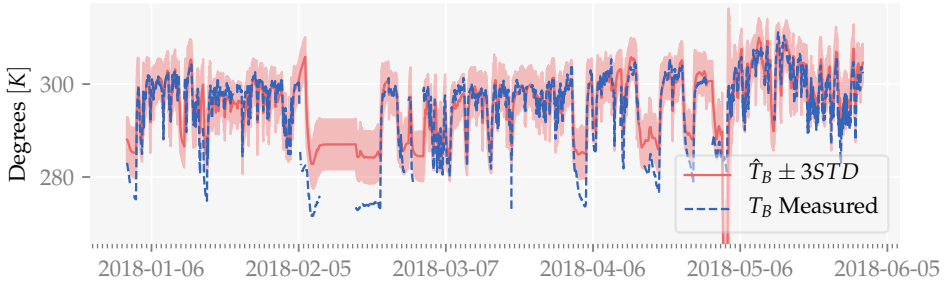


(b) CVA-1

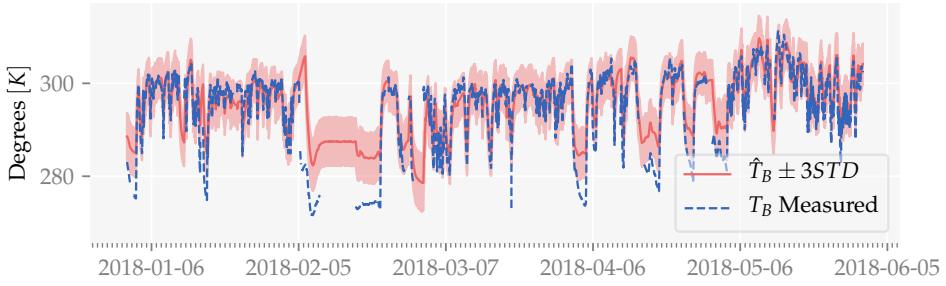


(c) CVA-2

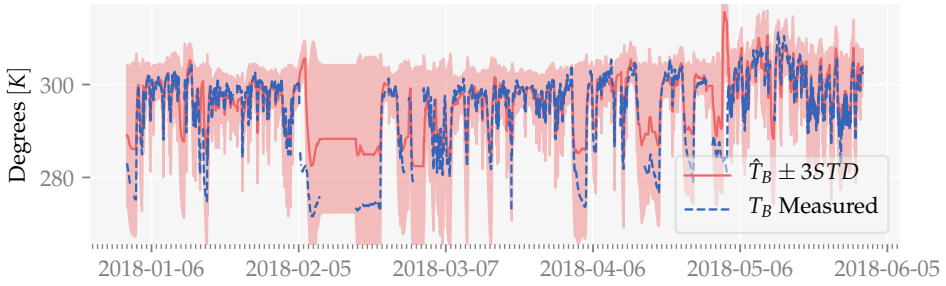
Figure 5.8: Long-Horizon Simulation of Bearing Temperature on WTUR6: LS-HB, CVA-1 and CVA-2: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. These plots are from a simulation on WTUR6, which has significant amounts of missing measurements compared to WTUR1. As in Figure 5.6, the predictions from `CVA-1` and `CVA-2` compared `BMA` (Figure 5.9a) illustrate how model averaging can improve calibration.



(a) BMA

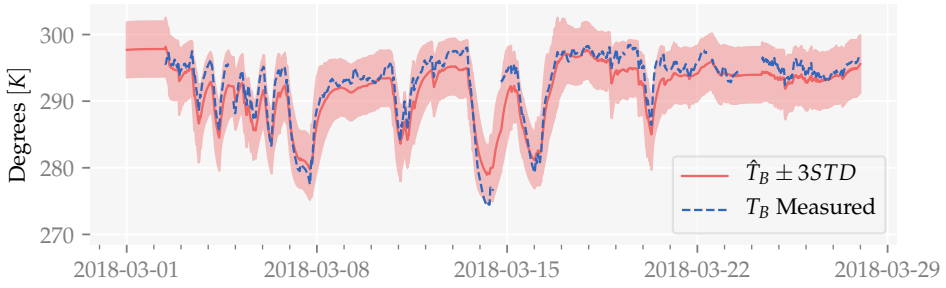


(b) MCD

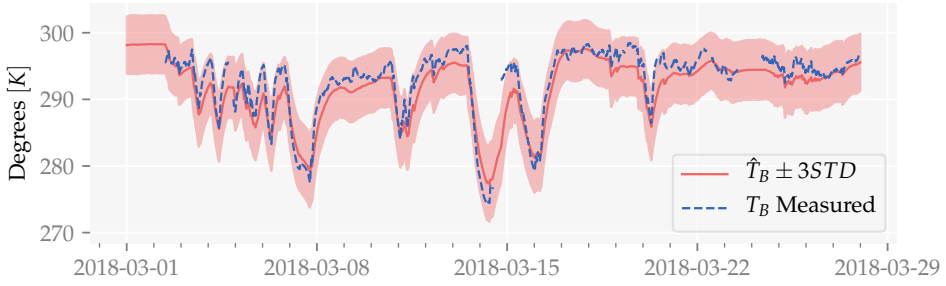


(c) DE

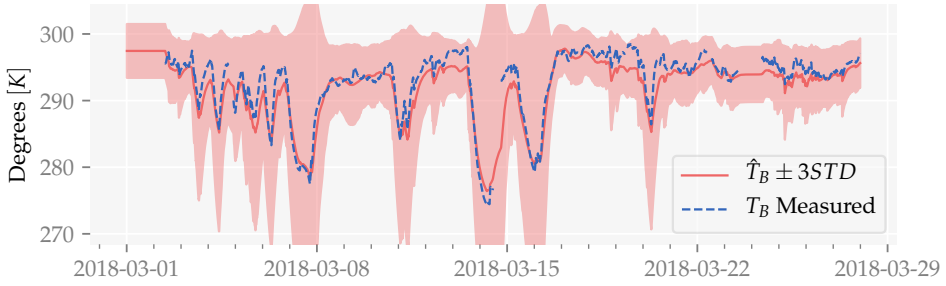
Figure 5.9: Long-Horizon Simulation of Bearing Temperature on WTUR6: BMA, MCD, and DE: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. The plots show a simulation on WTUR6. It can be observed that the width of the confidence intervals predicted by `DE` tend to be greater than they were for WTUR1 in Figure 5.7c. The predictive accuracy of `BMA` and `MCD` appears qualitatively similar to that of `DE`, but they do not provide accurate estimates of their predictive uncertainty.



(a) BMA



(b) MCD



(c) DE

Figure 5.10: Short-Horizon Simulation of Bearing Temperature on WTUR2: BMA, MCD and DE: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. These plots are from a simulation on WTUR2, which has a temperature offset compared to WTUR1. This is compensated for by the bias correction. The large variations in the confidence interval width from `DE` are very visible. In comparison, `BMA` and `MCD` yield qualitatively similar predictions.

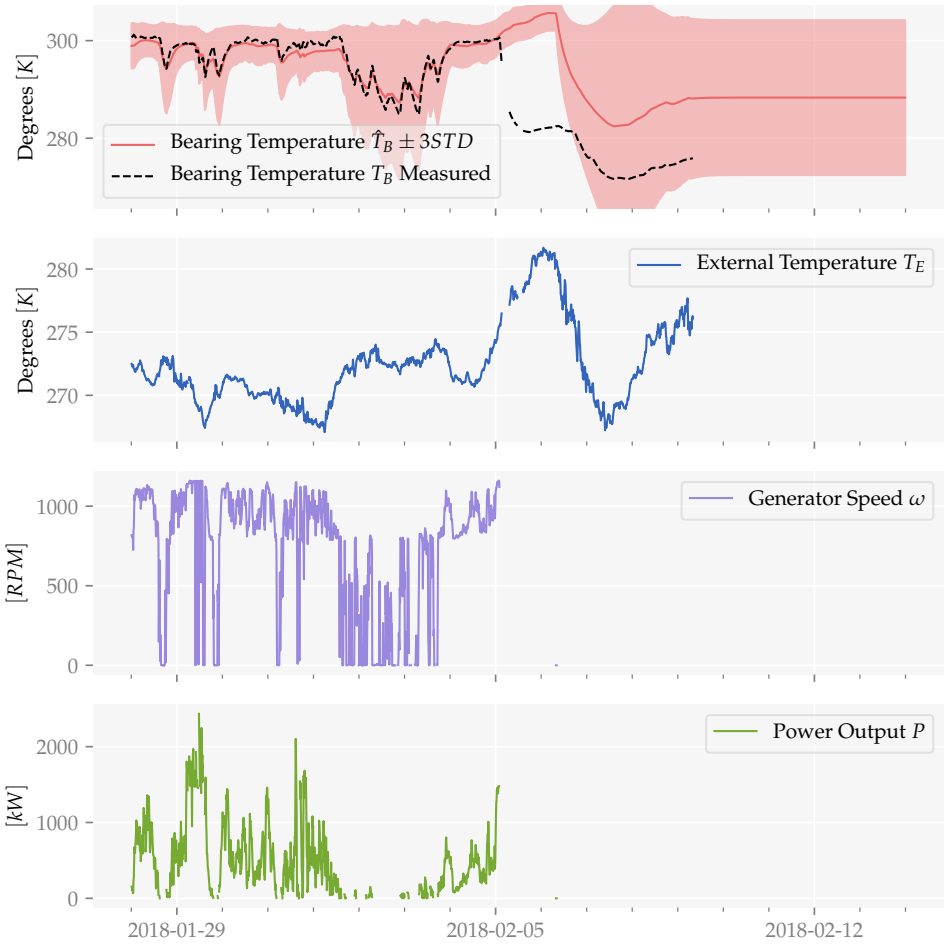


Figure 5.11: Visualizing the Effect of Missing Input Measurements on DE: Measured bearing temperatures (T_B) are given by the dashed *black* lines in the uppermost subplot. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red line and surrounding shaded area. The model inputs T_E, ω, and P are shown in the three lower subplots. This plot is from a simulation on WTUR6 - similarly to in Figure 5.9c, but showing a shorter time span. We can recognize the input measurements are missing for significant parts of the time period shown. Forward filling is used to replace these, resulting in sub-optimal predictive performance.

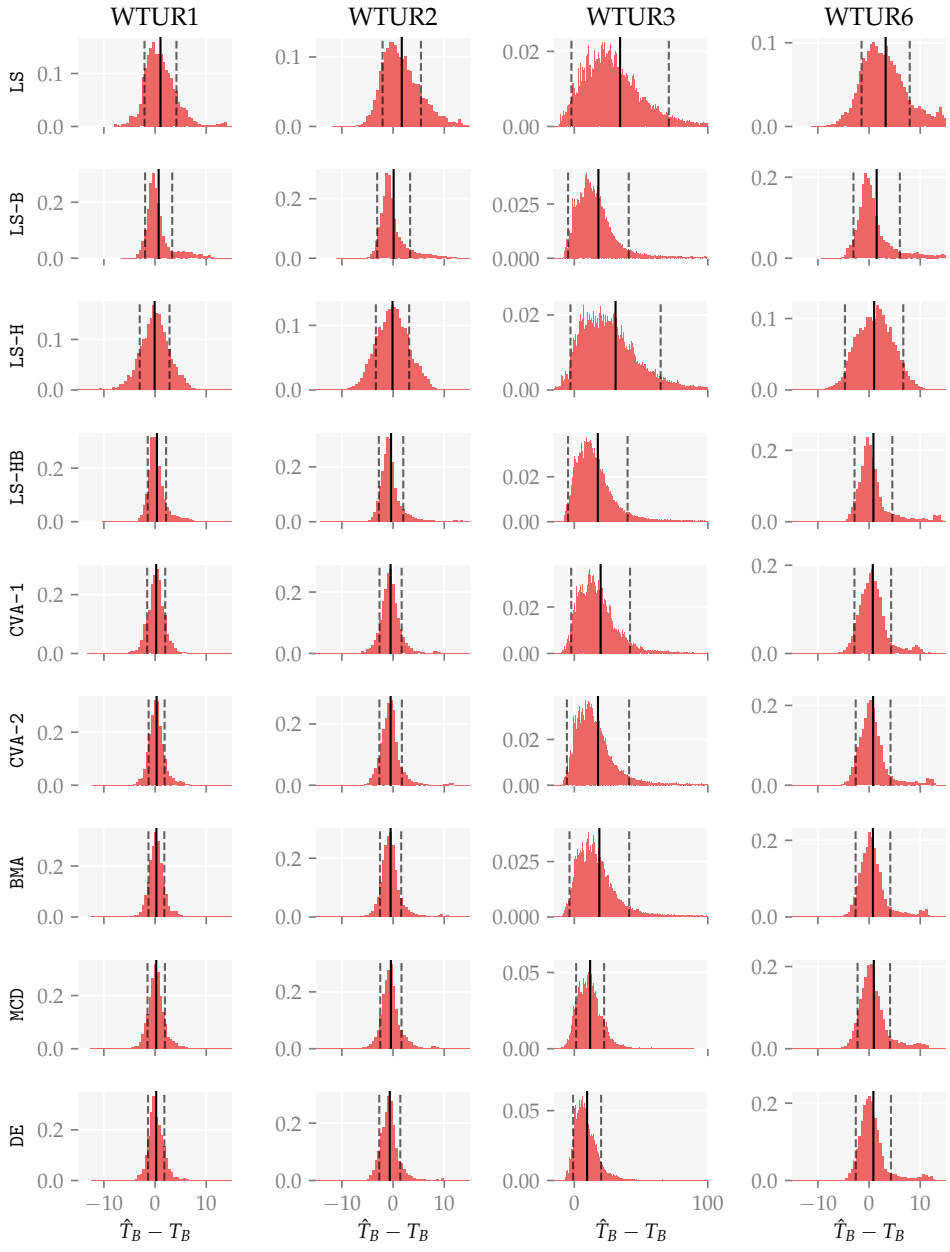


Figure 5.12: Long Horizon Simulations: Prediction Error Marginal Distributions: Histograms showing the marginal distributions of all bias corrected model prediction errors after simulation. Each column corresponds to a turbine, and each row corresponds to a model. Means $\pm 1STD$ are given by the vertical lines. Note the different x-scale for WTUR6. The bias correction compensates well for the bearing temperature offset between WTUR2 and WTUR1. `NN-MCD`, `NN-DE`, `CVA-1`, `CVA-2`, and `BMA` yield approximately equally good predictive accuracy on WTUR2 and WTUR6. The increase in prediction error for WTUR6 compared to WTUR1 is to some degree caused by missing input measurements during simulation. This can not be compensated for by a bias correction.

