November 2021

Learning State-Dependent Sensor Measurement Models To Improve Robot Localization Accuracy

Troi André Williams
University of South Florida

Learning State-Dependent Sensor Measurement Models

To Improve Robot Localization Accuracy

by

Troi André Williams

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Date of Approval:
November 4, 2021

Keywords: Deep Learning, Robotics, Automation, Range Sensing, Autonomous Navigation

Dedication

This dissertation is dedicated to my mother, father, sister, grandparents, best-friend, other loved ones, and ancestors who lived and sacrificed so that I can achieve this milestone.

Acknowledgments

Thank you, Dr. Yu Sun, for accepting me as one of your doctoral students. Thank you for also allowing me to build a line of work and apply for separate funding from the lab. Furthermore, thank you for the guidance and support you provided throughout my doctoral experience. Next, I would like to thank my committee for the support and feedback during the dissertation process. Finally, thank you to my colleagues in the Robot Perception and Action Lab. It was a pleasure meeting and sharing the lab with you.

I want to thank several entities for supporting me throughout the doctoral experience. These entities include the Florida Education Fund's McKnight Doctoral Fellowship Program, Alfred P. Sloan Foundation University Center of Exemplary Mentoring (UCEM)-2013-5-13MPHDT, the 2019 Microsoft Research Dissertation Research Grant, and the NSF Florida-Georgia Louis Stokes Alliance for Minority Participation Bridge to the Doctorate award HRD #1400837.

I want to thank Intel® for providing access to their AI DevCloud platform and the University of South Florida for providing access to GAIVI, a GPU cluster. I would also like to thank the International Conference on Intelligent Robots and Systems (IROS) for awarding me a travel grant to present at the 2019 IROS Conference in Macau, China.

Thank you to those who mentored me during the doctoral program and provided guidance. These individuals are too many to enumerate. However, some individuals include Mr. Benard Batson, Dr. Ransford Hyman Jr., Dr. Thorna Humphries, and Dr. Marc Boumedine.

Table of Contents

List of Tables

List of Figures

Abstract

This dissertation proposes a novel method called state-dependent sensor measurement models (SDSMMs). Such models dynamically predict the state-dependent bias and uncertainty of sensor measurements, ultimately improving fundamental robot tasks such as localization. In our first investigation, we introduced the state-dependent sensor measurement model framework, described their properties, stated the input and output of these models, and described how to train them. We also explained how to integrate such models with an Extended Kalman Filter and a Particle Filter, two popular robot state estimation algorithms. We validated the proposed framework through a series of localization tasks. The results showed that our framework outperformed the baseline sensor measurement models. Our second investigation explored how to learn accurate state-dependent sensor measurement models with limited sensor data. This work is motivated by the difficulty of collecting large numbers of sensor data for training. To alleviate the burden of collecting large datasets, we leverage transfer learning to train models with artificially generated sensor data followed by real sensor data. We used a series of bootstrap experiments to demonstrate that the proposed transfer learning method produced sensor models that are as accurate as models learned with significantly larger datasets. These results imply that we can quickly learn accurate sensor models with limited data, which is broadly beneficial for robot systems such as autonomous vehicles.

Chapter 1: Introduction

In the context of real-world mobile robots, state estimation is the problem of estimating the state of a robot [48][1]. Typically, states such as the exact location of a robot and the exact locations of obstacles in a robot's environment are not directly observable. However, such states can be inferred from sensor measurements [51]. Therefore, robots rely on sensors to infer their states and the states of their environment[2].

Although sensors play an integral role in estimating the state of a robot, their measurements are generally imperfect because they contain bias and noise. For example, consider a stationary, autonomous mobile robot that uses an onboard camera to measure its bearing to nearby obstacles. Each time such a robot measures its bearing to an obstacle, the robot may observe biased and noisy bearing measurements for several reasons. Observing biased and noisy measurements, in turn, can translate into noisy estimates of the robot's state. Additionally, the noisy state estimates can cause hazardous situations. For instance, the robot may incorrectly estimate that obstacles are not in its path and, as a result, may collide with those obstacles. This example illustrates that robots need an estimate of sensor measurement bias and noise so that they can determine how much they should *trust* a sensor measurement.

However, merely knowing the *average*, sensor measurement bias and noise is not enough because the bias and noise can vary depending on the states of a robot and its environment. For example, a combination of environment lighting and a camera lens imperfection can cause nearby, off-centered obstacles to appear further to the side than they actually are.

[1]A portion of this chapter was published in [58, 59, 60].

[2]An *environment* generally describes a set of features that we are interested in observing or tracking (for example, the locations of landmarks).

The environment lighting and lens imperfection, in turn, can produce bearing measurements with high bias and high noise. However, when obstacles are centered and far away, the robot may not experience the same issue as before; therefore, its measurements may have relatively less bias and less noise. Thus, this example implies that merely knowing the *average* sensor measurement bias and noise is not enough. Instead, a robot must predict the bias and noise of *each* measurement, given the robot and environment states.

To address the issue of predicting state-dependent measurement bias and noise, this dissertation introduces a novel method called *state-dependent, sensor measurement models* (SDSMMs). This method learns to predict parameters for state-dependent measurement distributions. Each state-dependent measurement distribution inherently describes the measurement bias and noise of each sensor measurement at a given state. Thus, a robot can use such distributions to compute quantities such as measurement bias and noise. We also demonstrate how to develop SDSMMs when limited data is available.

1.1 Literature Review

This dissertation explores a few topics in robotics and machine learning. Therefore, we use the following sections to review the literature that is related to our work. Section 1.1.1 discusses the robotics literature related to learning sensor measurement models and quantifying measurement noise. In Section 1.1.2, we discuss using transfer learning to train state-dependent sensor measurement models with small numbers of data.

1.1.1 Sensor Measurement Models

1.1.1.1 Overview

A sensor measurement model is an abstract representation that describes how a sensor measurement is created from a real-world physical sensor and its raw observations. The abstract representation is realized through an algorithm. This algorithm takes the raw observations from the physical sensor and outputs a measurement to an agent such as a robot.

An example of a sensor measurement model is a camera-based, landmark distance-and-bearing sensor model. Given an RGB camera image, this sensor model identifies recognizable features that signify a possible landmark (for example, a QR-code placed on walls to identify individual rooms). Then, using projective geometry, the sensor model estimates the distance and bearing of the feature relative to the camera sensor. In this example, the physical sensor is the camera. The raw observation is the RGB image, while the sensor measurement is the distance-and-bearing of a feature in the real world. Finally, the sensor model computes the relationship between the raw observation and the desired sensor measurement. As this example suggests, the realization of a sensor model depends on the physical sensor, its raw observations, and the output sensor measurement [51]. In this dissertation, we do not discuss the plethora of sensor models since it is out of the scope of this work. However, we encourage interested readers to review resources such as [51].

Aside from the desired output measurement (for example, the distance-and-bearing), sensor models also specify a measure of measurement uncertainty (or noise). The sensor measurement noise provides a way of quantifying the reliability of a sensor measurement. Quantifying this reliability is important throughout various robotics topics, including state estimation. The literature provides many methods for quantifying the sensor measurement noise. We discuss those methods in the remainder of this subsection.

1.1.1.2 Learning and Adapting Fixed Measurement Noises

One early method for quantifying sensor measurement noise was to estimate or learn a fixed measurement noise. For example, one could estimate the sensor measurement noise by choosing a set of near-zero values and adapting them later. A second method can use trial and error experiments to select a measurement noise covariance matrix that yielded the smallest error. Abbeel *et al.* [1] proposed five methods that learn the process and measurement noises for an Extended Kalman Filter using an iterative training algorithm. At the beginning of the training phase, the authors provided initial estimates for both noises.

Then the algorithm iteratively increased or decreased the entries of each noise covariance until the algorithm converged. A third method employed least-squares to estimate the covariance from the residuals between the sensor measurements and the ground-truth measurements.

Since a fixed measurement noise can reduce filter performance or lead to filter divergence [45], techniques were developed to adapt the measurement noise to improve filter performance. The motivation behind this idea is that the fixed measurement noise is near the true measurement noise. However, the noise needs to be adapted during runtime to cope with deviations experienced during runtime. One technique used to adapt measurement noise was fuzzy logic. For example, fuzzy logic was used to adapt the measurement noise covariance matrix of a Kalman Filter online and, in turn, improve the accuracy of GPS and differential GPS sensors using inertial navigation sensors [22, 57]. Chatterjee [7] also presented a fuzzy logic-based system that adapts the measurement noise covariance matrix of an EKF online. However, unlike [22] and [57], this fuzzy logic system was used to solve an EKF-based SLAM problem. This system uses statistical information from the EKF innovation and outputs a scale factor for each diagonal value in the measurement noise covariance matrix. Wei [55] proposed another fuzzy logic system in the context of EKF localization. Like [7], the output of the fuzzy logic system was based on information from the innovation. However, Wei [55] used the average ratio between the actual and theoretical measurement covariance matrices, while Chatterjee [7] used the difference between the two matrices. Furthermore, the fuzzy logic system in [55] outputted a single scale factor for the measurement noise covariance matrix.

Other techniques used prediction spaces or neural networks to adapt the measurement noise covariance matrix as well. For example, Peretroukhin *et al.* [36] used a prediction space to predict a scalar weight that scaled the measurement covariance matrix based on the quality of visual and inertial features. In addition to a fuzzy logic system, Wei [55] presented a neural network-based method that adapted the measurement noise covariance matrix using a single scale factor. This work also used statistics computed from the innovation as input to the neural network. Finally, Brossard *et al.* [5] used a convolutional neural network to adapt the diagonal values of the measurement noise using IMU measurements.

Our proposed method is more expressive than the above methods, where we predict measurement noise directly instead of learning or adapting static measurement noise.

1.1.1.3 Predicting Measurement Noises Directly

Another method for quantifying sensor measurement noise is to learn to predict the noise directly. This method differs from the work mentioned in Section 1.1.1.2. Here, the idea is to predict the novel measurement noises directly and dynamically for each input. Vega-Brown *et al.* [54] proposed a covariance estimation algorithm called CELLO that used hand-crafted features to predict the measurement noise covariance of a sensor. Similar to [54], Hu and Kantor [16] presented a covariance prediction algorithm that used hand-crafted predictors. However, [16] used positive definite matrix decomposition to decompose the noise matrix into diagonal and lower triangular matrices. Then Hu and Kantor employed scalar regression to compute the individual elements of both matrices, which eased the difficulty of computing a positive definite matrix. Petrovski *et al.* [42] learned a regression curve that predicted scalars for the diagonal elements of a measurement noise covariance matrix. Peretroukhin *et al.* [39] expanded their work in [36] with generalized kernels to dynamically estimate the measurement noise using visual and inertial cues for a visual odometry system. Liu *et al.* [28] proposed a method called DICE (Deep Inference for Covariance Estimation), which predicted measurement noise covariance matrices using a convolutional neural network (CNN). Unlike [54], [16], and [42], Liu *et al.* [28] focused on predicting the measurement noise for raw, high dimensional inputs (such as RGB images), while the other methods used low-dimensional hand-crafted features. Choi *et al.* [8] employed a sample-free method to predict aleatoric and epistemic uncertainties in the form of diagonal covariance matrices.

Like these methods, our method uses hand-crafted predictors and is extendable to high-dimensional data (such as camera images) since it utilizes neural networks. However, unlike these methods, our method predicts measurement distributions with full noise covariance matrices. We also assume that sensor measurements contain bias and, therefore, model the measurement bias too.

1.1.1.4 Learning Measurement Distributions

As a step beyond directly predicting measurement noise covariance matrices, the literature proposes methods that predict measurement distributions. For example, Ko and Fox used non-parametric Gaussian process (GP) regression to estimate Gaussian parameters for a measurement model as well as a measurement Jacobian matrix for different filters [21]. Meanwhile, Tallavajhula *et al.* [49] employed a non-parametric distribution regression to estimate non-parametric distribution to model sensors. Our method can predict complex distributions like [49]. However, we restrict our method to predicting Gaussian parameters similar to [21] for now. Unlike [21, 49], we do not maintain training data to predict distributions online, potentially producing a model with a smaller parameter footprint.

Other works utilized neural networks to dynamically predict full measurement distributions, which is similar to our work. Peretroukhin *et al.* [37, 38] employed a Bayesian CNN with dropout layers (which are retained during runtime) to predict the 3D direction of the sun and its associated Gaussian uncertainty. Gilitschenski *et al.* [13] proposed a deep learning method that predicts orientations and their uncertainties using a Bingham distribution. Peretroukhin *et al.* [40] used a multi-headed structure named HydraNet to predict an SO(3) orientation distribution for a visual odometry system. Our method does not rely on dropout layers to predict a measurement distribution and is not restricted to estimating orientation alone. Although our method currently predicts aleatoric uncertainty, we are interested in employing work such as [8] to predict epistemic uncertainty or multi-modal distributions in future work.

1.1.1.5 Learning the Parameters for State Estimators

A final set of work realizes state estimators using neural networks. Since a measurement model is a component of a state estimator, these state estimators inherently learn a sensor measurement model as well. Examples of this idea are BackProp KF and Differentiable Particle Filters. BackProp KF [15] is a trainable Kalman Filter that jointly learns a system model and a measurement model, both of which include variable noise covariance matrices. Like [15], Differentiable Particle Filters [17] also jointly learns a system model and a measurement model for a Particle Filter.

Since such methods learn measurement and system models jointly, these methods might require retraining with novel measurements *and* odometry data if one or more sensors are replaced. Also, these methods might be fixed to a specific system (for example, a particular robot and its sensors). However, our method is only tied to a specific sensor and does not require novel odometry data to learn new measurement models.

1.1.2 Learning Sensor Measurement Models with Limited Data

Learning sensor measurement models requires sizeable training datasets, especially when the underlying mechanism is a deep learning model. However, gathering sizeable training datasets can be difficult due to resources and the number of tasks, both of which can be expensive or time-consuming. To overcome the difficulty of learning with limited data, we leverage deep transfer learning.