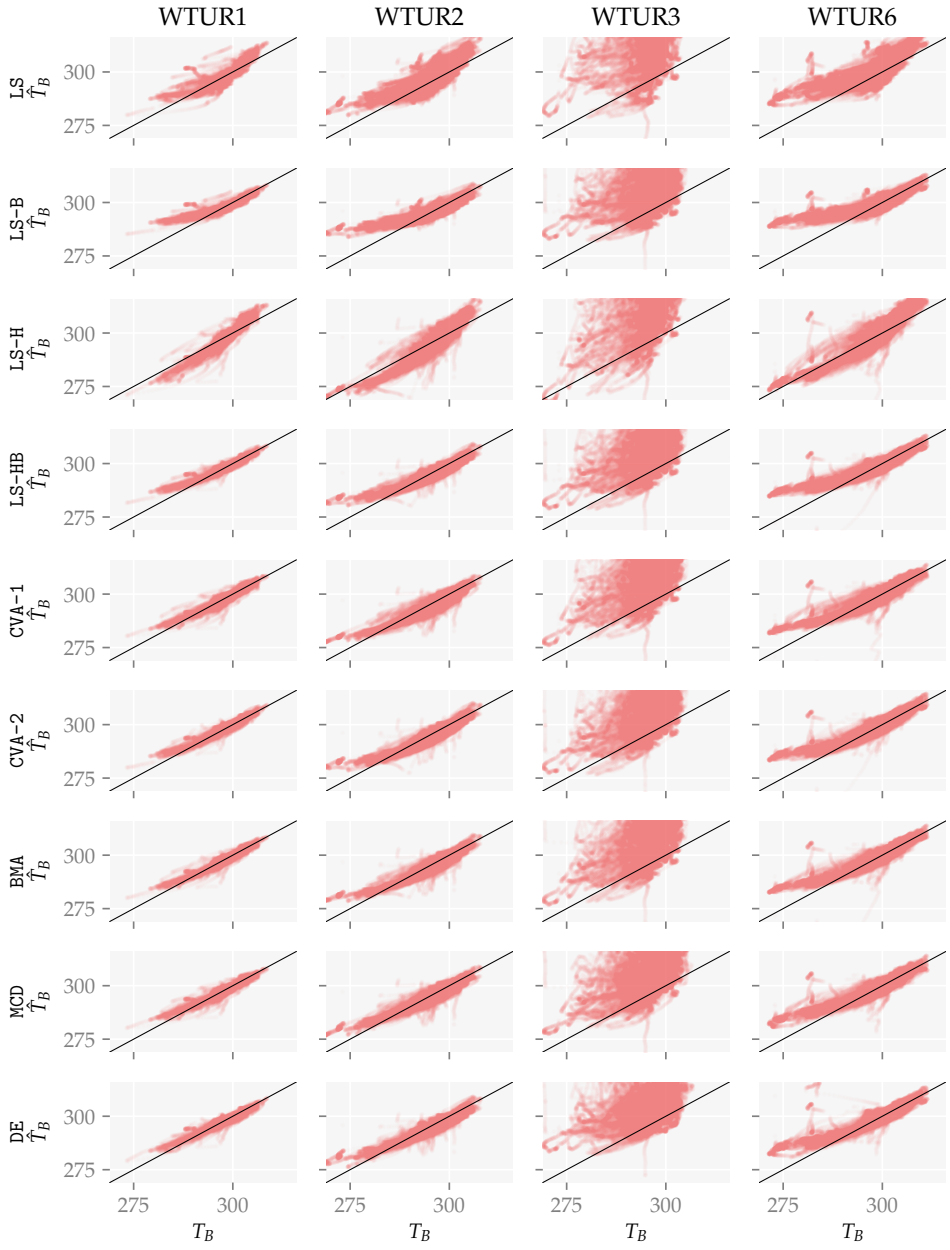


Figure 5.13: Simulation Prediction Correlation Diagrams: Correlation diagrams showing the measured temperature (T_B) versus predicted temperature ($\hat{T}_B$) of simulations. Each column corresponds to a turbine, and each row corresponds to a model. Predictions closer to the solid black diagonal line are better. The predictions for WTUR3 are almost pure noise. For the other turbines, the predictions from the system identification-based and neural network-based models appear qualitatively similar. We can recognize from the scatters that models tend to be biased at lower bearing temperatures for WTUR1, WTUR2 and WTUR6. Interestingly, `LS-H` is the only model which does not exhibit this behaviour - though its predictions on average appear to be far more noisy than most other models.

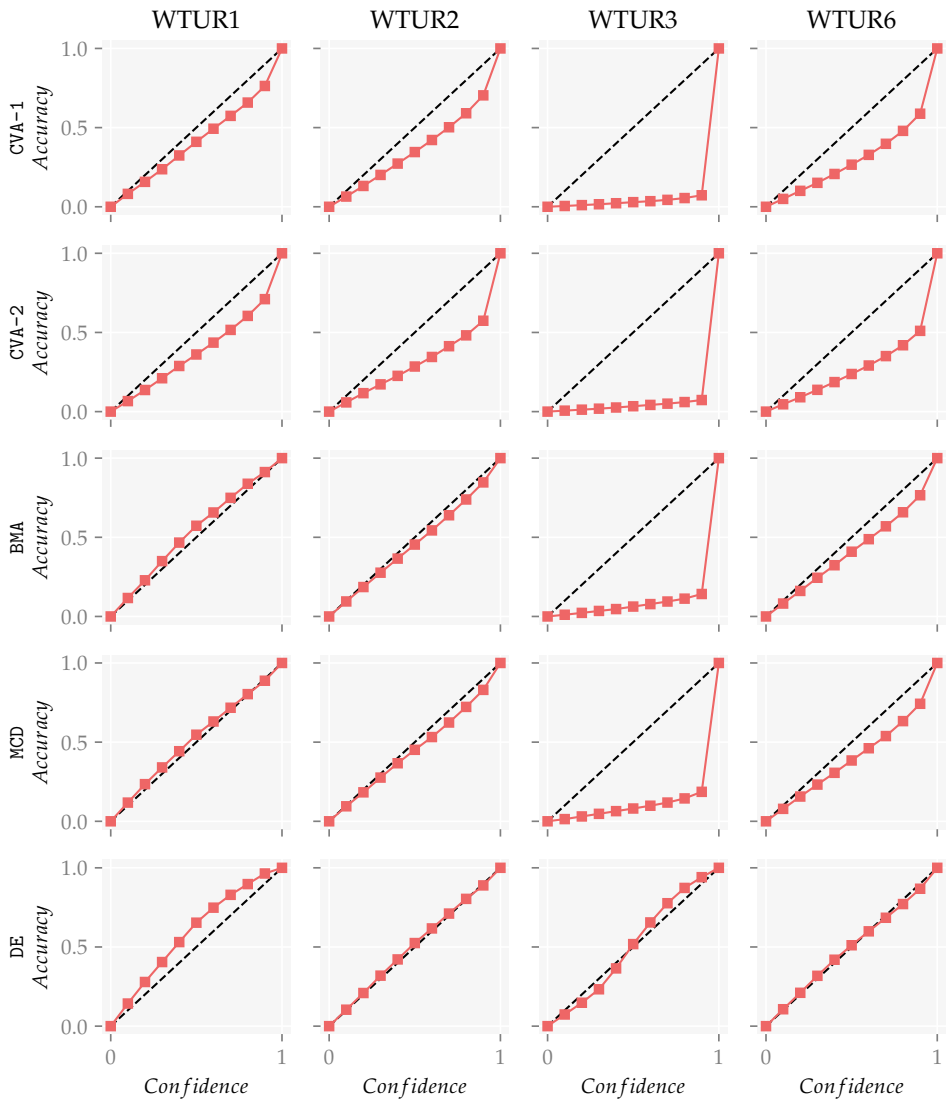


Figure 5.14: Simulation Regression Reliability Diagrams: Regression reliability diagrams showing predicted confidence interval coverage (*Confidence*) versus empirical coverage of the intervals (*Accuracy*). Each column corresponds to a turbine, and each row corresponds to a model. The model calibration curves are given by the solid red line. The markers indicate where the coverage has been evaluated. The coverage is given in the range $[0, 1]$, where 0 denotes no coverage, and 1 denotes full (100%) coverage. The dashed black lines indicate optimal calibration - closer to this is better. Markers *below* the dashed black line indicate the model is overconfident. Markers *above* the dashed black line indicate the model is underconfident. Both `BMA`, `DE`, and `MCD` are underconfident on WTUR1 - most of all `DE`. However, `DE` is significantly better calibrated than any other model on WTUR2, WTUR3, and WTUR6.

Model	WTUR1	WTUR2	WTUR3	WTUR6
LS	2.3700	3.0749	34.7823	4.2531
LS-B	1.8517	2.0986	18.5188	2.8331
LS-H	2.2594	2.5561	31.4240	3.2679
LS-HB	1.2976	1.6540	18.0648	2.2333
CVA1	1.2956	1.5818	20.0866	2.2472
CVA2	1.1559	1.5638	18.2199	2.1464
BMA	1.1399	1.4978	19.1556	2.0900
MCD	1.2461	1.5194	12.2560	2.1373
DE	1.2705	1.5740	10.0346	2.0612

(a) Mean Absolute Error (MAE) [K]

Model	WTUR1	WTUR2	WTUR3	WTUR6
LS	10.3387	16.9514	2505.7783	32.6566
LS-B	7.0183	10.3568	841.1924	22.7610
LS-H	8.5632	10.5939	2094.6558	33.4457
LS-HB	3.2180	5.8309	806.6307	14.3540
CVA1	3.1443	4.9184	873.0285	13.3859
CVA2	2.4526	5.0571	856.6162	12.2108
BMA	2.4225	4.5268	850.4112	12.0720
MCD	2.9249	4.4870	249.7213	10.9293
DE	2.6833	4.6343	201.6541	12.5213

(b) Root Mean Square Error (RMSE) [K]

Table 5.4: Wind Turbine Simulation - MAE and RMSE: Results from Section 5.7.1 (Case A: Simulation).

Model	WTUR1	WTUR2	WTUR3	WTUR6
LS	0.6371	0.5180	−75.6632	0.4111
LS-B	0.7536	0.7055	−24.7359	0.5896
LS-H	0.6994	0.6988	−63.0851	0.3969
LS-HB	0.8870	0.8342	−23.6785	0.7412
CVA1	0.8896	0.8602	−25.7099	0.7586
CVA2	0.9139	0.8562	−25.2078	0.7798
BMA	0.9150	0.8713	−25.0180	0.7823
MCD	0.8973	0.8724	−6.6401	0.8029
DE	0.9058	0.8682	−5.1695	0.7742

(a) R^2 (Predictions)

Table 5.5: Wind Turbine Simulation - R^2: Results from Section 5.7.1 (Case A: Simulation)

Model	WTUR1	WTUR2	WTUR3	WTUR6
LS	0.00 %	0.00 %	0.00 %	0.00 %
LS-B	0.00 %	0.00 %	0.00 %	0.00 %
LS-H	0.00 %	0.00 %	0.00 %	0.00 %
LS-HB	0.00 %	0.00 %	0.00 %	0.00 %
CVA1	82.79 %	77.85 %	8.59 %	67.42 %
CVA2	77.60 %	64.79 %	8.38 %	58.50 %
BMA	94.49 %	90.23 %	17.13 %	82.64 %
MCD	93.01 %	88.46 %	22.22 %	80.72 %
DE	97.83 %	93.88 %	96.76 %	93.08 %

(a) Empirical Coverage of Predicted 95% Confidence Intervals

Model	WTUR1	WTUR2	WTUR3	WTUR6
LS	50.00 %	50.00 %	50.00 %	50.00 %
LS-B	50.00 %	50.00 %	50.00 %	50.00 %
LS-H	50.00 %	50.00 %	50.00 %	50.00 %
LS-HB	50.00 %	50.00 %	50.00 %	50.00 %
CVA1	6.85 %	11.52 %	38.28 %	17.57 %
CVA2	10.98 %	16.62 %	37.97 %	20.28 %
BMA	3.38 %	3.23 %	35.41 %	7.27 %
MCD	1.74 %	3.82 %	33.77 %	8.80 %
DE	4.85 %	1.09 %	4.03 %	1.30 %

(b) Expected Calibration Error (ECE)

Table 5.6: Wind Turbine Simulation - Prediction Coverage and ECE: Results from Section 5.7.1 (Case A: Simulation). The calibration curve of the models are illustrated in Figure 5.14.

5.7.2 Case B: Filtering

Overview

The predictive accuracy in this experiment is notably increased compared to Section 3.2.4, which is expected. However, few models consistently outperform the naive model. Overall, `MCD`, `LS-H` and `LS` yield the most accurate predictions. These outperform the naive model in RMSE on all turbines except WTUR3.

Just as in Section 3.2.4, the anomaly in the measurements from WTUR3 (see Figure 5.5) still has a severe impact on the predictions. This means that none of the models yield acceptable predictive means for this turbine.

Contrary to in Section 5.7.1, none of the models yield calibrated uncertainty estimates. `CVA-1` and `CVA-2` yield estimates than the other models, but it is not consistent. The trend is that the models are under-confident - which is reasonable, considering they are optimized for simulation rather than filtering.

Predictive Accuracy

The mean absolute error, root mean square error, and R^2 of the steps of the predictive means are given in Table 5.7a, Table 5.7b, and Table 5.8a.

Whereas the results in Section 3.2.4 were largely correlated with the order the models were developed in, this is not the case here. Perhaps most importantly, we can observe in Tables 5.7a and 5.7b that the least squares' models tend to be more accurate compared to `CVA-1` and `CVA-2`. Considering that the former are optimized for single-step predictions and the latter are optimized for multi-step predictions, this is not surprising. In practice, it yields a similar observation to as in Section 5.7.1 - that the optimization criteria should be chosen based on the context we wish to apply the model in.

There is a clear qualitative difference between the leas squares' models and the rest, which can be seen for `LS-H` in Figures 5.15 and 5.19: As `LS-H` predicts the next temperature based on the previous measured bearing temperature, it does not update an internal state when measurements are missing. As a consequence, `LS-H` yields (nearly) similar predictions for long time periods when no measurements are available.

Both `LS`, `LS-H`, and `MCD` yield good results. We can see in Table 5.7b that they consistently outperform the naive model in terms of RMSE. Figure 5.21 also illustrates that they have significantly more accurate predictions than `CVA-1` and `CVA-2`. `DE` generalizes worse than `MCD` when filtering. Considering the size-difference between the networks, this points at `DE` having overfitted on `WTUR1`.

`LS-H` is the model most similar to what is used by [30] for one-step-ahead predictions for wind turbine bearing temperature. The results here indicate that it is a good choice for that purpose. However, their reported RMSE for one-step-ahead

predictions is lower than what all of the models evaluated here (except CVA-2) result in. This can be indicative that the sensors in the wind turbines they consider have a higher noise output or are less accurate. It might also be due to slight differences in the input features used.

Figures 5.19 and 5.20 provide an illustration of just how significant the measurement anomaly in WTUR3 is. Despite filtering, none of the models yield good predictions over time on WTUR3. Interestingly, the *lack* of internal state update in LS-H when measurements are missing proves beneficial in this case. This can for instance be seen in in Figures 5.19a and 5.19b, just after 2018-02-05. LS-H predicts a temperature close to the previous measurements whereas CVA-1 predicts a rapid increase in bearing temperature once no measurement updates are available.

We can recognize from Table 5.8a that the models offer far less information for this case in comparison to Section 5.7.1, where no bearing temperature measurements were available. This is not a very large surprise. The regressors used by the models are *indicative* of the direction in which the temperature will change. However, when the temperature is close to an equilibrium, the unmodeled phenomena will govern the fine-grained details in the temperature evolution. As can be seen from Figure 5.22, the predicted temperature steps do tend to be correlated with the measured temperature steps - but not perfectly.

A common challenge with one-step-ahead predictors is that they behave like a naive model (i.e. predict the previous value). This will give a prediction time series which is mostly identical to the measurement time series, but shifted a single time step. Figures 5.17 and 5.18 are included to examine this.

Judging by the (admittedly very short) time span shown there, LS-H, MCD, and DE appear to to provide more informative predictions than CVA-1 and CVA-2. This corresponds well with their RMSE values in Table 5.7b. Interestingly, the R^2 values of CVA-1's steps in Table 5.8a are similar to those of MCD, and better than those of DE. This likely ties into the uncertainties in the model predictions, which can be seen in Figures 5.15b, 5.15c, 5.16a and 5.16b. Higher predictive uncertainty in the neural network predictions will result in the Kalman filter compensating more. This means the inner state of the neural network models are likely closer to the previous measured bearing temperature after the Kalman update than CVA-1 is. As a consequence, DE yields lower RMSE than CVA-1 despite not predicting as informative steps.

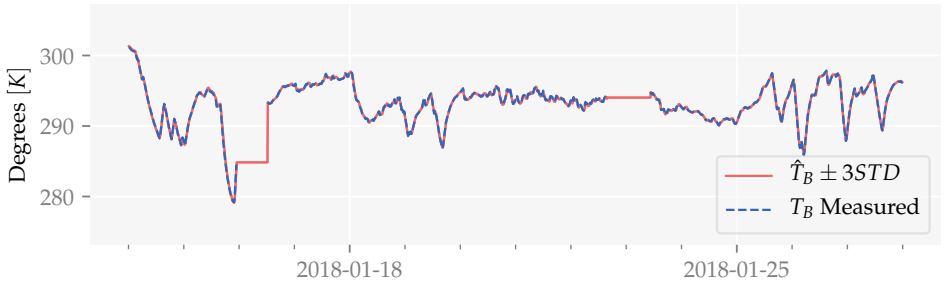
Predictive Calibration

Tables 5.9a and 5.9b show the models' 95% confidence interval coverage and expected calibration error.

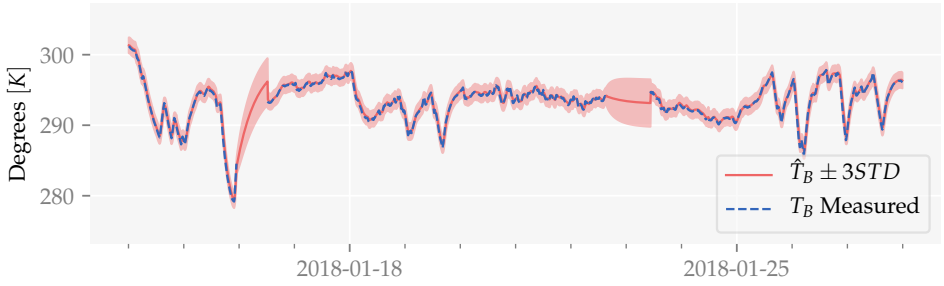
Compared to in Section 3.2.4, the models are worse calibrated. Looking at Figure 5.23, we can recognize the confidence intervals predicted by DE and MCD during filtering are not of any use. This is the case for CVA-1 and CVA-2 as well.

Though `CVA-2` is reasonably calibrated on WTUR2, it does not generalize between the turbines. This can be seen in Table 5.9b.

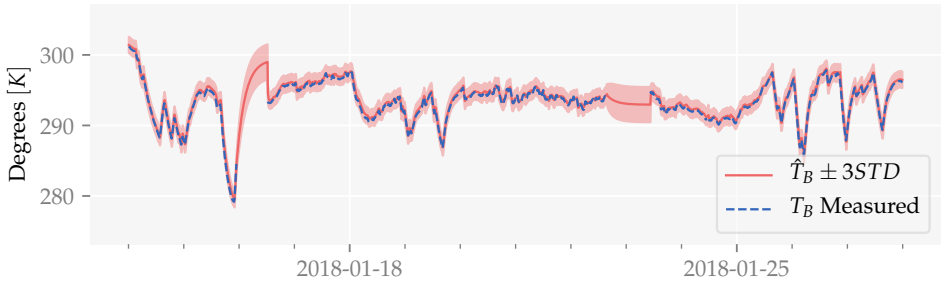
The lack of calibration can be explained by the models being optimized for simulation rather than filtering: The process uncertainties from all models were found based on their simulation performance. Just like the least squares' models that are optimized on single step predictions perform suboptimally in Case A (Section 5.7.1), the models optimized on longer trajectories tend to give supoptimal predictions here.



(a) LS-H

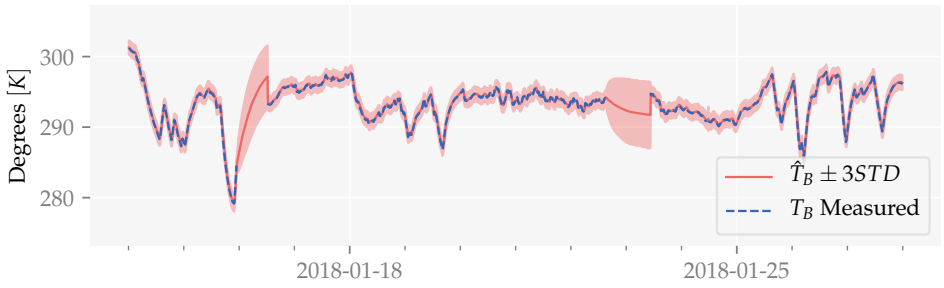


(b) CVA-1

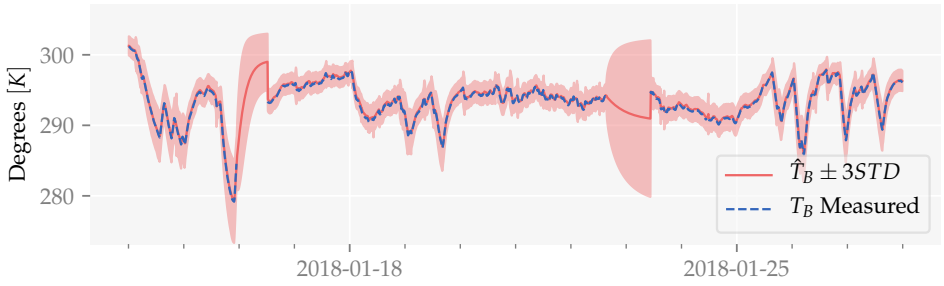


(c) CVA-2

Figure 5.15: Short-Horizon Filtering of Bearing Temperature on WTUR2: LS-H, CVA-1 and CVA-2: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. These plots are from a filtering on WTUR2, which has a temperature offset compared to WTUR1. No bias correction is performed.

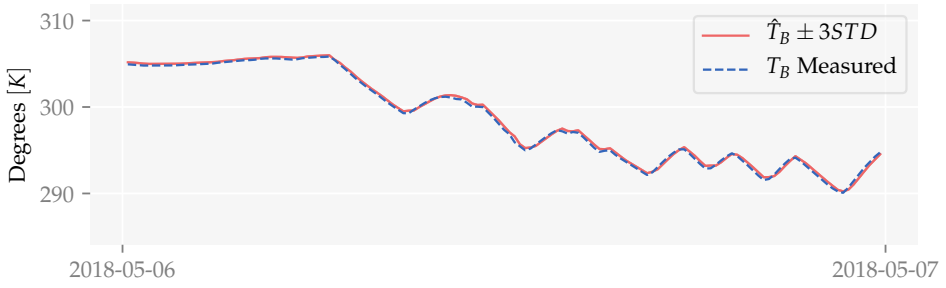


(a) MCD

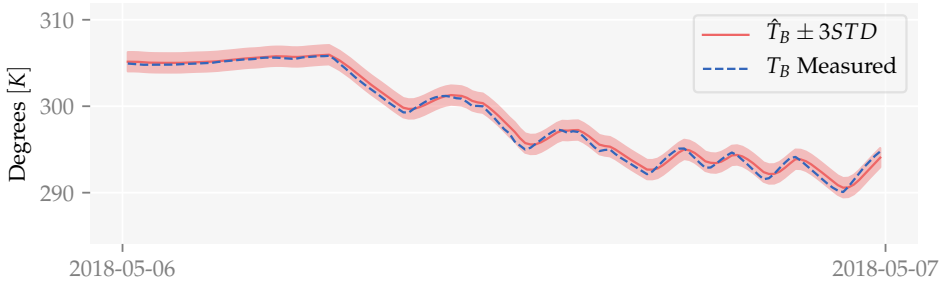


(b) DE

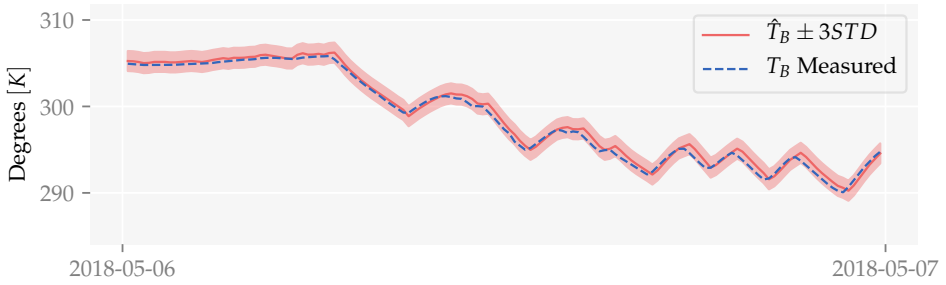
Figure 5.16: Short-Horizon Filtering of Bearing Temperature on WTUR2: MCD and DE: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. These plots are from a filtering on WTUR2, which has a temperature offset compared to WTUR1. No bias correction is performed.



(a) LS-H

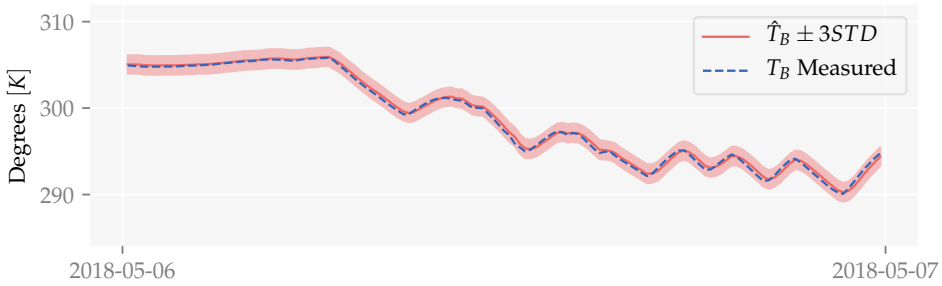


(b) CVA-1

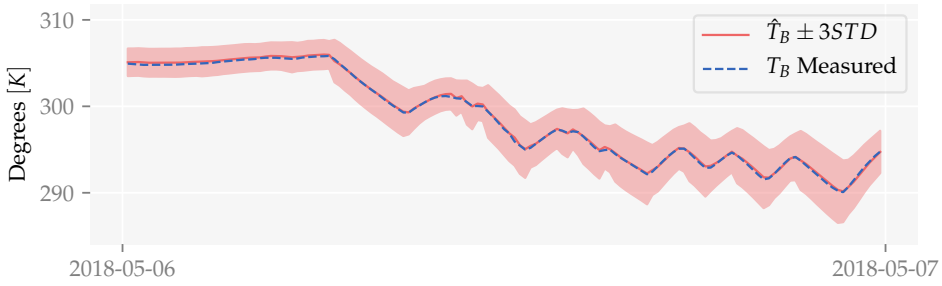


(c) CVA-2

Figure 5.17: Very Short-Horizon Filtering of Bearing Temperature on WTUR6: LS-H, CVA-1 and CVA-2: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. These plots are from a filtering on WTUR6, which has significant amount of missing measurements. No bias correction is performed.

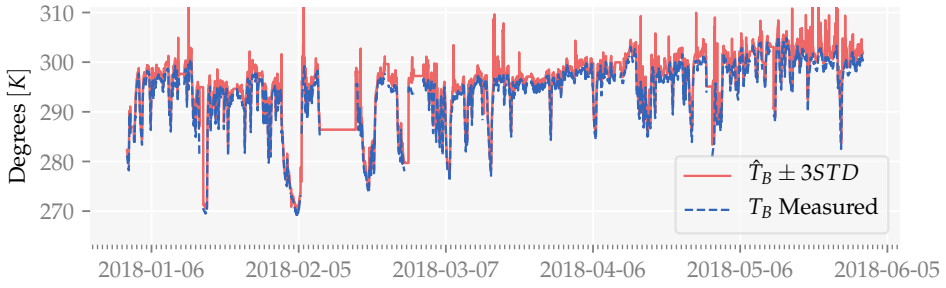


(a) MCD

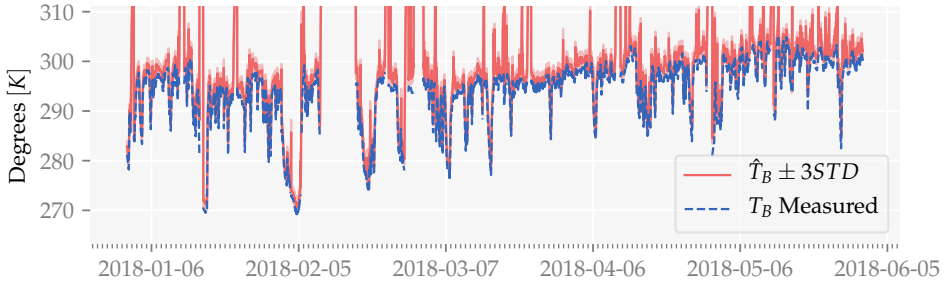


(b) DE

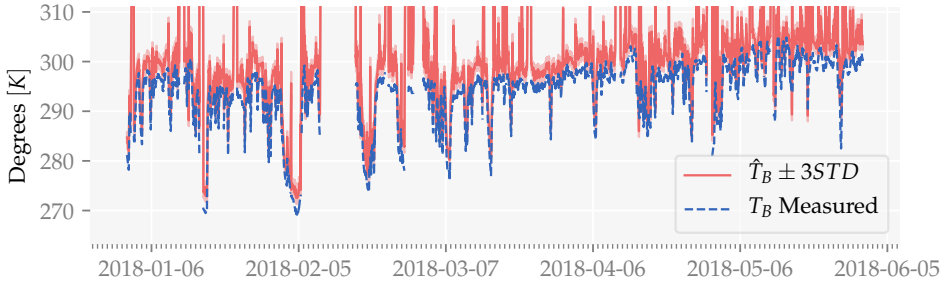
Figure 5.18: Very Short-Horizon Filtering of Bearing Temperature on WTUR6: MCD and DE: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. These plots are from a filtering on WTUR6, which has significant amount of missing measurements. No bias correction is performed.