Deep transfer learning is a machine learning technique that enables a model to learn a novel task by training on a separate yet related task [56, 34]. This technique is popular in areas such as computer vision and natural language processing (for example, see [56, 34]). The motivation behind deep transfer learning is that training a model on a diverse dataset will allow the model to a set of generic feature extractors [14]; this model is referred to as a pre-trained model. The generic feature extractors can be reused to solve target tasks using another set of training data [14].

Formally, transfer learning is defined as the process of improving the performance of a learner (model) $g(\cdot)$ on a target domain $\mathcal{D}_{\mathcal{T}}$ with its corresponding target task $\mathcal{T}_{\mathcal{T}}$ by leveraging a source domain $\mathcal{D}_{\mathcal{S}}$ and its corresponding source task $\mathcal{T}_{\mathcal{S}}$ [56]. A domain is composed of a feature space $\mathcal{C}$ and a marginal probability distribution $P(\mathbf{\Lambda})$, where $\mathbf{\Lambda} = \{\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_l\} \in \mathcal{C}$. In the context of this dissertation, $\boldsymbol{\lambda}_i$ denotes an input combined state, $\mathcal{C}$ is the space of all possible feature vectors for the model, l is the number of feature vectors in the sample of states $\mathbf{\Lambda}$. For a domain $\mathcal{D}$, a task $\mathcal{T}$ is composed of a label space and a model $g(\cdot)$ [56]. We train the model $g(\cdot)$ using pairs of a feature vector and label. In the context of this dissertation, we say that the label space is $\mathcal{Z}$ if we want the model to predict an expected measurement and noise. Therefore, the model $g(\cdot)$ is trained using the feature vector and label pairs $\{\boldsymbol{\lambda}_i, \mathbf{z}_i\}$, where $\boldsymbol{\lambda}_i \in \mathcal{C}$ and $\mathbf{z}_i \in \mathcal{Z}$. If we desire the model to predict measurement bias and noise, we say that the label space is $\mathcal{E}$. As a result, we train the model $g(\cdot)$ using the feature vector and label pairs $\{\boldsymbol{\lambda}_i, \boldsymbol{\xi}_i\}$, where $\boldsymbol{\xi}_i \in \mathcal{E}$. For a given $\boldsymbol{\lambda}_i$, $\mathbf{z}_i$ is a corresponding observed sensor measurement, while $\boldsymbol{\xi}_i$ is a corresponding observed measurement error.

Transfer learning is beneficial when we have a limited number of data available to learn a target task [56]. Particularly, we can use the parameters from the pre-trained model to initialize the parameters of a novel model that will be used for a target task [14]. Weiss *et al.* [56] lists three different categories of transfer learning. Each case mostly depends on if the domains are different (whether or not the tasks are the same) or the tasks are different (whether or not the domains are the same). This dissertation assumes the source and target domains are the same, while the source and target tasks are somewhat related. Particularly, the source and target label spaces are the same. However, the source and target models are different because the amount of bias and noise may differ between the source and target label spaces.

In this dissertation, we use deep transfer learning to learn state-dependent sensor measurement models with limited data. We chose deep transfer learning since it has been successfully applied to deep learning applications with limited data [56, 34] and deep learning-based, sensor measurement models (for examples, see [37, 38, 28]). In addition, transfer learning was applied to sensor modeling or localization in non-deep learning settings as well (for examples, see [27, 29, 52]). We generate a large dataset of artificial sensor measurements for our source domain using a physical model of the sensor of interest (for example, a distance-and-bearing sensor). Unlike real sensor measurements, we specify the amount of bias and noise in the artificial measurements. By extension, our source task is to train an SDSMM with artificial measurements to predict their measurement biases and noises. Our target domain and task are the data from a real sensor and fine-tuning the SDSMM to predict the measurement bias and noise for the real sensor. To our knowledge, we are the first to employ transfer learning for learning sensor measurement bias and noise using data generated from a physical model of a sensor and data from a real sensor.

1.2 Dissertation Contributions

This dissertation contributes the following to the robotics community:

1. We propose a state-dependent sensor measurement models framework, which predicts multivariate measurement distributions for each given state.

2. We present a method that trains SDSMMs with limited real data in cases where we can only collect a small number of data feasibly.

3. We explore how to regularize SDSMMs using the noise from ground truth sensors.

4. We discuss methods for integrating SDSMMs with the Extended Kalman Filter, the Particle Filter, and the Extended Kalman Particle Filter.

5. We derive back-propagation equations for a multi-modal, multivariate mixture density network.

Chapter 2: Learning State-Dependent Sensor Measurement Models

2.1 Introduction

Before we introduce state-dependent sensor measurement models, we state the assumptions of the work and common notations[3]. In this dissertation, we consider a robot with state $\mathbf{x}_t \in \mathbb{R}^r$ that is equipped with a sensor. The sensor captures raw measurements of the environment. Then software processes the raw measurement to produce a low-dimensional measurement $\mathbf{z}_t \in \mathbb{R}^s$, where $s \leq r$. An example of this scenario is a robot that uses an RGB camera to estimate the range and bearing of landmarks in the immediate area. We assume the sensor or a computer atop the robot runs the software that processes the raw measurements. The robot uses the measurement and a measurement model of the sensor to infer the state of the robot $\bar{\mathbf{x}}_t$. Due to environmental factors and random phenomena, each low-dimensional measurement $\mathbf{z}_t$ contains bias and noise. Furthermore, the bias and noise of each measurement vary as a function of a robot and environment state. For instance, states such as ambient lighting, robot motion, and camera lens distortions can impact the amount of bias and noise in camera-based measurements. Let $\boldsymbol{\lambda}_t$ denote a multi-dimensional vector called a combined state. This vector contains a set of robot and environment states that correlate with the measurement bias $\boldsymbol{\delta}_t \in \mathbb{R}^s$ and noise $\boldsymbol{\sigma}_t \in \mathbb{R}^s$ for a specific sensor. Examples of states can include the amount of ambient light, the perceived motion of the robot, or the positions of landmarks in robot-relative coordinates. We refer to such bias and noise as state-dependent bias and noise. If a measurement model is inaccurate, the model will poorly predict these biases and noises. The inaccurate predictions will also cause fundamental robot tasks such as localization and mapping to fail, reducing the safety of the robot,

[3] A portion of this chapter was published in [58].

humans, other objects, and other beings in the surrounding environment. Therefore, we aim to learn accurate models that predict state-dependent measurement biases and noises given a set of robot and environment states.

2.2 State-Dependent Sensor Measurement Models

2.2.1 Overview

Generally speaking, a *state-dependent, sensor measurement model* (SDSMM) learns how a robot and environment state affects the state-dependent bias and noise in a sensor measurement. Formally, we define an SDSMM as a trainable model, $g(\boldsymbol{\lambda}_i; \mathbf{w})$, that predicts parameters Θ_i for a conditional measurement probability distribution $p(\mathbf{z}_i \mid \Theta_i)$. The input to an SDSMM is a *combined state* $\boldsymbol{\lambda}_i$, which was described in Section 2.1. If one chooses the features of a combined state appropriately, the SDSMM can generalize to unseen environments.

The output of the model is a set of parameters Θ_i for a conditional measurement probability distribution. Since input states and output parameters may have different representations, an SDSMM also maps from the state space to the measurement space. In this dissertation, we restrict the parameters to a multivariate mean $\boldsymbol{\mu}$ and full covariance matrix $\boldsymbol{\Sigma}$ for a Gaussian distribution. We chose a Gaussian distribution because it is a typical distribution used for measurement models. Furthermore, we can use these parameters in state estimators such as an Extended Kalman Filter. The multivariate mean and full covariance matrix are a function of a combined state. In the context of SDSMMs, the Gaussian mean represents either a measurement bias or an expected measurement, the latter of which incorporates measurement bias. The choice between both representations depends on the choice of the input and target variables. The full covariance matrix represents heteroscedastic measurement noise [19]. Since the output parameters describe a conditional probability distribution, we can use the distribution for multiple purposes. Some examples include sampling expected measurements and computing the probability of a measurement $\mathbf{z}$ occurring.

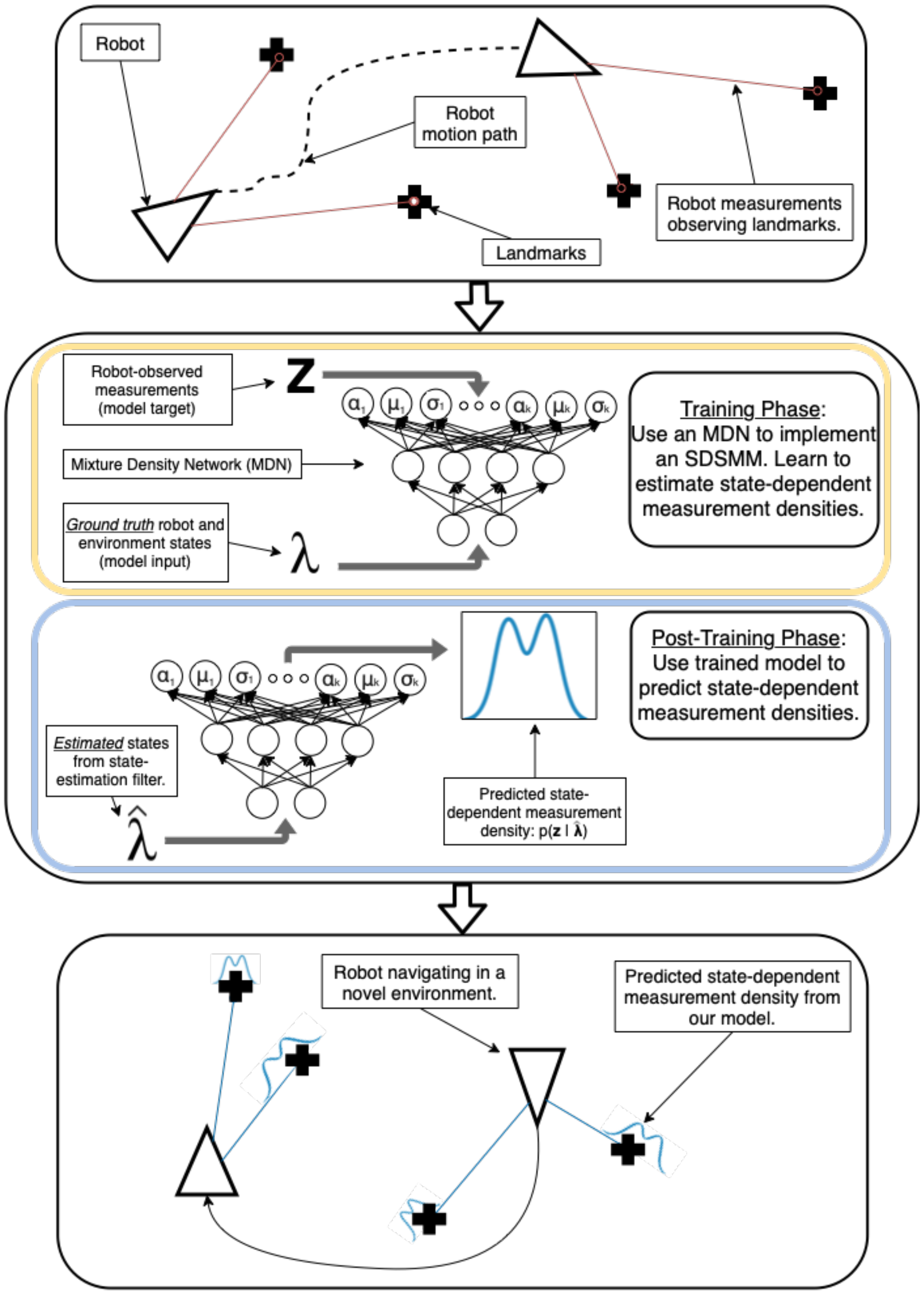

Robot
Robot motion path
Robot measurements observing landmarks.
Landmarks
Robot-observed measurements (model target)
Mixture Density Network (MDN)
Ground truth robot and environment states (model input)
Training Phase: Use an MDN to implement an SDSMM. Learn to estimate state-dependent measurement densities.
Post-Training Phase: Use trained model to predict state-dependent measurement densities.
Estimated states from state-estimation filter.
Predicted state-dependent measurement density: p(z | λ̂)
Robot navigating in a novel environment.
Predicted state-dependent measurement density from our model.

Figure 2.1: How to develop a state-dependent sensor measurement model in three stages. State-dependent sensor measurement models (SDSMMs) are trained offline and deployed online via a three-step process. First, we collect data (sensor measurements and ground-truth states) from an environment (top). Second, we train the SDSMM with the collected data (middle). Finally, a robot uses the SDSMM (with the original sensor) to dynamically predict a measurement distribution for each observed landmark in a *novel* environment (bottom).

SDSMMs have other properties as well. First, filters can incorporate SDSMMs with minimal effort; we show examples in Chapter 4. Second, an SDSMM and a specific sensor form a pair; therefore, the model is deployed on whichever platform the sensor resides. Third, after training an SDSMM, we can use the model to identify combined states that correlate with high measurement bias, high measurement noise, or both. Finally, one can use SDSMMs to develop competency-aware robots that predict if they can safely navigate an environment or manipulate an object.