(a) LS-H

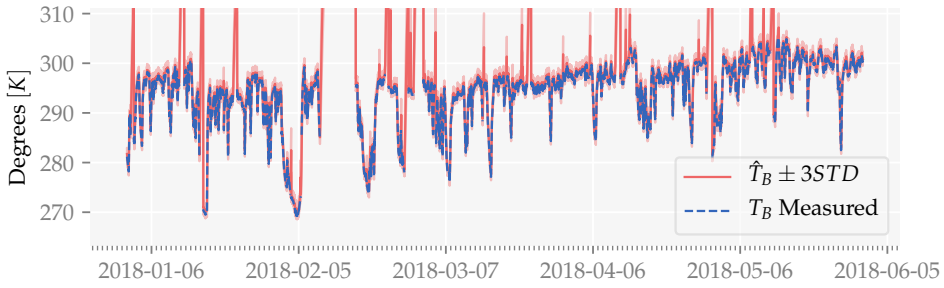


(b) CVA-1

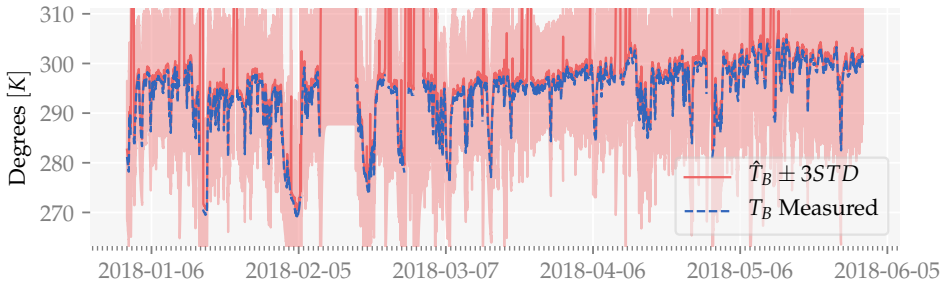


(c) CVA-2

Figure 5.19: Long-Horizon Filtering of Bearing Temperature on WTUR3: LS-H, CVA-1 and CVA-2: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. These plots are from a filtering on WTUR3, which has anomalous external temperature measurements. No bias correction is performed.



(a) MCD



(b) DE

Figure 5.20: Long-Horizon Filtering of Bearing Temperature on WTUR3: MCD and DE: Measured bearing temperatures (T_B) are given by the dashed blue lines. Predicted bearing temperatures ($\hat{T}_B$) along with $\pm 3STD$ confidence intervals are given by the solid red lines and surrounding shaded area. These plots are from a filtering on WTUR3, which has anomalous external temperature measurements. No bias correction is performed.

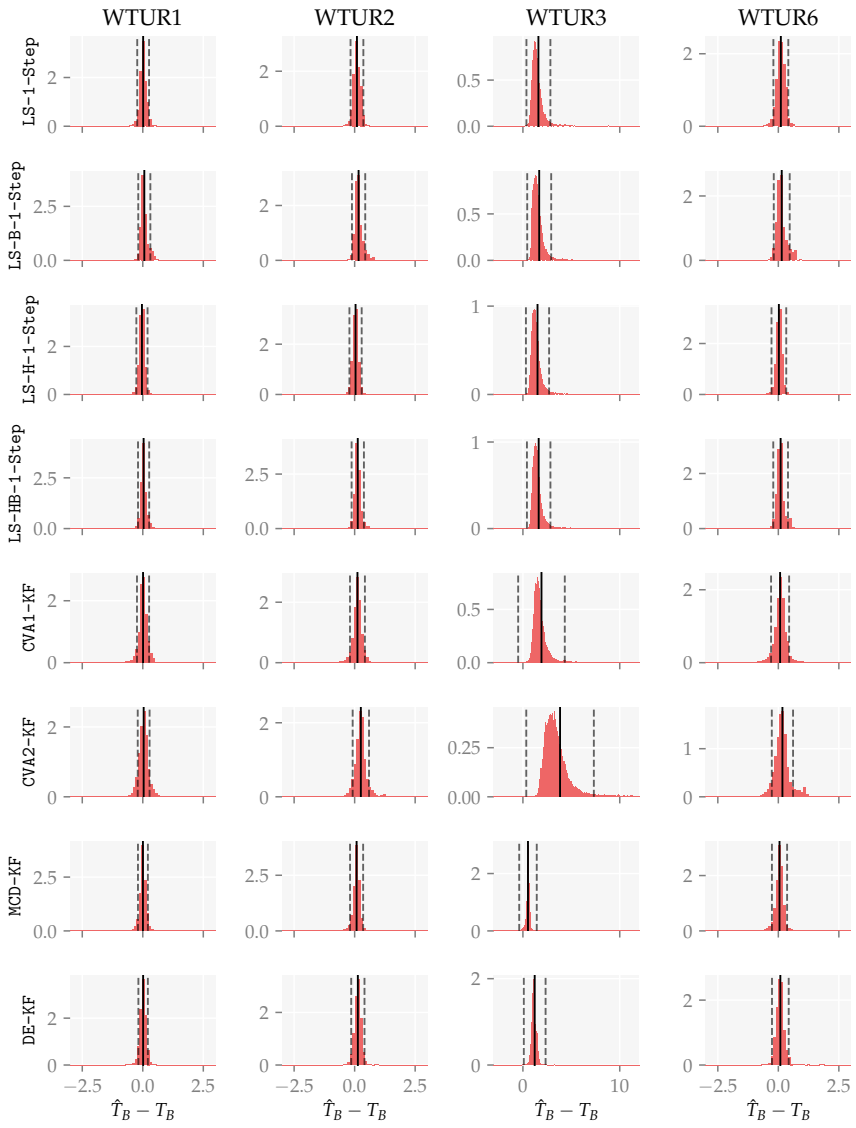


Figure 5.21: One-Step-Ahead Filtering Errors: Histograms showing the marginal distributions of the model one-step-ahead prediction errors when filtering. This is the scenario from Case B Each column corresponds to a turbine, and each row corresponds to a model. Means $\pm 1STD$ are given by the vertical lines. Compared to in Section 5.7.1 the predictive errors when filtering are much more densely distributed - as is to be expected.

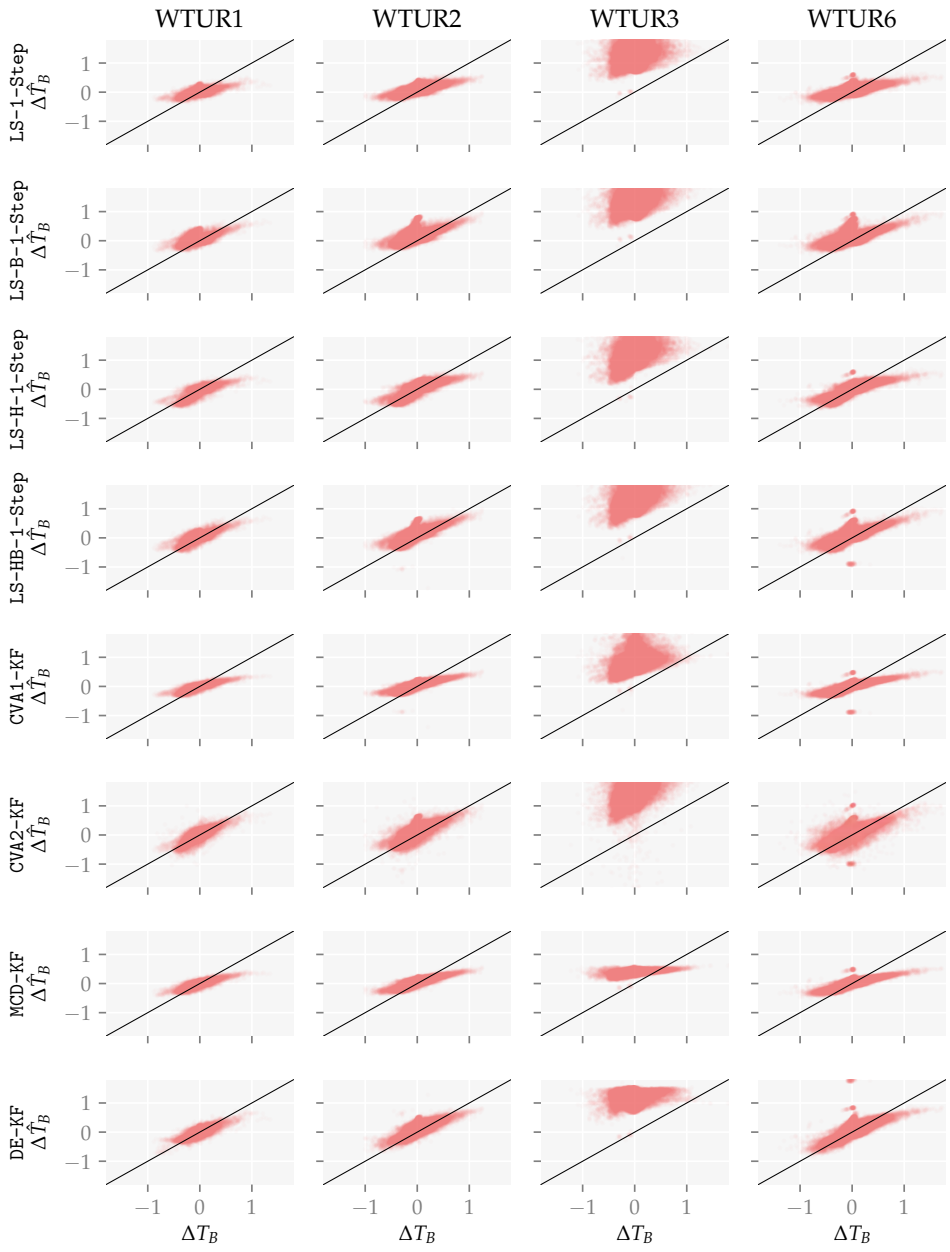


Figure 5.22: Filtering Step Correlation Diagrams: Correlation diagrams showing the steps in measured temperature (ΔT_B) versus predicted steps ($\Delta \hat{T}_B$) of one-step-ahead predictions. Each column corresponds to a turbine, and each row corresponds to a model. Predictions closer to the solid black diagonal line are better. The models that yield the best results in Table 5.7 appear to have the least vertical spread in the correlation plots here. Remark that whereas Figure 5.13 shows the correlation between predictions, this figure shows the correlation in the steps. Interestingly, the predictions from `CVA-2` and `DE` appear better correlated than `MCD` for edge-cases, but have significantly more vertical spread on average.

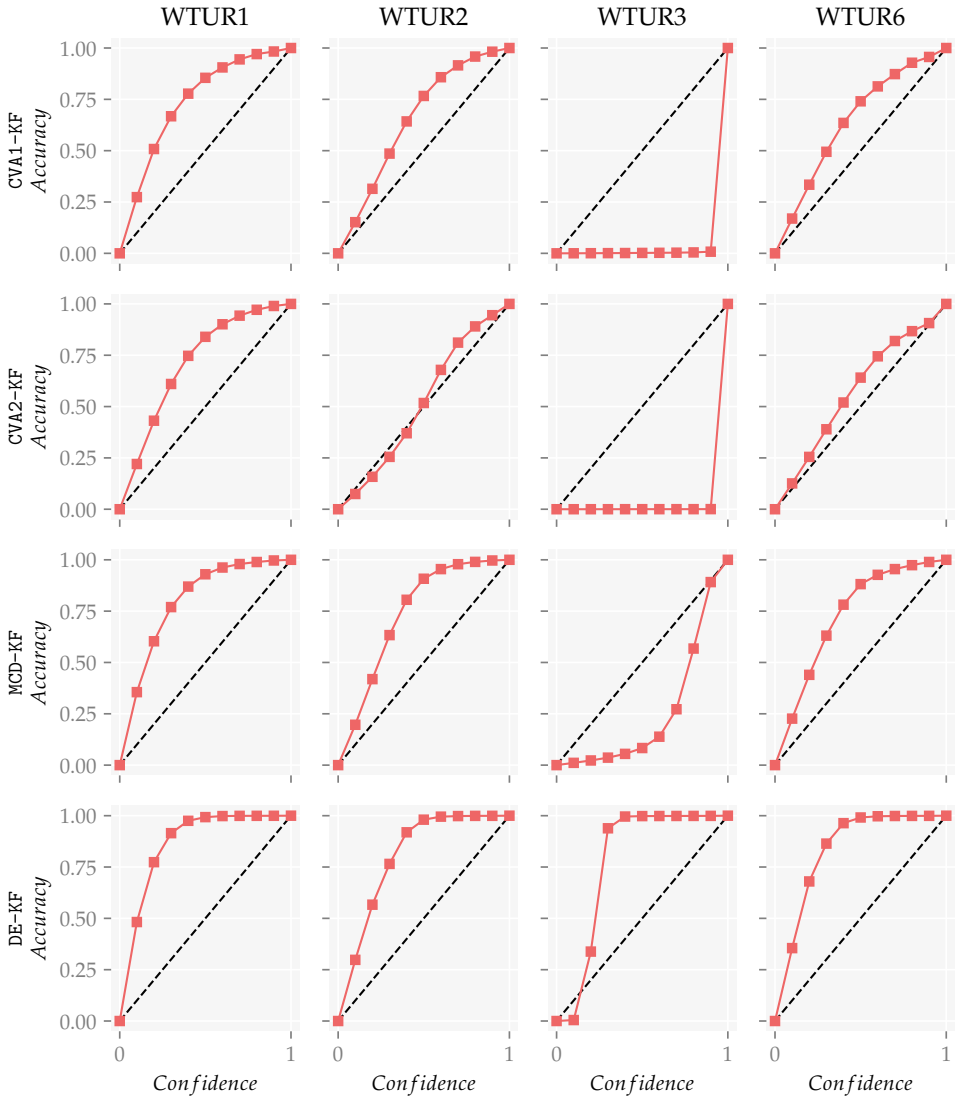


Figure 5.23: Filtering Regression Reliability Diagrams: Regression reliability diagrams showing predicted confidence interval coverage (*Confidence*) versus empirical coverage of the intervals (*Accuracy*) of one-step-ahead predictions. Each column corresponds to a turbine, and each row corresponds to a model. The model calibration curves are given by the solid red line. The markers indicate where the coverage has been evaluated. The coverage is given in the range $[0, 1]$, where 0 denotes no coverage, and 1 denotes full (100%) coverage. The dashed black lines indicate optimal calibration - closer to this is better. Markers *below* the dashed black line indicate the model is overconfident. Markers *above* the dashed black line indicate the model is underconfident. In comparison to Figure 5.14, `DE` and `MCD` yield far worse calibration when filtering. The models tend to yield underconfident predictions in this case - aside from on WTUR3. Interestingly, `CVA-1` and `CVA-2` yield better calibrated predictive uncertainty than `DE` and `MCD`. This is not very surprising, considering they predict the tightest confidence intervals in Section 5.7.1, which can e.g. be seen in Figures 5.8b and 5.8c. Still, the calibration is not nearly as good as `DE` has in Section 5.7.1.