2.2.2 Learning State-Dependent Sensor Measurement Models

To learn a state-dependent sensor measurement model, we train the model on a set of sensor measurements and their corresponding combined states. Let $\boldsymbol{\Lambda} = [\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \dots, \boldsymbol{\lambda}_I]$ denote a set of I combined states. Let $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_I]$ denote the set of sensor measurements where $\mathbf{z}_i$ was observed at state $\boldsymbol{\lambda}_i$ and $1 \leq i \leq I$. Let $\boldsymbol{\mathcal{D}} = [(\mathbf{z}, \boldsymbol{\lambda})_1, (\mathbf{z}, \boldsymbol{\lambda})_2, \dots, (\mathbf{z}, \boldsymbol{\lambda})_I]$ denote a set of measurement and state pairs where the members are independently and identically distributed. Finally, let $g(\boldsymbol{\lambda}; \mathbf{w})$ denote a state-dependent, sensor measurement model with trainable parameters $\mathbf{w}$ and input $\boldsymbol{\lambda}$. An SDSMM learns to predict Θ, parameters for a conditional measurement probability distribution $p(\mathbf{z} \mid \Theta)$, through learning a set of model parameters $\mathbf{w}$ that maximizes the likelihood of observing the data $\boldsymbol{\mathcal{D}}$. Written formally, we train our model through maximizing the likelihood function:

$$\mathcal{L}(\boldsymbol{\mathcal{D}}, \mathbf{w}) = \prod_{i=1}^{I} \Big(p(\mathbf{z}_i \mid \Theta_i) \Big), \tag{2.1}$$

$$\Theta_i = g(\boldsymbol{\lambda}_i; \mathbf{w}_i), \tag{2.2}$$

where $\mathcal{L}(\boldsymbol{\mathcal{D}}; \mathbf{w})$ is the likelihood function, Θ_i is the parameters for the conditional measurement probability distribution $p(\mathbf{z}_i \mid \Theta_i)$, and $g(\boldsymbol{\lambda}_i; \mathbf{w}_i)$ is the state-dependent sensor measurement model that we are training. In practice, one usually minimizes the negative log-likelihood to improve numerical stability:

$$\mathcal{E}(\boldsymbol{\mathcal{D}}, \mathbf{w}) = -\sum_{i=1}^{I} \ln \Big(p(\mathbf{z}_i \mid \Theta_i) \Big). \tag{2.3}$$

Since an SDSMM learns directly from sensor measurements and their corresponding states, the model parameters **w** also learn how the input state correlates with measurement bias and noise. As a result, the trained model learns to predict state-dependent measurement bias and noise. In addition to measurement noise, we are also interested in learning measurement bias because *a priori*, human-derived measurement models typically do not capture how states and partial sensor calibrations affect measurement bias. As a result, these unmodelled correlations may cause a state estimator (such as a Kalman Filter) to overestimate or underestimate measurement updates when correcting the predicted state of a robot.

2.3 Mixture Density Networks

2.3.1 Overview

A mixture density network (MDN) [4] is a trainable model that consists of a feed-forward deep neural network followed by a mixture model [32] (Figure 2.2). Examples of deep neural networks used include multi-layer perceptions [8, 58], convolutional [13], or recurrent neural networks [61]. The last layer of the neural network uses linear units (that is, no activation function) and outputs the vector **y**. The mixture model serves as the output layer of an MDN, takes **y** as input, and then outputs a set of parameters that completely describe how the target data was generated [4]. As a result, the mixture model allows an MDN to represent arbitrary probability distributions of the desired quantity conditioned upon an input. Mathematically, such probability distributions are defined by a linear combination of

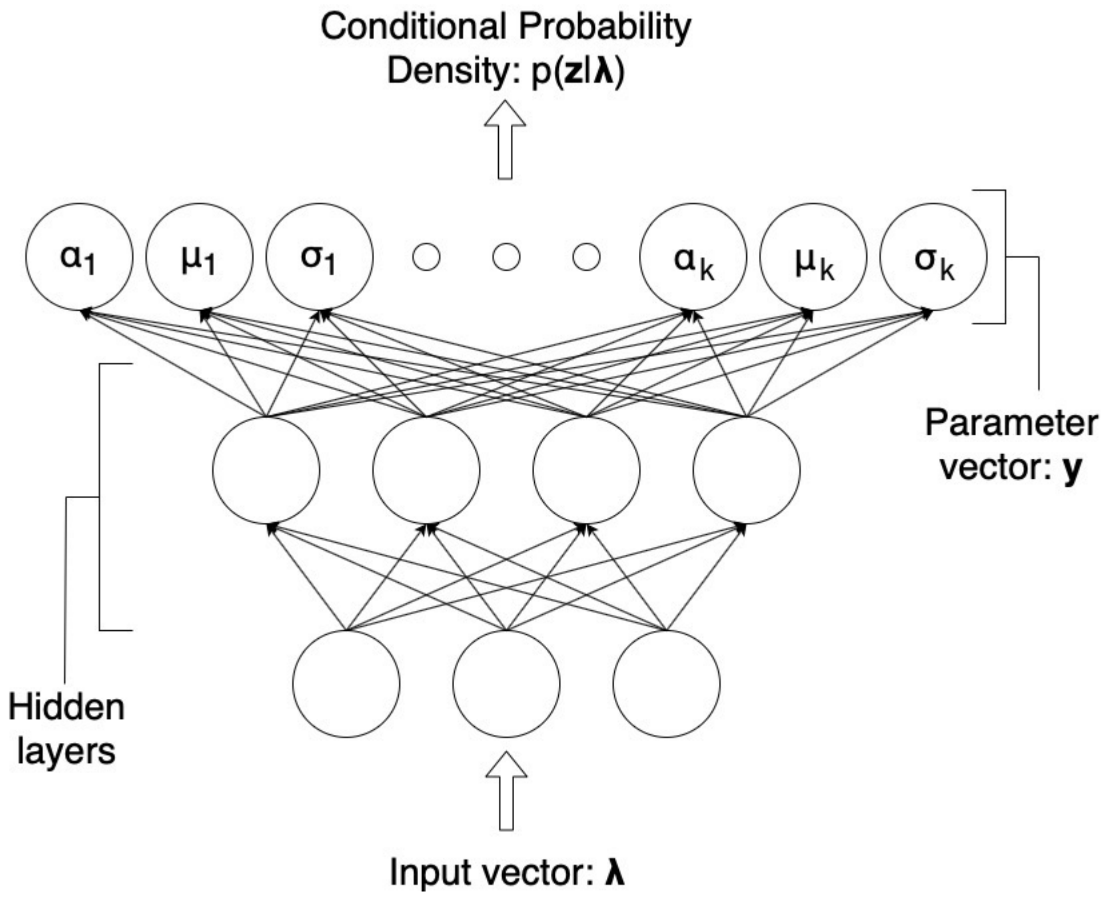


Figure 2.2: The overall architecture of a mixture density network. This neural network can output parameters $\Theta = \{\alpha, \mu, \sigma\}$ for an arbitrary probability distribution $p(\mathbf{z} \mid \Theta)$ conditioned upon some input $\boldsymbol{\lambda}$ [4].