Model	WTUR1	WTUR2	WTUR3	WTUR6
Naive	0.1159	0.1207	0.1243	0.1420
LS-1-Step	0.1076	0.1408	1.6047	0.1680
LS-B-1-Step	0.1167	0.1814	1.6854	0.1870
LS-H-1-Step	0.0930	0.1010	1.5123	0.1131
LS-HB-1-Step	0.0926	0.1484	1.6315	0.1455
CVA1-KF	0.1380	0.1843	1.9171	0.2049
CVA2-KF	0.1537	0.2877	3.8386	0.2767
MCD-KF	0.0980	0.1287	0.5304	0.1381
DE-KF	0.0960	0.1594	1.2162	0.1481

(a) Mean Absolute Error (MAE) [K]

Model	WTUR1	WTUR2	WTUR3	WTUR6
Naive	0.0703	0.0808	0.0735	0.1095
LS-1-Step	0.0598	0.0793	4.0946	0.1058
LS-B-1-Step	0.0664	0.1019	4.3632	0.1282
LS-H-1-Step	0.0548	0.0655	3.7073	0.0932
LS-HB-1-Step	0.0543	0.0813	4.1237	0.1014
CVA1-KF	0.0629	0.1082	9.4713	0.1444
CVA2-KF	0.0612	0.1797	26.8938	0.2215
MCD-KF	0.0420	0.0796	1.0938	0.1008
DE-KF	0.0384	0.0929	2.7443	0.1260

(b) Root Mean Square Error (RMSE) [K]

Table 5.7: Wind Turbine Filtering - MAE and RMSE: Results from Section 5.7.2 (Case B: Filtering).

Model	WTUR1	WTUR2	WTUR3	WTUR6
Naive	0.0000	0.0000	0.0000	0.0000
LS-1-Step	0.3215	0.0181	−122.5148	0.0650
LS-B-1-Step	0.0659	−0.4823	−130.7144	−0.4222
LS-H-1-Step	0.4891	0.3101	−110.7246	0.3239
LS-HB-1-Step	0.4847	−0.0413	−123.4234	0.1395
CVA1-KF	0.5706	0.3173	−39.3820	0.4715
CVA2-KF	0.4586	−0.0894	−168.0358	0.0225
MCD-KF	0.5679	0.3078	−5.0804	0.5032
DE-KF	0.5623	0.0386	−42.5079	0.2320

(a) R^2 (Steps)

Table 5.8: Wind Turbine Filtering - R^2 of Predicted Steps: Results from Section 5.7.2 (Case B: Filtering).

Model	WTUR1	WTUR2	WTUR3	WTUR6
Naive	0.00 %	0.00 %	0.00 %	0.00 %
LS-1-Step	0.00 %	0.00 %	0.00 %	0.00 %
LS-B-1-Step	0.00 %	0.00 %	0.00 %	0.00 %
LS-H-1-Step	0.00 %	0.00 %	0.00 %	0.00 %
LS-HB-1-Step	0.00 %	0.00 %	0.00 %	0.00 %
CVA1-KF	98.95 %	98.88 %	1.36 %	97.02 %
CVA2-KF	99.54 %	96.72 %	0.07 %	92.87 %
MCD-KF	99.84 %	99.84 %	96.78 %	99.47 %
DE-KF	99.97 %	99.95 %	99.93 %	99.94 %

(a) Empirical Coverage of Predicted 95% Confidence Intervals

Model	WTUR1	WTUR2	WTUR3	WTUR6
Naive	50.00 %	50.00 %	50.00 %	50.00 %
LS-1-Step	50.00 %	50.00 %	50.00 %	50.00 %
LS-B-1-Step	50.00 %	50.00 %	50.00 %	50.00 %
LS-H-1-Step	50.00 %	50.00 %	50.00 %	50.00 %
LS-HB-1-Step	50.00 %	50.00 %	50.00 %	50.00 %
CVA1-KF	21.69 %	14.31 %	40.69 %	13.13 %
CVA2-KF	19.57 %	4.41 %	40.89 %	6.99 %
MCD-KF	26.86 %	21.65 %	22.02 %	20.95 %
DE-KF	33.05 %	27.47 %	26.92 %	30.44 %

(b) Expected Calibration Error (ECE)

Table 5.9: Wind Turbine Filtering - Prediction Coverage and ECE: Results from Section 5.7.2 (Case B: Filtering). The calibration curve of the models are illustrated in Figure 5.23.

5.7.3 Model Interpretation

Overview

The features learned by the neural networks tend to be stronger than the features learned by `CVA-1`. Though not identical, the temperature dynamics learned by `DE`, `MCD` and `CVA-1` are overall similar. The uncertainties learned by the neural networks are inversely correlated with the density of the training data. Though not identical between `MCD` and `DE`, their contours are reasonably similarly shaped.

What separates the temperature models?

Figure 5.24 compares the temperature dynamics learned by `DE`, `MCD`, and `CVA-1`. The contour plots indicate that the neural networks learn stronger features than the linear model does. The largest differences appear to be for the generator speed ω. We can recognize that the models have all learned vector fields that are physically sensible: The contours all indicate that an increased the bearing temperature will result in larger temperature drops each time step. `MCD` has very jagged contours - a consequence of the bimodal distribution over the parameters and the small network size. As 100 forward passes are done at each input coordinate, this might not be simple to mitigate.

The negative correlation between power output P and bearing temperature change is interesting. It likely ties into the correlation between generator speed and power output: The wind energy that *isn't* converted to electrical energy is dissipated. For a given generator speed, a lower power output will lead to larger heat fluxes into the main bearing.

Overall, the differences between the models aren't enormous. The neural networks have learned stronger features - especially `DE` - but the overall relations between inputs and predicted bearing temperature change are similar.

What Uncertainties have the Neural Networks Learned?

Figure 5.25 compares the predictive uncertainty learned by `DE` and `MCD`, along with the density of the training data. The uncertainties are given in terms of the predictive uncertainty standard deviation - which is the square root of their output process noise variance Q for each input combination. Similarly to in Section 5.7.3, the contours from `MCD` are jagged.

Figure 5.25 provides us some insight on the predictive uncertainty learned by the networks. The uncertainty tends to be negatively correlated with the training data density, which is reasonable: The networks will learn the dynamics better for areas of the input space to which they are more exposed. We can recognize that `DE` yields the larger variations in predictive uncertainty, as it learns the inherent noise as a function of the inputs. Variations in the uncertainty of `MCD` are only due to the uncertainty in the model parameters. Though magnitudes and finer details differ between the models, the uncertainty contours have similar shapes.

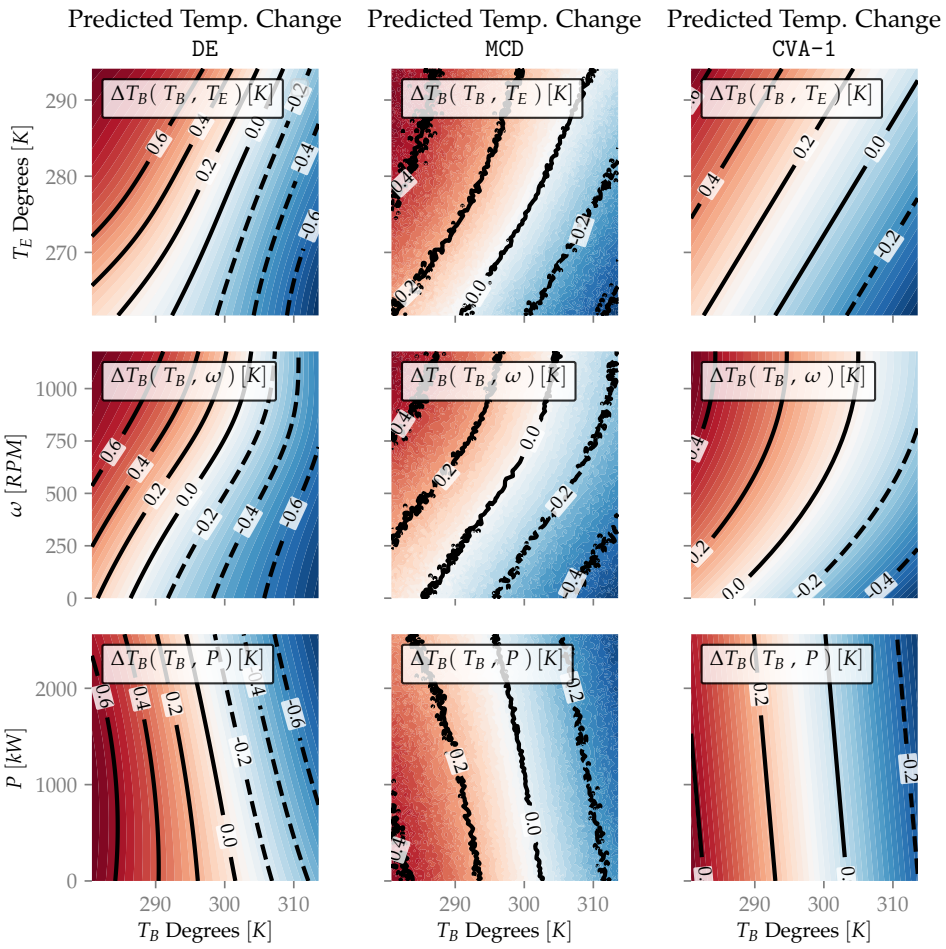


Figure 5.24: Bearing Temperature Change Contours: The contour plots show the predicted temperature change ΔT_B as a function of the current bearing temperature T_B (along the x-axes) and an additional input (along the y-axes). Each column corresponds to a model. Each row corresponds to an input: External temperature T_E at the top, generator speed ω in the middle, and power output P at the bottom. The dotted contour lines (with blue background color) indicate the bearing temperature is predicted to drop. The continuous contour lines (with red background color) indicate the bearing temperature is predicted to rise. As each row shows the different models' predictions for the same combination of inputs, we can expect them to be reasonably similar.

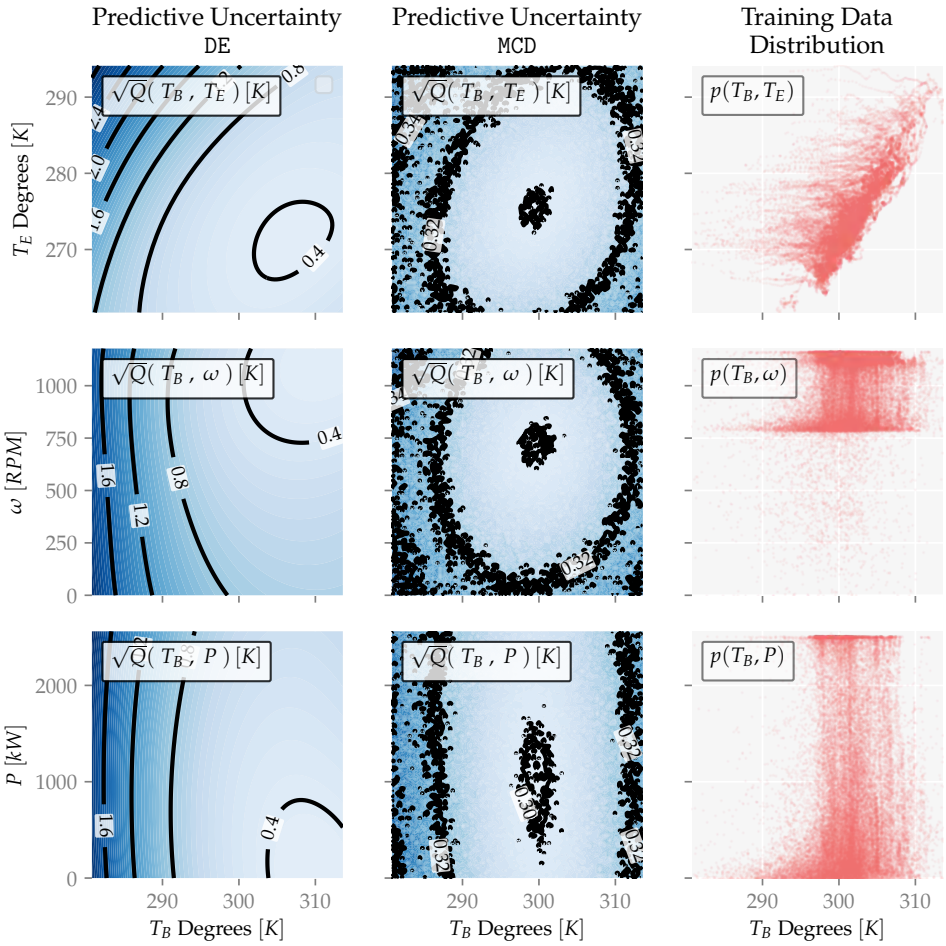


Figure 5.25: Predictive Uncertainty Contours and Training Data Distribution: The contour plots show the uncertainty $\sqrt{Q}$ in the predicted temperature change as a function of the current bearing temperature T_B (along the x-axes) and an additional input (along the y-axes). $\sqrt{Q}$ is the standard deviation of the predicted process noise. Each row corresponds to an input: External temperature T_E at the top, generator speed ω in the middle, and power output P at the bottom. The two first columns show the uncertainty contours of the neural networks. The rightmost column shows the joint distribution of the training data in terms of the bearing temperature and the additional input given by the row. The contour lines (and the background color shade) indicate the magnitude of the predictive uncertainty. Remark how the predictive uncertainties in each row tend to be lower where the data distribution in the same row is denser.

5.7.4 Summary of Results

Table 5.10 gives a very brief summary of the results from simulation and filtering.

Though the context will dictate what model is optimal, the trend is that the neural network models were slightly more versatile than the linear models: Outperforming the least-squares' models when simulating, and yielding better results than the system identification-based models when filtering. In addition to this, the neural network-based models generalized surprisingly well when applied to other turbines. Aside from that, model averaging consistently improved predictive performance and predictive uncertainty calibration of the system identification-based models when simulating. This indicates it can be a useful approach to obtain improved predictions when multiple models are available.

`DE` yielded *significantly* better calibrated uncertainty estimates than the other models when simulating on new turbines. No models offered calibrated predictive uncertainty estimates then filtering.

The results demonstrate the importance of having a similar training environment and operation (evaluation) environment for maximizing performance. Models optimized for simulation tend to yield good results for simulation - and models optimized for one-step-ahead predictions tend to yield good results for one-step-ahead predictions.

The predictive uncertainty of both `MCD` and `DE` are negatively correlated to the density of the training data. The uncertainties predicted by `DE` have larger variations in magnitude. This is due to `DE` learning the inherent noise in the uncertainty as a function of its inputs.

	Simulation	**Filtering**
Accuracy	`BMA`, `MCD`, `CVA-2`, `DE`	`LS-H`, `MCD`
Calibration	`DE`	-

Table 5.10: Overview of Wind Turbine Experiment Results: A brief overview of which models yielded better results for the two cases in Section 5.6.

Chapter 6

Discussion

6.1 A Retrospective View on Wind Turbine Modeling

The work related to Chapter 5 has spanned the entirety of the semester. We there show how the methods presented Chapter 3 can be applied for a real-world case. The implementations considered iterate on what has been explored by previous authors for similar modeling tasks.