kernel functions

$$p(\mathbf{z} \mid \Theta) = \sum_{k=1}^{m} \alpha_k(\boldsymbol{\lambda})\phi_k(\mathbf{z}|\boldsymbol{\lambda}), \tag{2.4}$$

where m is the number of kernel functions in the model, $\alpha_k(\boldsymbol{\lambda}) \in \mathbb{R}$ is the mixing coefficient of the k-th component, and $\phi_k(\mathbf{z}|\boldsymbol{\lambda})$ is the k-th kernel function. Following [4], we use a Gaussian probability density function as our kernel. However, we used two versions of a Gaussian PDF. The first version comes from [4] and is defined as:

$$\phi_k(\mathbf{z}|\boldsymbol{\lambda}) = \frac{1}{(2\pi)^{s/2}\Big[\sigma_k(\boldsymbol{\lambda})\Big]^s} \exp\left\{ -\frac{\Big|\Big|\mathbf{z} - \boldsymbol{\mu}_k(\boldsymbol{\lambda})\Big|\Big|^2}{2\Big[\sigma_k(\boldsymbol{\lambda})\Big]^2} \right\} \tag{2.5}$$

where s is the number of dimensions in the vector $\mathbf{z}$, and $\sigma_k(\boldsymbol{\lambda}) \in \mathbb{R}$ and $\boldsymbol{\mu}_k(\boldsymbol{\lambda}) \in \mathbb{R}^s$ represent the conditional standard deviation and multivariate mean of the k-th Gaussian PDF. Here, $\alpha_k(\boldsymbol{\lambda})$, $\boldsymbol{\mu}_k(\boldsymbol{\lambda})$, and $\sigma_k(\boldsymbol{\lambda})$ are the parameters for a Gaussian Mixture Model (GMM) with a *common variance*. Each GMM parameter is derived from the vector $\mathbf{y}$. In [4], $[\sigma_k(\boldsymbol{\lambda})]^2$ is referred to as a common variance and, unlike $\boldsymbol{\mu}_k(\boldsymbol{\lambda})$, the common variance is restricted to one dimension.

Our second version implements a full covariance matrix to increase the expressiveness of the MDN. The kernel function, $\phi_k(\mathbf{z}|\boldsymbol{\lambda})$, is defined as:

$$\phi_k(\mathbf{z}|\boldsymbol{\lambda}) = \frac{1}{(2\pi)^{s/2} \cdot \Big|\boldsymbol{\Sigma}_k(\boldsymbol{\lambda})\Big|^{1/2}} \exp\left\{ \Big(\mathbf{z} - \boldsymbol{\mu}_k(\boldsymbol{\lambda})\Big)^\top \cdot \Big[\ \boldsymbol{\Sigma}_k(\boldsymbol{\lambda})\ \Big]^{-1} \cdot \Big(\mathbf{z} - \boldsymbol{\mu}_k(\boldsymbol{\lambda})\Big) \right\} \tag{2.6}$$

where $\boldsymbol{\Sigma}_k(\boldsymbol{\lambda}) \in \mathbb{R}^{s \times s}$ represents the conditional multivariate full covariance of the k-th Gaussian PDF. In this version, $\alpha_k(\boldsymbol{\lambda})$, $\boldsymbol{\mu}_k(\boldsymbol{\lambda})$, and $\boldsymbol{\Sigma}_k(\boldsymbol{\lambda})$ are the parameters for a GMM with a *full covariance matrix* and are also derived from the vector $\mathbf{y}$.

Since the MDN conditional parameters completely describe a probability distribution, we can compute a collection of statistics about the distribution using the values in the parameter vector. Bishop [4] provides several statistics that an MDN can calculate.

2.3.2 Computing the Parameters of a Mixture Density Network

We now describe how to compute the output parameters of an MDN. First, we describe how to compute the k-th mixture coefficient and Gaussian mean. Then we explain how to compute the Gaussian common variance and the Gaussian full covariance. For the remainder of this dissertation, we use the abbreviated symbols α_k, $\boldsymbol{\mu}_k$, σ_k, and $\boldsymbol{\Sigma}_k$ to represent the k-th mixture coefficient, Gaussian mean, Gaussian common variance, and Gaussian full covariance matrix, respectively. Let y_j denote the j-th entry of the network output vector $\mathbf{y}$.

Each mixture coefficient α_k represents a probability of a component. Therefore, all the coefficients must sum to unity: $\sum_{k=1}^{m} \alpha_k = 1$. We use the *softmax* function to compute each mixture coefficient α_k:

$$\alpha_k = \frac{\exp(y_k^{\alpha})}{\sum_{c=1}^{m} \exp(y_c^{\alpha})}, \tag{2.7}$$

where y_k^{α} corresponds to the appropriate entry in the output vector $\mathbf{y}$. To compute the multivariate Gaussian means, we use the output of the deep neural network with no activation function:

$$\boldsymbol{\mu}_k = \begin{bmatrix} \mu_{k,1} \\ \mu_{k,2} \\ \vdots \\ \mu_{k,s} \end{bmatrix} = \begin{bmatrix} y_{k,1}^{\mu} \\ y_{k,2}^{\mu} \\ \vdots \\ y_{k,s}^{\mu} \end{bmatrix}. \tag{2.8}$$

Each Gaussian common variance represents a positive scale in (2.5). Therefore, we apply the exponential function to the appropriate network output entries:

$$\sigma_k = \exp\left(y_k^{\sigma}\right). \tag{2.9}$$

Finally, the multivariate Gaussian full covariance matrix must be a positive definite matrix in (2.6). As a result, our network would need to predict m $\mathbb{R}^{s\times s}$ positive definite matrices. We want to eliminate this constraint such that we can use Stochastic Gradient Descent-based methods to optimize our neural network. Therefore, following Liu *et al.* [28], our neural network predicts a decomposition of the covariance matrices using the LDL decomposition. The decomposition of each covariance matrix is defined as:

$$\boldsymbol{\Sigma}_k = L(\mathbf{y}_k^L)D(\mathbf{y}_k^D)L(\mathbf{y}_k^L)^\top, \tag{2.10}$$

where $\mathbf{y}_k^L \in \mathbb{R}^{(s^2-s)/2}$ and $\mathbf{y}_k^D \in \mathbb{R}^s$ are entries of the vector $\mathbf{y}$. $L(\mathbf{y}_k^L)$ is a predefined function that arranges the entries in $\mathbf{y}_k^L$ into a lower unitriangular matrix:

$$L(\mathbf{y}_k^L) = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ y_{k,1}^L & 1 & 0 & \dots & 0 \\ y_{k,2}^L & y_{k,3}^L & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{k,((s^2-s)/2)-2}^L & y_{k,((s^2-s)/2)-1}^L & y_{k,(s^2-s)/2}^L & \dots & 1 \end{bmatrix}. \tag{2.11}$$

$D(\mathbf{y}_k^D)$ is another predefined function that arranges the entries in $\mathbf{y}_k^D$ into a diagonal matrix:

$$D(\mathbf{y}_k^D) = \begin{bmatrix} \exp(y_{k,1}^D) & 0 & 0 & \dots & 0 \\ 0 & \exp(y_{k,2}^D) & 0 & \dots & 0 \\ 0 & 0 & \exp(y_{k,3}^D) & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \exp(y_{k,s}^D) \end{bmatrix}. \tag{2.12}$$

We use the exponential function $\exp(\cdot)$ to ensure the diagonal entries are positive, guaranteeing that the decomposition exists and is unique.