It's interesting that it appears feasible to obtain reasonably calibrated uncertainty estimates when simulating the turbine bearing temperatures. Though the models are experimental, this result shows that the notion of learning both turbine dynamics and their uncertainty is realistic. This can prove very useful: It indicates we can apply a model on new turbines and obtain trustable confidence intervals for the turbine bearing temperature, after a simple bias correction.

As improved data handling significantly increased predictive performance, it is likely that better pre-processing techniques can yield further improvements. In retrospect, focusing primarily on a real-world case when developing the methods might not have been an ideal choice. Distinguishing between unsuitable methods and inaccuracies stemming from the data quality has at times been challenging. However, the benefit of the wind turbine experiment is that it shows the methods from Chapter 3 are actually applicable in a real-world scenario.

Though the primary focus of this thesis is angled towards neural networks with uncertainty estimation, a major benefit of the traditional system identification approaches considered in Chapter 5 is that they are well-established and quick to employ. As implementations are readily available, system identification can enable rapid model identification with limited effort required. This makes them a versatile and useful option - especially when taking into account the value of time.

The results from the various linear model iterations in Chapter 5 demonstrate

how much impact small changes in a model can have. These changes were primarily motivated by the physics which govern the system in question. This yields a positive observation: A high level understanding of governing physics can make it far easier to make sensible modeling decisions - also when the parameter identification is data-driven. Although exact equations might not be possible to derive, they can often be feasible to approximate for physical processes. This enables us to make good choices on what information to feed to a model in order to facilitate learning and minimize its complexity.

6.2 The Relevance of Grid-Based Uncertainty Propagation

Grid-based uncertainty propagation through machine learning models that have a variable output uncertainty is theoretically grounded and feasible to implement. This opens up interesting new possibilities, such as using neural networks to represent both system dynamics and process noise in a state space formulation.

In general, grid-based uncertainty propagation methods are very suitable for filtering when sensor measurements are available. This is also the case for long-horizon simulations of dissipative systems, as in Chapter 5. Long-horizon simulations for a general system with a higher-dimensional state space might not be as ideal: The system cannot be reasonably assumed to maintain a Gaussian (or near-Gaussian) state probability distribution over time. This will however be system-dependent, as seen in Chapter 4. A reasonable approach for accurate predictions with a multivariate system might be to switch to a Monte Carlo simulation when simulating, and use grid-based methods when filtering.

The uncertainty propagation method used in this thesis builds upon existing grid-based uncertainty propagation algorithms. As a consequence, its accuracy is subject to the accuracy of the underlying algorithm being used. This means that for complex dynamics, higher accuracy might be achievable by employing higher order grid-based propagation methods. However, it's possible that this could be negated by inaccuracy stemming from the Gaussian assumption.

Grid-based uncertainty propagation methods can yield high-accuracy results with significantly lower computational requirements than a Monte Carlo simulation. However, they will not be suitable for all use-cases, as they impose assumptions on the uncertainty distributions. Still, being derivative-free, they are applicable for systems where gradients are not available and alternative methods such as linearization is not possible. Even in cases where linearization is possible, grid-based methods can in general be expected to yield the more accurate results.

6.3 Representing Process Dynamics and Process Noise with Neural Networks

The primary experiment considered in this thesis indicates it's possible to learn system dynamics and associated uncertainties using neural networks. However, it does not guarantee the approach is applicable for more general systems. This includes systems with higher order dynamics or multiple hidden states. More thorough experimentation with the method would have been desirable, but the scope of the thesis was already broad. However, the overall approach appears promising. It's beneficial that it's interpretable both in terms of traditional modeling techniques and recent advances in neural networks. This indicates it could prove a sensible way to employ machine learning models with uncertainty estimation for common control engineering tasks.

The two neural network models used in this thesis - based on Monte Carlo dropout and Deep Ensembles - each have strength and drawbacks. The main experiment indicates that learning a lower bound on the process uncertainty as a function of model inputs is beneficial, though this requires a larger parameter count. This corresponds with results in existing research. In addition, the results indicate that optimizing directly on error in predictive mean is necessary to maximize predictive accuracy. This is not surprising, but can make it challenging to directly apply certain uncertainty estimation techniques for neural networks when representing dynamics. It's possible to combine the two uncertainty estimation methods employed in the thesis. This could enable us to leverage each of their strengths.

Process output measurement noise was not learned by the networks employed in this thesis, but approximated based on a data-sheet. This was likely not an ideal choice, and including the sensor characteristic as a learnable parameter in the models could improve the uncertainty estimates. Aside from this, it was observed that uncertainty estimation suffered when filtering, as the networks are optimized to predict their uncertainty when simulating. A possible approach to combating this could be to include context as an input feature to the model, to indicate if it's currently used for simulation or filtering. This could potentially compensate for the models being under-confident when applied for filtering.

Though the implementations used in the thesis are experimental, they hint at exciting potential. System models that generalize well and estimate their uncertainty are an enticing concept well suited for industrial application. Even if the networks used don't individually perform optimally on all accounts, this can likely be mitigated by fusing the methods. It appears that formulating a neural network-based system model capable of jointly learning highly accurate dynamics and calibrated uncertainty estimates is achievable - but not entirely straightforward.

6.4 Are Uncertainty Estimates Necessary?

Machine learning and artificial intelligence are becoming widespread. We cannot expect a user interacting with a machine learning-based system to have an in-depth knowledge of its inner workings. Likewise, we cannot expect them to have a full understanding of the ways in which its predictions can fail. A machine learning algorithm with calibrated predictive uncertainty estimates can enable users to make informed decisions without necessitating a complete understanding of its inner mechanics. Similarly, they can enable an algorithm to contain an error before it causes a failure. In a sense, calibrated uncertainty estimates can add some degree of transparency to a 'black-box' algorithm. However, the utility of this relies fully on the accuracy of said estimates.

Although the models considered Chapter 5 are simple, they illustrate important challenges that uncertainty estimation can bring. Including uncertainty estimation in an algorithm arguably shifts the responsibility of making critical judgement on its output away from humans. This can have severe safety implications if important decisions are made based on inaccurate or downright wrong uncertainty estimates. Looking at model predictions *in context of the ground truth*, we can judge their validity reasonably well. However, for successful large scale uncertainty estimation, we *need to know in advance* that the uncertainty estimates are reliable.

Some methods for uncertainty estimation in neural networks are very promising. Incorporating uncertainty estimation in a system can provide useful information for making decisions and evaluating system performance. Just as any other method, the context will dictate whether uncertainty estimation is necessary. You want your car autopilot to be fail-safe. It isn't as critical if your phone's voice assistant performs a wrong action because it didn't ask you to repeat yourself. If uncertainty estimation *is* included in an algorithm, it's crucial that it provides calibrated uncertainty estimates or that the calibration curve (reliability diagram) is known. Otherwise, the downsides of incorporating it will likely outweigh the benefits, as users can be presented with false guarantees of trustworthiness.

6.5 Future Work

To iterate on the model representation from Section 3.2, looking into higher order process dynamics could be useful. This would make the approach applicable for a far wider variety of processes. Further, it could be of interest to apply a combination of the uncertainty estimation methods used for the neural networks (or entirely new methods) in order to further improve on the models' predictive quality. In addition to this, it's possible that uncertainty estimate calibration when filtering could be improved by including the context (filtering or simulating) as a model input feature. This would only require small changes, but could be interesting to look into. As the approach from Section 3.2 is similar to Gaussian Process dynamic state space models, it's possible that methods already employed there could be borrowed to improve the approach.

Many processes have non-Gaussian noise, and learning this could be useful for fine-tuning predictive uncertainty. This could likely be parametrized by a neural network output, similarly to how Gaussians are parametrized by the neural networks used in this thesis.

Adding soft constraints to neural network optimization criteria has been explored by previous authors. It would be interesting to look into whether physically motivated constraints such as passivity and input-to-state stability can be beneficial when learning system dynamics and uncertainty.

Lastly, it would be interesting to explore smoothing algorithms for the models considered in this thesis. They could possibly be suitable for limiting the effect of missing data points both when learning dynamics and when performing inference.

Chapter 7

Conclusion

In this master's thesis, we have shown how we can extend common grid-based uncertainty propagation methods in order to apply them on neural networks with predictive uncertainty estimation. We have then demonstrated how this can be applied to formulate a Gaussian filtering algorithm applicable for process models with variable output uncertainty. Further, we have shown how this can be applied to learn and estimate the uncertainty in a dynamical system model represented by a neural network.

These methods are then used to develop dynamical models for the main bearing temperature of a specific type of wind turbine using real data. When applying these models on *new* wind turbines, their predictive accuracy is comparable to a Bayesian model average of two system identification-based models. Further, it appears these methods can be capable of providing much improved uncertainty estimate calibration when simulating on new turbines - though a trade-off between predictive accuracy and calibration was observed for the models used. When applied for filtering, the neural network-based models outperform the models identified using a system identification algorithm. Still, a model optimized for one-step-ahead predictions yields the highest predictive accuracy when measurements are available, which underlines the importance of considering application context when developing a model: A model optimized for simulation won't necessarily be optimal for one-step-ahead predictions.

Observations from the main experiment aligns with existing research on predictive uncertainty estimation for neural networks: Learning the uncertainty in the model parameters and learning the uncertainty in the model output are both beneficial for estimating the total uncertainty in the prediction. A combination of the neural network methods used in this thesis would likely yield further improvements, as this could leverage their individual strengths. Still, the neural networks considered in the main experiment are smaller than often considered in similar research, and it's promising that some uncertainty estimation techniques are applicable in light of this.

Overall, the results indicate that a neural network-based model with uncertainty estimation can be a versatile approach for dynamic process modeling, capable of generalizing surprisingly well when simulating in new environments (new wind turbines). Coupled with the possibility of learning more calibrated uncertainty estimates, this hints at exciting potential.

The main conclusions we can draw from the work in this thesis, are

1. Existing grid-based uncertainty propagation methods (e.g. the *unscented transform*) can be extended to propagate uncertainty through machine learning models with variable output uncertainty. This makes it possible to apply predictive uncertainty estimation techniques for neural networks in combination with computationally efficient derivative free uncertainty propagation methods. Further, it's possible to formulate a Gaussian filtering algorithm for dynamical models with variable output uncertainty using these methods.
2. We can apply existing uncertainty estimation methods for small-scale neural networks. These can be used to model a dynamical system and it's uncertainties. This enables jointly identifying process dynamics and process uncertainty through stochastic optimization.
3. These techniques can be applied for real-world systems. Process dynamics and uncertainties for the main bearing temperature of wind turbines from a wind farm in Norway were modeled using neural networks with uncertainty estimation. However, when simulated on new turbines, it was found that only one of the methods considered - Deep Ensembles - yielded calibrated (*accurate*) uncertainty estimates. A trade-off between predictive accuracy and calibration was observed for the models applied in this thesis, but it is very likely that this can be mitigated by combining the uncertainty estimation methods considered and fine-tuning the optimization procedures.
4. An understanding of the governing physics in a process can be very useful. Approximations of the process dynamics can help us make good modeling choices by indicating what input features are sensible to include in a process model, also when using data-driven machine learning to identify the model parameters. The most substantial increases in simulation accuracy observed in the main experiment resulted from model changes directly motivated by the governing physics.

Bibliography

[1] Python, *Version 3.6.9*, https://www.python.org/, Python Software Foundation.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, 'Pytorch: An imperative style, high-performance deep learning library', in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[3] L. Luo, Y. Xiong, Y. Liu and X. Sun, 'Adaptive gradient methods with dynamic bound of learning rate', in *Proceedings of the 7th International Conference on Learning Representations*, New Orleans, Louisiana, 2019.

[4] R. R. L. Jr, *FilterPy: Python Kalman filtering and optimal estimation library*, https://github.com/rlabbe/filterpy.

[5] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt and S. 1. 0. Contributors, 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python', *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: https://doi.org/10.1038/s41592-019-0686-2.

[6] MATLAB, *9.7.0.1296695 (r2019b) update 4*, The MathWorks Inc., 2019.

[7] C. of Practice in Computer Science Education at NTNU, *Copcse-ntnu thesis template*, https://github.com/COPCSE-NTNU/thesis-NTNU.

[8] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[9] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, 'End to end learning for self-driving cars', *arXiv preprint arXiv:1604.07316*, 2016.

[10] N. Laptev, J. Yosinski, L. E. Li and S. Smyl, 'Time-series extreme event forecasting with neural networks at uber', in *International Conference on Machine Learning*, vol. 34, 2017, pp. 1–5.

[11] S. Eldevik, *Ai + safety. safety implications for artificial intelligence: Why we need to combine causal- and data-driven models*, https://www.dnvgl.com/oilgas/download/artificial-intelligence-ai-and-safety.html. [Online]. Available: https://www.dnvgl.com/oilgas/download/artificial-intelligence-ai-and-safety.html.

[12] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez *et al.*, 'A berkeley view of systems challenges for ai', *arXiv preprint arXiv:1712.05855*, 2017.

[13] I. 262, *Iso 31000:2018(en) risk management — guidelines*, https://www.iso.org/standard/65694.html.

[14] M. C. Kennedy and A. O'Hagan, 'Bayesian calibration of computer models', *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 3, pp. 425–464, 2001.

[15] J. M. Hernández-Lobato and R. Adams, 'Probabilistic backpropagation for scalable learning of bayesian neural networks', in *International Conference on Machine Learning*, 2015, pp. 1861–1869.

[16] Y. Gal and Z. Ghahramani, 'Dropout as a bayesian approximation: Representing model uncertainty in deep learning', in *international conference on machine learning*, 2016, pp. 1050–1059.

[17] B. Lakshminarayanan, A. Pritzel and C. Blundell, 'Simple and scalable predictive uncertainty estimation using deep ensembles', in *Advances in neural information processing systems*, 2017, pp. 6402–6413.

[18] L. Zhu and N. Laptev, 'Deep and confident prediction for time series at uber', *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017. DOI: 10.1109/icdmw.2017.19.

[19] D. Kim, K. Min, H. Kim and K. Huh, 'Vehicle sideslip angle estimation using deep ensemble-based adaptive kalman filter', *Mechanical Systems and Signal Processing*, vol. 144, p. 106 862, 2020, ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2020.106862.

[20] J. Snoek, Y. Ovadia, E. Fertig, B. Lakshminarayanan, S. Nowozin, D. Sculley, J. Dillon, J. Ren and Z. Nado, 'Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift', in *Advances in Neural Information Processing Systems*, 2019, pp. 13 969–13 980.

[21] S. H. Lee and W. Chen, 'A comparative study of uncertainty propagation methods for black-box-type problems', *Structural and Multidisciplinary Optimization*, vol. 37, no. 3, p. 239, 2009.

[22] C. E. Papadopoulos and H. Yeung, 'Uncertainty estimation and monte carlo simulation method', *Flow Measurement and Instrumentation*, vol. 12, no. 4, pp. 291–298, 2001, ISSN: 0955-5986. DOI: https://doi.org/10.1016/S0955-5986(01)00015-2.