2.4 Implementing State-Dependent Sensor Measurement Models

Any method implementing an SDSMM must be able to:

1. Map from an input state space to a measurement space,
2. Model how an input state can affect measurement bias and noise, and

3. Output a conditional measurement probability distribution given an input state.

In this dissertation, we use MDNs to implement SDSMMs for two reasons. First, neural networks are regarded as universal function approximators [9]. As a result, a neural network can learn a state-to-measurement space mapping and how states correlate with measurement bias and noise, fulfilling requirements #1 and #2. Second, MDNs also predict a set of parameters that can completely describe a probability distribution conditioned upon some input, which fulfills requirement #3. In the context of SDSMMs, the input to an MDN is a combined state $\boldsymbol{\lambda}$. Once given a combined state, the MDN outputs a set of parameters that describes a conditional measurement probability distribution. For example, if the output parameters describe a Gaussian distribution, the Gaussian mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ (or common variance σ^2) represent the state-dependent expected measurement (which inherently models measurement bias) and measurement noise, respectively. However, suppose the output parameters describe a mixture of Gaussian distributions. In that case, the mixture coefficients α, Gaussian means $\boldsymbol{\mu}$, and Gaussian covariances $\boldsymbol{\Sigma}$ are used to form a more complex multi-modal distribution, which inherently describes the measurement bias and noise.

An MDN learns to predict parameters for measurement probability distributions through training on sensor measurements that were observed at corresponding states. Let $\boldsymbol{\mathcal{D}} = \{(\mathbf{z}_i, \boldsymbol{\lambda}_i)\}_{i=1}^{I}$ denote a sensor measurement $\mathbf{z}_i$ that was observed at a combined state $\boldsymbol{\lambda}_i$. To train an MDN, we first combine the negative log-likelihood function (2.3) and the MDN equation (2.4) to form a loss function for our model:

$$\mathcal{E}(\boldsymbol{\mathcal{D}}, \mathbf{w}) = \sum_{i=1}^{I} \Big(- \ln \Big\{ p(\mathbf{z}_i \mid \Theta_i) \Big\} \Big) = \sum_{i=1}^{I} \left(- \ln \left\{ \sum_{k=1}^{m} \alpha_k(\boldsymbol{\lambda}_i) \phi_k(\mathbf{z}_i | \boldsymbol{\lambda}_i) \right\} \right). \tag{2.13}$$

Then we use a Stochastic Gradient Descent-based method to optimize the neural network parameters $\mathbf{w}$ such that they minimize the negative log-likelihood (2.13) of observing the

Table 2.1: Time complexities required to compute all parameters for a distribution.

MDN Formulation	Parameter	Time Complexity
Both	α	$\mathbf{O}(\text{ m })$
	$\boldsymbol{\mu}$	$\mathbf{O}(\text{ m })$
Common Variance Only	σ	$\mathbf{O}(\text{ m })$
Full Covariance Only	$\boldsymbol{\Sigma}$	$\mathbf{O}(\ m \cdot s^{2.4}\ )$
	$L(\cdot)$	$\mathbf{O}(\text{ m })$
	$D(\cdot)$	$\mathbf{O}(\text{ m })$

data $\boldsymbol{\mathcal{D}}$. We note that the loss function (2.13) works for both Gaussian functions (2.5) and (2.6). One only needs to use the appropriate Gaussian function for $\phi_k(\mathbf{z}_i|\boldsymbol{\lambda}_i)$.

To optimize the network parameters, we also need to compute the derivatives of the error $\mathcal{E}$ with respect to the parameters $\mathbf{w}$ in the neural network. We derived a set of equations that compute such derivatives in Appendix B.

2.5 Analyzing the Complexity of Mixture Density Networks

Many robotics applications require distributions with multi-dimensional covariance matrices. For example, we may require 2D, 3D, or 6D covariance matrices to represent the measurement uncertainty for range-and-bearing or LiDAR sensors. However, employing full covariance matrices instead of 1D variances increases the time complexity during inference[4] and the dimension of the network output vector $\mathbf{y}$. Therefore, we discuss how the dimension of $\mathbf{y}$ and the time complexity for inference increase as we increase the number of components and the distribution dimension for Gaussian distributions with full covariance matrices. We also provide a similar discussion for Gaussian distributions with common variances for comparison.

[4]The time complexity for multi-dimensional covariance matrices also increases during back-propagation. However, inference during the runtime tends to be more time-sensitive.

2.5.1 Dimension Growth of the Neural Network Output Vector

The dimension of the network output vector $\mathbf{y}$ for an MDN with a full covariance matrix or a common variance depends on the number of components m and the dimension size s of the distribution. An MDN that employs the full covariance (as described in (2.6)) requires a network output vector $\mathbf{y}$ with $(1+\frac{s(s+3)}{2})\times m$ dimensions. We can observe that the dimension of $\mathbf{y}$ exhibits a linear growth as m increases. However, the dimension of $\mathbf{y}$ exhibits a quadratic growth as s increases. The quadratic growth is caused by the number of values needed to compute the full covariance matrix.

By comparison, an MDN that uses a common variance (as described in (2.5)) requires a network output vector $\mathbf{y}$ with $(s+2)\times m$ dimensions [4]. Like with a full covariance matrix, the dimension of $\mathbf{y}$ grows linearly. However, unlike a full covariance matrix, the dimension of $\mathbf{y}$ also grows linearly as s increases.

Due to linear growths, as both s and m grow, an MDN with common variances requires less memory compared to an MDN with full covariance matrices. However, the MDN may become less expressive when we employ common variances, especially as s increases.

2.5.2 Time Complexity for Computing the Output Parameters

Table 2.1 shows the time complexities required to compute all parameters for an entire distribution. For an MDN with common variances or full covariance matrices, the equations for the mixture coefficients and multivariate Gaussian means remain the same. We calculate all mixture coefficients $\{\alpha_k\}_{k=1}^{m}$ using a sequence of m exponential operations, one summation, and m division operations (shown in (2.7)), which can be performed in $\mathbf{O}(\ m\ )$. To compute a $\boldsymbol{\mu}_k$, we extract a constant number of values from predefined elements within the output vector $\mathbf{y}$ and compute the mean in constant time; see (2.8).

To compute a common variance, we extract the appropriate entry from the output vector $\mathbf{y}$ and apply the exponential function, all of which can be done in constant time. Therefore, we can compute all m common variances in $\mathbf{O}(\ m\ )$.

Calculating a full covariance matrix requires relatively more time due to multiple matrix-matrix multiplications. We can compute the lower unitriangular and diagonal matrices (using (2.11) and (2.12), respectively) in constant time since the arrangement of the entries is fixed. To calculate each covariance matrix $\mathbf{\Sigma}$, we require $\mathbf{O}\left(s^{2.4} \right)$ to perform matrix-matrix multiplication; see (2.10). Therefore, the time complexity for calculating all m $\mathbf{\Sigma}$ is $\mathbf{O}\left(m \cdot s^{2.4} \right)$.