[23] S. J. Julier and J. K. Uhlmann, 'A general method for approximating nonlinear transformations of probability distributions', Tech. Rep., 1996.

[24] A. H. Abdelaziz, S. Watanabe, J. R. Hershey, E. Vincent and D. Kolossa, 'Uncertainty propagation through deep neural networks', 2015.

[25] E. E. Vesterkjær, *Uncertainty propagation and applications to neural networks. TTK4550 specialization project report*, 2019.

[26] W. E. Larimore, 'Canonical variate analysis in identification, filtering, and adaptive control', in *29th IEEE Conference on Decision and Control*, 1990, 596–604 vol.2.

[27] A. E. Raftery, T. Gneiting, F. Balabdaoui and M. Polakowski, 'Using bayesian model averaging to calibrate forecast ensembles', *Monthly weather review*, vol. 133, no. 5, pp. 1155–1174, 2005.

[28] D. Michie, D. J. Spiegelhalter, C. Taylor *et al.*, 'Machine learning', *Neural and Statistical Classification (Reprint from Nature, Vol. 218, No. 5136, pp. 19-22, April 6, 1968)*, vol. 13, no. 1994, pp. 1–298, 1994.

[29] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu and Y. Zhang, 'Short-term residential load forecasting based on lstm recurrent neural network', *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 841–851, 2017.

[30] P. Cambron, A. Tahan, C. Masson and F. Pelletier, 'Bearing temperature monitoring of a wind turbine using physics-based model', *Journal of Quality in Maintenance Engineering*, pp. 00–00, Aug. 2017. DOI: 10.1108/JQME-06-2016-0028.

[31] G. Chowdhary and R. Jategaonkar, 'Aerodynamic parameter estimation from flight data applying extended and unscented kalman filter', *Aerospace Science and Technology*, vol. 14, no. 2, pp. 106–117, 2010, ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2009.10.003.

[32] F. Rosenblatt, 'The perceptron: A probabilistic model for information storage and organization in the brain.', *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[33] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, 'You only look once: Unified, real-time object detection', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[34] S. Ren, K. He, R. Girshick and J. Sun, 'Faster r-cnn: Towards real-time object detection with region proposal networks', in *Advances in neural information processing systems*, 2015, pp. 91–99.

[35] M. Raissi, P. Perdikaris and G. E. Karniadakis, 'Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations', *arXiv preprint arXiv:1711.10561*, 2017.

[36] S. E. Ahmed, S. M. Rahman, O. San, A. Rasheed and I. M. Navon, 'Memory embedded non-intrusive reduced order modeling of non-ergodic flows', *Physics of Fluids*, vol. 31, no. 12, p. 126 602, 2019, ISSN: 1089-7666. DOI: 10.1063/1.5128374. [Online]. Available: http://dx.doi.org/10.1063/1.5128374.

[37] T. Q. Chen, Y. Rubanova, J. Bettencourt and D. K. Duvenaud, 'Neural ordinary differential equations', in *Advances in neural information processing systems*, 2018, pp. 6571–6583.

[38] J. Long, E. Shelhamer and T. Darrell, 'Fully convolutional networks for semantic segmentation', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[39] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, 'Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs', *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[40] Y. LeCun, Y. Bengio and G. Hinton, 'Deep learning', *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[41] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[42] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning*. MIT press, 2016.

[43] X. Glorot, A. Bordes and Y. Bengio, 'Deep sparse rectifier neural networks', in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[44] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980*, 2014.

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, 'Dropout: A simple way to prevent neural networks from overfitting', *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[46] A. K. Y. Gal, 'What uncertainties do we need in bayesian deep learning for computer vision?', *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[47] Y. Wen, P. Vicol, J. Ba, D. Tran and R. Grosse, 'Flipout: Efficient pseudo-independent weight perturbations on mini-batches', *arXiv preprint arXiv:1803.04386*, 2018.

[48] J. Gast and S. Roth, *Lightweight probabilistic deep networks*, 2018. arXiv: 1805.11327 [cs.CV].

[49] A. G. Wilson and P. Izmailov, 'Bayesian deep learning and a probabilistic perspective of generalization', *arXiv preprint arXiv:2002.08791*, 2020.

[50] O. Egeland and J. T. Gravdahl, *Modeling and simulation for automatic control*. Marine Cybernetics Trondheim, Norway, 2002.

[51] C.-T. Chen, *Linear system Theory and Design, International Edition*. Oxford University Press, Inc., 2010, ISBN: 0-19-511595-3.

[52] H. K. Khalil and J. W. Grizzle, *Nonlinear Systems, Third Edition*. Prentice Hall, 2002, vol. 3.

[53] P. Van Overschee and B. De Moor, 'N4sid: Subspace algorithms for the identification of combined deterministic-stochastic systems', *Automatica*, vol. 30, no. 1, pp. 75–93, 1994.

[54] A. Delgado, C. Kambhampati and K. Warwick, 'Dynamic recurrent neural network for system identification and control', *IEE Proceedings - Control Theory and Applications*, vol. 142, no. 4, pp. 307–314, 1995.

[55] E. B. Kosmatopoulos, M. M. Polycarpou, M. A. Christodoulou and P. A. Ioannou, 'High-order neural network structures for identification of dynamical systems', *IEEE Transactions on Neural Networks*, vol. 6, no. 2, pp. 422–431, 1995.

[56] S. Hochreiter and J. Schmidhuber, 'Long short-term memory', *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[57] A. Graves, A. Mohamed and G. Hinton, 'Speech recognition with deep recurrent neural networks', in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[58] M. Raissi, P. Perdikaris and G. E. Karniadakis, 'Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations', *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[59] E. A. Wan and R. Van Der Merwe, 'The unscented kalman filter for nonlinear estimation', in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*, IEEE, 2000, pp. 153–158.

[60] E. A. Wan, R. Van Der Merwe and A. T. Nelson, 'Dual estimation and the unscented transformation', in *Advances in neural information processing systems*, 2000, pp. 666–672.

[61] R. E. Kalman, 'A new approach to linear filtering and prediction problems', *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[62] D. Giurghita and D. Husmeier, 'Statistical modelling of cell movement', *Statistica Neerlandica*, Apr. 2018. DOI: 10.1111/stan.12140.

[63] L. A. McGee and S. F. Schmidt, 'Discovery of the kalman filter as a practical tool for aerospace and industry', 1985.

[64] R. G. Brown and P. Y. C. Hwang, *Introduction to Random Sigmals and Kalman Filtering, Third Edition*. John Wiley & Sons, 1997, vol. 3, ISBN: 0-471-12839-2.

[65] N. J. Higham, 'Cholesky factorization', *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 2, pp. 251–254, 2009.

[66] S. J. Julier and J. K. Uhlmann, 'New extension of the kalman filter to nonlinear systems', in *Signal processing, sensor fusion, and target recognition VI*, International Society for Optics and Photonics, vol. 3068, 1997, pp. 182–193.

[67] S. J. Julier and J. K. Uhlmann, 'Unscented filtering and nonlinear estimation', *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[68] K. Ito and K. Xiong, 'Gaussian filters for nonlinear filtering problems', *IEEE Transactions on Automatic Control*, vol. 45, no. 5, pp. 910–927, 2000.

[69] B. Jia, M. Xin and Y. Cheng, 'Sparse gauss–hermite quadrature filter with application to spacecraft attitude estimation', *Journal of Guidance, Control, and Dynamics*, vol. 34, pp. 367–379, Mar. 2011. DOI: 10.2514/1.52016.

[70] B. Jia, M. Xin and Y. Cheng, 'Sparse-grid quadrature nonlinear filtering', *Automatica*, vol. 48, no. 2, pp. 327–341, 2012, ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2011.08.057.

[71] R. Van Der Merwe *et al.*, 'Sigma-point kalman filters for probabilistic inference in dynamic state-space models', PhD thesis, 2004.

[72] J. Stoer and R. Bulirsch, *Introduction to numerical analysis*. Springer Science & Business Media, 2013, vol. 12.

[73] P. K. Kythe and P. Puri, 'Computational methods for linear integral equations', 2002.

[74] S. J. Julier, 'A skewed approach to filtering', in *Signal and Data Processing of Small Targets 1998*, International Society for Optics and Photonics, vol. 3373, 1998, pp. 271–282.

[75] J. A. Hoeting, D. Madigan, A. E. Raftery and C. T. Volinsky, 'Bayesian model averaging: A tutorial', *Statistical science*, pp. 382–401, 1999.

[76] R. E. Kass and A. E. Raftery, 'Bayes factors', *Journal of the american statistical association*, vol. 90, no. 430, pp. 773–795, 1995.

[77] T. Gneiting and A. E. Raftery, 'Strictly proper scoring rules, prediction, and estimation', *Journal of the American statistical Association*, vol. 102, no. 477, pp. 359–378, 2007.

[78] R. Buizza, P. Houtekamer, G. Pellerin, Z. Toth, Y. Zhu and M. Wei, 'A comparison of the ecmwf, msc, and ncep global ensemble prediction systems', *Monthly Weather Review*, vol. 133, no. 5, pp. 1076–1097, 2005.

[79] M. K. A. M. Vollmer, 'Uncertainty in machine learning applications: A practice-driven classification of uncertainty', *International Conference on Computer Safety, Reliability, and Security*, 2018.

[80] S. S. S. K. M. H. O. A. Knoll, 'Uncertainty in machine learning: A safety perspective on autonomous driving', *International Conference on Computer Safety, Reliability, and Security*, 2018.

[81] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy and A. Bouchachia, 'A survey on concept drift adaptation', *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[82] C. Blundell, J. Cornebise, K. Kavukcuoglu and D. Wierstra, 'Weight uncertainty in neural networks', *arXiv preprint arXiv:1505.05424*, 2015.

[83] D. P. Kingma and M. Welling, 'Auto-encoding variational bayes', *arXiv preprint arXiv:1312.6114*, 2013.

[84] Y. Gal and Z. Ghahramani, *Dropout as a bayesian approximation: Appendix*, 2015. arXiv: 1506.02157 [stat.ML].

[85] A. Shafaei, M. Schmidt and J. J. Little, 'A less biased evaluation of out-of-distribution sample detectors', *arXiv preprint arXiv:1809.04729*, 2018.

[86] T. P. Minka, 'A family of algorithms for approximate bayesian inference', PhD thesis, Massachusetts Institute of Technology, 2001.

[87] A. Loquercio, M. Segu and D. Scaramuzza, 'A general framework for uncertainty estimation in deep learning', *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3153–3160, 2020, ISSN: 2377-3774. DOI: 10.1109/lra.2020.2974682. [Online]. Available: http://dx.doi.org/10.1109/LRA.2020.2974682.

[88] K. Chen, K. Chen, Q. Wang, Z. He, J. Hu and J. He, 'Short-term load forecasting with deep residual networks', *IEEE Transactions on Smart Grid*, vol. 10, no. 4, pp. 3943–3952, 2019.

[89] X. Liu, T. Xiao, S. Si, Q. Cao, S. Kumar and C.-J. Hsieh, *Neural sde: Stabilizing neural ode networks with stochastic noise*, 2019. arXiv: 1906.02355 [cs.LG].

[90] M. P. Naeini, G. Cooper and M. Hauskrecht, 'Obtaining well calibrated probabilities using bayesian binning', in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[91] C. Tofallis, 'A better measure of relative prediction accuracy for model selection and model estimation', *Journal of the Operational Research Society*, vol. 66, no. 8, pp. 1352–1362, 2015.

[92] R. E. Walpole, R. H. Myers, S. L. Myers and K. Ye, *Probability & statistics for engineers & scientists*, 2012.

[93] M. Deisenroth and S. Mohamed, 'Expectation propagation in gaussian process dynamical systems', in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 2609–2617.

[94] M. P. Deisenroth and S. Mohamed, *Expectation propagation in gaussian process dynamical systems: Extended version*, 2012. arXiv: 1207.2940 [stat.ML].

[95] I. J. Goodfellow, J. Shlens and C. Szegedy, 'Explaining and harnessing adversarial examples', *arXiv preprint arXiv:1412.6572*, 2014.

[96] J. A. Cuesta and C. Matran, 'Notes on the wasserstein metric in hilbert spaces', *Ann. Probab.*, vol. 17, no. 3, pp. 1264–1276, Jul. 1989. DOI: 10.1214/aop/1176991269. [Online]. Available: https://doi.org/10.1214/aop/1176991269.

[97] N. Pawsey and A. Barratt, 'Evaluation of a variable-pitch vertical axis wind turbine', *Wind Engineering*, vol. 23, no. 1, pp. 23–30, 1999, ISSN: 0309524X, 2048402X.

[98] S. Seo, 'A review and comparison of methods for detecting outliers in univariate data sets', PhD thesis, University of Pittsburgh, 2006.

[99] J. Nocedal and S. J. Wright, *Numerical Optimization, Second Edition*. Springer, 2006, vol. 2.

[100] K. Narendra and P. Gallman, 'An iterative method for the identification of nonlinear systems using a hammerstein model', *IEEE Transactions on Automatic Control*, vol. 11, no. 3, pp. 546–550, 1966.

[101] G.-B. Huang, Q.-Y. Zhu and C.-K. Siew, 'Extreme learning machine: Theory and applications', *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[102] I. E. C. (IEC), 'EN 60751: 2008. industrial platinum resistance thermometers and platinum temperature sensors', 2008.

[103] WIKA, 'WIKA data sheet in 00.17', 2016.

[104] A. Botchkarev, 'Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology', *arXiv preprint arXiv:1809.03006*, 2018.

Appendix A

Grid Based Uncertainty Propagation with Sigma Point Augmentation

This appendix presents an extension to the method in Section 3.1. The resulting approach is suitable for filtering problems with nonlinear measurement functions $\boldsymbol{h}$, in cases where the state transition and process noise is modeled by a machine learning algorithm with uncertainty estimation.

A.1 Derivation

The starting point for this is the same as in Section 3.1: A state space model, and the Bayesian filtering equations.

$$\begin{aligned}
\boldsymbol{x}_k &= \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}) + \boldsymbol{w}_k \quad \in \mathbb{R}^n && \text{(A.1)}\\
\boldsymbol{y}_k &= \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{u}_k, t) + \boldsymbol{v}_k \quad \in \mathbb{R}^m && \text{(A.2)}\\
\boldsymbol{w}_k &\sim \mathcal{N}(\mathbf{0}_n, \mathbf{Q}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1})) && \text{(A.3)}\\
\boldsymbol{v}_k &\sim \mathcal{N}(\mathbf{0}_m, \mathbf{R}) && \text{(A.4)}
\end{aligned}$$

As shown in Section 3.1, the first two moments of an uncertainty propagated through Equation (A.1) can be expressed as,

$$\begin{aligned}
E[\boldsymbol{x}_k] &= \int_{\mathbb{R}^n} \boldsymbol{x}_k p(\boldsymbol{x}_k) d\boldsymbol{x}_k \\
&= \int_{\mathbb{R}^n} \int_{\mathbb{R}^n} \boldsymbol{x}_k p(\boldsymbol{x}_k | \boldsymbol{x}_{k-1}) d\boldsymbol{x}_k p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1} \qquad \text{(A.5)}\\
&= \int_{\mathbb{R}^n} E[\boldsymbol{x}_k | \boldsymbol{x}_{k-1}] p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1}
\end{aligned}$$

$$\begin{aligned} E[\boldsymbol{x}_k \boldsymbol{x}_k^\top] &= \int_{\mathbb{R}^n} \boldsymbol{x}_k \boldsymbol{x}_k^\top p(\boldsymbol{x}_k) d\boldsymbol{x}_k \\ &= \int_{\mathbb{R}^n} \int_{\mathbb{R}^n} \boldsymbol{x}_k \boldsymbol{x}_k^\top p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) d\boldsymbol{x}_k p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1} \\ &= \int_{\mathbb{R}^n} E[\boldsymbol{x}_k \boldsymbol{x}_k^\top|\boldsymbol{x}_{k-1}] p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1} \end{aligned} \tag{A.6}$$

Numerical rules can capture lower order moments of probability distributions perfectly. This enables us to substitute each of the expectations in the integrals with a suitable numerical rule expressed by a set of N_o weighted points $(w^{(j)}_{k|\boldsymbol{x}_{k-1}}, \boldsymbol{x}^{(j)}_{k|\boldsymbol{x}_{k-1}})$.

$$E[\boldsymbol{x}_k] = \int_{\mathbb{R}^n} \Big(\sum_j w^{(j)}_{k|\boldsymbol{x}_{k-1}} \boldsymbol{x}^{(j)}_{k|\boldsymbol{x}_{k-1}} \Big) p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1} \tag{A.7}$$

$$E[\boldsymbol{x}_k \boldsymbol{x}_k^\top] = \int_{\mathbb{R}^n} \Big(\sum_j w^{(j)}_{k|\boldsymbol{x}_{k-1}} \boldsymbol{x}^{(j)}_{k|\boldsymbol{x}_{k-1}} (\boldsymbol{x}^{(j)}_{k|\boldsymbol{x}_{k-1}})^\top \Big) p(\boldsymbol{x}_{k-1}) d\boldsymbol{x}_{k-1} \tag{A.8}$$

We can similarly apply a numerical rule to approximate the outer integral. Expressing this outer set of N_i weighted points as $(w^{(i)}_{k-1}, \boldsymbol{x}^{(i)}_{k-1})$ we then obtain,

$$E[\boldsymbol{x}_k] \simeq \sum_i w^{(i)}_{k-1} \Big(\sum_j w^{(j)}_{k|\boldsymbol{x}^{(i)}_{k-1}} \boldsymbol{x}^{(j)}_{k|\boldsymbol{x}^{(i)}_{k-1}} \Big) \tag{A.9}$$

$$E[\boldsymbol{x}_k \boldsymbol{x}_k^\top] \simeq \sum_i w^{(i)}_{k-1} \Big(\sum_j w^{(j)}_{k|\boldsymbol{x}^{(i)}_{k-1}} \boldsymbol{x}^{(j)}_{k|\boldsymbol{x}^{(i)}_{k-1}} (\boldsymbol{x}^{(j)}_{k|\boldsymbol{x}^{(i)}_{k-1}})^\top \Big) \tag{A.10}$$

The inner substitution enables us to retain some information on the higher moments, similarly to in [71, Algorithm 8], which is useful if the measurement function $\boldsymbol{h}$ is nonlinear. In other cases, the formulation presented in Equations (3.4) and (3.5) will be sufficient.

A.2 Applications to Nonlinear Filtering

We can modify the Unscented Kalman Filtering algorithm [71, Algorithm 8] to enable filtering with models that have variable uncertainty when the measurement function $\boldsymbol{h}$ is nonlinear, yielding an extension of Algorithm 2. This is outlined in Algorithm 6, with the sigma point set described in Algorithm 1. The process state transition and the process noise in Algorithm 6 are expressed using the functions $\boldsymbol{f}$ and $\mathbf{Q}$ in Equations (A.1) and (A.3) respectively.

The main difference from Algorithm 2 is that the predictions from the model (i.e. the state transition output distributions) in Algorithm 6 are discretized using the unscented transform. This is referred to as sigma point *augmentation* (similarly to in [71]).

Sigma point augmentation is performed in Equation (A.16). The uncertainty in the predictions is there approximated as $2n+1$ *conditional* sigma points, for each of the $2n+1$ input sigma points, resulting in $(2n+1)(2n+1)$ total sigma points. In practice, this discretizes the model outputs such that they can be further propagated through $\boldsymbol{h}$. It should be noted that only $2n+1$ model evaluations ($\boldsymbol{f}$ and $\boldsymbol{Q}$) are performed - similarly to in Algorithm 2 - so the complexity is not quadratic in terms of the model evaluation complexity. A convenient way to express the $(2n+1)(2n+1)$ sigma points is as a 3-dimensional tensor, such that the first dimension corresponds to the index of the input sigma points, the second dimension corresponds to the index of the conditional output sigma points, and the third corresponds to the state space. Letting the notation $\mathbf{T}^{(i,j,l)}$ refer to indices (i,j,l) of dimensions $(1,2,3)$ of $\mathbf{T}$, the augmentation of the sigma points can then be expressed as in Equation (A.16),

$$\boldsymbol{\mathcal{X}}_{k|k-1}^{(i)} = \underbrace{\left[\boldsymbol{\mathcal{X}}_{k|k-1}^{*(i)}, \boldsymbol{\mathcal{X}}_{k|k-1}^{*(i)} \pm \sqrt{(n+\kappa)\boldsymbol{\mathcal{Q}}_k^{(i)}}\right]^\top}_{[N_o, n]}, \quad i = 0, \ldots, N_i \tag{A.11}$$

And $\boldsymbol{\mathcal{X}}_{k|k-1}$ will have dimensions $[N_i, N_o, n]$. For propagation through a state transition function, N_i and N_o will be the same.

The two operations which are likely to incur the highest computational cost for this method, are *1)* evaluating the model, and *2)* using a grid-based approximation for the conditional output distributions. In practice, this can largely be parallelized. Importantly, batched Cholesky decompositions (necessary to discretize the conditional output distributions) can be performed on GPUs using `PyTorch`.

A.3 Relations to Existing Methods

This is a general approach that is conceptually similar to the method used in [71, Algorithm 8, Eq. (3.174)] for incorporating additive process noise in the unscented Kalman filter. Both methods first propagate the input uncertainty (approximated by a grid-based method) through a function, and then augment the outputs with additional points that represent the uncertainties in the function output. What separates the two approaches is that [71] uses the unscented transform to express the *joint* distribution of the state uncertainty and process uncertainty, whereas the approach presented here uses *separate* unscented transforms: One to express the input uncertainty, and one to the output uncertainty conditional to a given input. In fact, the difficulty of expressing these conditional uncertainties using the method from [71, Algorithm 8] is what motivated this method originally.

Algorithm 6: Unscented Kalman Filter with Variable Model Uncertainty and Sigma Point Augmentation

$$\hat{\boldsymbol{x}}_0 = E[\boldsymbol{x}_0], \quad \mathbf{P}_0 = E[(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0)(\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_0)^\top]$$
$$N = 2n+1, \text{ where } n = \dim \boldsymbol{x}, \quad \kappa \geq 0$$

for $k = 1 \ldots \infty$

1. **Compute sigma points**: According to Algorithm 1

$$\boldsymbol{\mathcal{X}}_{k-1} = \left[\hat{\boldsymbol{x}}_{k-1} \quad \hat{\boldsymbol{x}}_{k-1} \pm \sqrt{(n+\kappa)\mathbf{P}_{k-1}}\right]^\top \tag{A.12}$$

2.1. **Time update**: Using process dynamics and process noise

$$\boldsymbol{\mathcal{X}}^*_{k|k-1}, \boldsymbol{\mathcal{Q}}_k = \boldsymbol{f}(\boldsymbol{\mathcal{X}}_{k-1}, \boldsymbol{u}_{k-1}), \quad \mathbf{Q}(\boldsymbol{\mathcal{X}}_{k-1}, \boldsymbol{u}_{k-1}) \tag{A.13}$$

$$\hat{\boldsymbol{x}}_{k|k-1} = \sum_{i=0}^{N-1} w^{(i)} \boldsymbol{\mathcal{X}}^{*(i)}_{k|k-1} \tag{A.14}$$

$$\mathbf{P}_{k|k-1} = \sum_{i=0}^{N-1} w^{(i)} \{(\boldsymbol{\mathcal{X}}^{*(i)}_{k|k-1} - \hat{\boldsymbol{x}}_{k|k-1})(\boldsymbol{\mathcal{X}}^{*(i)}_{k|k-1} - \hat{\boldsymbol{x}}_{k|k-1})^\top + \boldsymbol{\mathcal{Q}}^{(i)}_k\} \tag{A.15}$$

2.2. **Time update**: Augment sigma points and predict next output

$$\boldsymbol{\mathcal{X}}^{(i)}_{k|k-1} = \left[\boldsymbol{\mathcal{X}}^{*(i)}_{k|k-1}, \boldsymbol{\mathcal{X}}^{*(i)}_{k|k-1} \pm \sqrt{(n+\kappa)\boldsymbol{\mathcal{Q}}^{(i)}_k}\right]^\top, \quad i = 0, \ldots, N-1 \tag{A.16}$$

$$\boldsymbol{\mathcal{Y}}_{k|k-1} = \boldsymbol{h}(\boldsymbol{\mathcal{X}}_{k|k-1}, \mathbf{u}_{k-1}) \tag{A.17}$$

$$\hat{\boldsymbol{y}}_{k|k-1} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} w^{(i)} w^{(j)} \boldsymbol{\mathcal{Y}}^{(i,j)}_{k|k-1} \tag{A.18}$$

$$\mathbf{P}_{yy,k} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} w^{(i)} w^{(j)} \{(\boldsymbol{\mathcal{Y}}^{(i,j)}_{k|k-1} - \hat{\boldsymbol{y}}_{k|k-1})(\boldsymbol{\mathcal{Y}}^{(i,j)}_{k|k-1} - \hat{\boldsymbol{y}}_{k|k-1})^\top\} \tag{A.19}$$

$$\mathbf{P}_{xy,k} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} w^{(i)} w^{(j)} \{(\boldsymbol{\mathcal{X}}^{(i,j)}_{k|k-1} - \hat{\boldsymbol{x}}_{k|k-1})(\boldsymbol{\mathcal{Y}}^{(i,j)}_{k|k-1} - \hat{\boldsymbol{y}}_{k|k-1})^\top\} \tag{A.20}$$

3. **Measurement update**:

$$\mathbf{K}_k = \mathbf{P}_{xy,k}(\mathbf{P}_{yy,k} + \mathbf{R}_k)^{-1} \tag{A.21}$$

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_{k|k-1} + \mathbf{K}_k(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1}) \tag{A.22}$$

$$\mathbf{P}_k = \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{P}^\top_{xy,k} \tag{A.23}$$

end **for**

Appendix B

Wind Turbine Experiment

B.1 Turbine Data Overview

Tables B.1 to B.4 give details on the data for Wind Turbines 1, 2, 3, and 6, respectively.

	T_B Degrees $[K]$	T_E Degrees $[K]$	P $[kW]$	ω $[RPM]$
Values	82747.0	79203.0	63565.0	78051.0
Mean	299.68	275.42	906.67	798.76
Std	5.48	5.97	839.76	398.51
Min	273.15	258.96	0.0	0.0
25%	297.23	271.06	190.83	786.25
50%	300.3	274.68	588.52	914.63
75%	303.15	279.09	1521.15	1113.19
Max	318.5	301.93	2561.11	1177.16

Table B.1: WTUR1 Data Summary: Statistics of the bearing temperature (T_B), external temperature (T_E), power output (P), and rotation speed (ω) sensor readings. This is the turbine whose data was used for model parameter identification. It has the least amount of missing measurement values.

	T_B Degrees $[K]$	T_E Degrees $[K]$	P $[kW]$	ω $[RPM]$
Values	78593.0	79126.0	60695.0	75030.0
Mean	296.8	275.38	894.2	788.87
Std	5.83	5.92	827.55	400.97
Min	268.42	259.06	0.0	0.0
25%	294.63	271.05	188.93	785.95
50%	297.83	274.7	579.1	892.5
75%	300.6	279.02	1522.18	1110.72
Max	313.49	301.01	2648.94	1180.22

Table B.2: WTUR2 Data Summary: Statistics of the bearing temperature (T_B), external temperature (T_E), power output (P), and rotation speed (ω) sensor readings. This wind turbine has a temperature offset compared to Turbine 1, but aside from that the statistics are relatively similar.

	T_B Degrees $[K]$	T_E Degrees $[K]$	P $[kW]$	ω $[RPM]$
Values	77738.0	78331.0	58221.0	73575.0
Mean	295.91	342.16	879.48	773.93
Std	5.77	47.31	794.67	420.32
Min	269.23	273.15	0.0	0.0
25%	293.5	320.05	202.02	758.57
50%	296.91	333.25	586.27	906.47
75%	299.88	350.1	1451.25	1111.74
Max	312.36	1127.69	2496.93	1177.16

Table B.3: WTUR3 Data Summary: Statistics of the bearing temperature (T_B), external temperature (T_E), power output (P), and rotation speed (ω) sensor readings. This wind turbine has anomalous external temperature readings, possibly due to mislabeled data.

	T_B Degrees $[K]$	T_E Degrees $[K]$	P $[kW]$	ω $[RPM]$
Values	75870.0	75872.0	53043.0	67968.0
Mean	298.43	278.2	912.52	767.16
Std	7.51	6.08	809.05	422.75
Min	271.61	261.83	0.0	0.0
25%	296.18	273.71	222.35	711.74
50%	300.35	277.54	629.78	897.53
75%	303.24	282.03	1477.92	1109.1
Max	316.15	304.61	2586.82	1187.33

Table B.4: WTUR6 Data Summary: Statistics of the bearing temperature (T_B), external temperature (T_E), power output (P), and rotation speed (ω) sensor readings. A large amount of readings are missing - more than for any of the other turbines. Aside from that, the distribution statistics are similar to for Turbine 1.

B.2 System Identification Hyperparameters

System identification was done using `MATLAB`'s [6] `n4sid` method, with *Canonical Variate Analysis* weighting, and *Observable Canonical System Form*. For a first order model this has no impact, but it reduces the degrees of freedom in the parameters of higher order models. The loss function used is the simulation square error. Further options are given in Table B.5

N4SID Option	Value
InitialState	`'estimate'`
N4Weight	`'CVA'`
N4Horizon	`'auto'`
Focus	`'simulation'`
EnforceStability	1

Table B.5: System Identification Hyperparameters: `MATLAB` `n4sid` options used for system identification of `CVA-1` and `CVA-2`. The prediction horizon for optimization were automatically determined by `n4sid`. This yielded optimization horizons of 2 steps for `CVA-1` and 3 steps for `CVA-2` (with 4 and 8 previous inputs used for initialization).

Eirik Ekjord Vesterkjær

Grid-based U.P. and neural networks with U.E.

NTNU
Norwegian University of
Science and Technology