An MDN with common variances also requires less time to compute its GMM parameters than with full covariance matrices. However, as mentioned in Section 2.5.1, an MDN with common variances may be less expressive. Furthermore, technologies such as FPGAs or small systems with integrated graphics processing units can reduce the compute time of MDNs with full covariance matrices, making such models more feasible for real-time applications.

2.6 Simulated Experiment

2.6.1 Dataset

We used a simulated environment with known measurement biases and noises to validate the approach described in this chapter. A simulated environment allowed us to directly specify the true measurement biases and noises for any given state $\boldsymbol{\lambda}$, which is often impractical in the real world. For our simulated dataset, we designed a simulated distance-and-bearing sensor whose measurement biases and noises correlate with the position of a landmark. The measurements came from a single Gaussian distribution. Specifically, let

$$\mathbf{z}_i \sim \mathcal{N}\left(\mathbf{z}_i^* + \boldsymbol{\delta}_i, \mathbf{Q}_i \right), \tag{2.14}$$

where $\mathbf{z}_i = [\rho_i, \phi_i]^\top$ represents a range-and-bearing measurement in polar coordinates, while $\boldsymbol{\lambda}_i = [\lambda_{x,i}, \lambda_{y,i}]^\top$ represents the true Cartesian position of an observed landmark. Both $\mathbf{z}_i$ and $\boldsymbol{\lambda}_i$ are defined in the sensor's coordinate frame. All units of length (ranges and positions) are in meters, while the angles (bearings) are in radians. $\mathbf{z}_i^*$ represents an *a priori*, range-

and-bearing measurement

$$\mathbf{z}_i^* = \begin{bmatrix} \rho_i^* \\ \phi_i^* \end{bmatrix} = \begin{bmatrix} \sqrt{\lambda_{x,i}^2 + \lambda_{y,i}^2} \\ \mathsf{atan2}(\lambda_{y,i}, \lambda_{x,i}) \end{bmatrix}. \tag{2.15}$$

Each *a priori* measurement is unbiased and noise-free. $\boldsymbol{\delta}_i$ denotes a true, state-dependent nonlinear bias observed at a given combined state:

$$\boldsymbol{\delta}_i = \begin{bmatrix} \delta_{\rho,i} \\ \delta_{\phi,i} \end{bmatrix} = \begin{bmatrix} e^{0.06\rho_i^*} - 0.065(\rho_i^* - 3) - 1 \\ \frac{\pi}{180}(0.0055(\frac{180}{\pi}|\phi_i^*|)^2) \end{bmatrix} \tag{2.16}$$

that outputs a state-dependent bias $\boldsymbol{\delta}_i = [\boldsymbol{\delta}_{\rho,i}, \boldsymbol{\delta}_{\phi,i}]^\top$. The true range bias $\boldsymbol{\delta}_{\rho,i}$ and true bearing bias $\boldsymbol{\delta}_{\phi,i}$ are affected by the true distance ρ_i^* and bearing ϕ_i^* of a landmark, respectively. Finally, $\mathbf{Q}_i$ represents a true, state-dependent nonlinear measurement covariance noise and is also a function of a combined state $\boldsymbol{\lambda}_i$:

$$\mathbf{Q}_i = \begin{bmatrix} \boldsymbol{\sigma}_{\rho,i}^2 & \boldsymbol{\sigma}_{\rho,i} \cdot \boldsymbol{\sigma}_{\phi,i} \\ \boldsymbol{\sigma}_{\phi,i} \cdot \boldsymbol{\sigma}_{\rho,i} & \boldsymbol{\sigma}_{\phi,i}^2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma}_{\rho,i} & 0 \\ 0 & \boldsymbol{\sigma}_{\phi,i} \end{bmatrix} \times \begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{\sigma}_{\rho,i} & 0 \\ 0 & \boldsymbol{\sigma}_{\phi,i} \end{bmatrix}, \tag{2.17}$$

where $\boldsymbol{\sigma}_{\rho,i}$ and $\boldsymbol{\sigma}_{\phi,i}$ are the true range and bearing noises, respectively. We compute the range and bearing standard deviations as

$$\boldsymbol{\sigma}_{\rho,i} = 0.001063 + 0.0007278\rho_i^* + 0.0035\left|\frac{\phi_i^*}{0.5236}\right| \cdot \rho^{1.5} + 0.0008(\rho_i^*)^2, \tag{2.18}$$

$$\boldsymbol{\sigma}_{\phi,i} = 1.5 - \ln(3\rho_i^* + 0.5) + \left(0.8 + 0.4\left|\frac{\phi_i^*}{0.5236}\right|\right)(\rho_i^*)^{\frac{2}{3}}, \tag{2.19}$$

where both noises are affected by the true range and bearing measurements. The equations for the true bias and noise were randomly created. However, we purposely included polynomial, exponential, and logarithmic functions to create a complex model.

To create our simulated dataset, we first sampled 25,000 states using a uniform distribution. Then we computed a true range-and-bearing measurement $\mathbf{z}_i^*$ for each combined state $\boldsymbol{\lambda}_i$ using (2.15). Afterward, for each $\mathbf{z}_i^*$, we computed a true measurement bias and covariance noise using (2.16) and (2.17), respectively. We created a biased, noisy measurement using (2.14). Finally, we created the training dataset $\boldsymbol{\mathcal{D}} = \{(\mathbf{z}_i, \boldsymbol{\lambda}_i)\}_{i=1}^{I}$ by combining the measurements $\mathbf{z}_i$ with their corresponding states $\boldsymbol{\lambda}_i$.

2.6.2 Experimental Setup

The MDN was a multilayer perceptron with five layers. The first four layers had 35 units per layer and used the ReLU activation function, while the last layer had five units and outputted the vector $\boldsymbol{\gamma}$. The vector was used to compute a state-dependent expected measurement $\boldsymbol{\mu}$ and measurement noise $\boldsymbol{\Sigma}$ (see (2.8) and (2.10), respectively). PyTorch 1.6.0 [35] was used to implement our MDN. We trained the MDN using the negative log-likelihood loss function (2.13) and Adam optimizer [20, 43].

This MDN was trained for 50 epochs with a learning rate of 5×10^{-3} and an L2 regularization of 5×10^{-4}. We also multiplied the learning rate by 0.1 whenever the loss plateaued. We used cross-validation to determine the learning rate, number of epochs, learning schedule, and amount of L2 regularization.

2.6.3 Simulated Dataset Results and Discussion

We analyzed the simulated experiments to determine how well our proposed method predicts measurement distributions accurately. First, we used Kullback-Leibler (KL) divergence [24, 23] to measure the difference between the MDN predicted distributions and the true distributions created in Section 2.6.1. Figure 2.3 illustrates the predicted distributions become more accurate as training progressed.

The we visually compared ten random MDN predictions $(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ against their corresponding true distributions $\mathcal{N}(\mathbf{z}_i^* + \boldsymbol{\delta}_i^*, \mathbf{Q}_i^*)$. Figure 2.4 illustrates that the predicted distributions

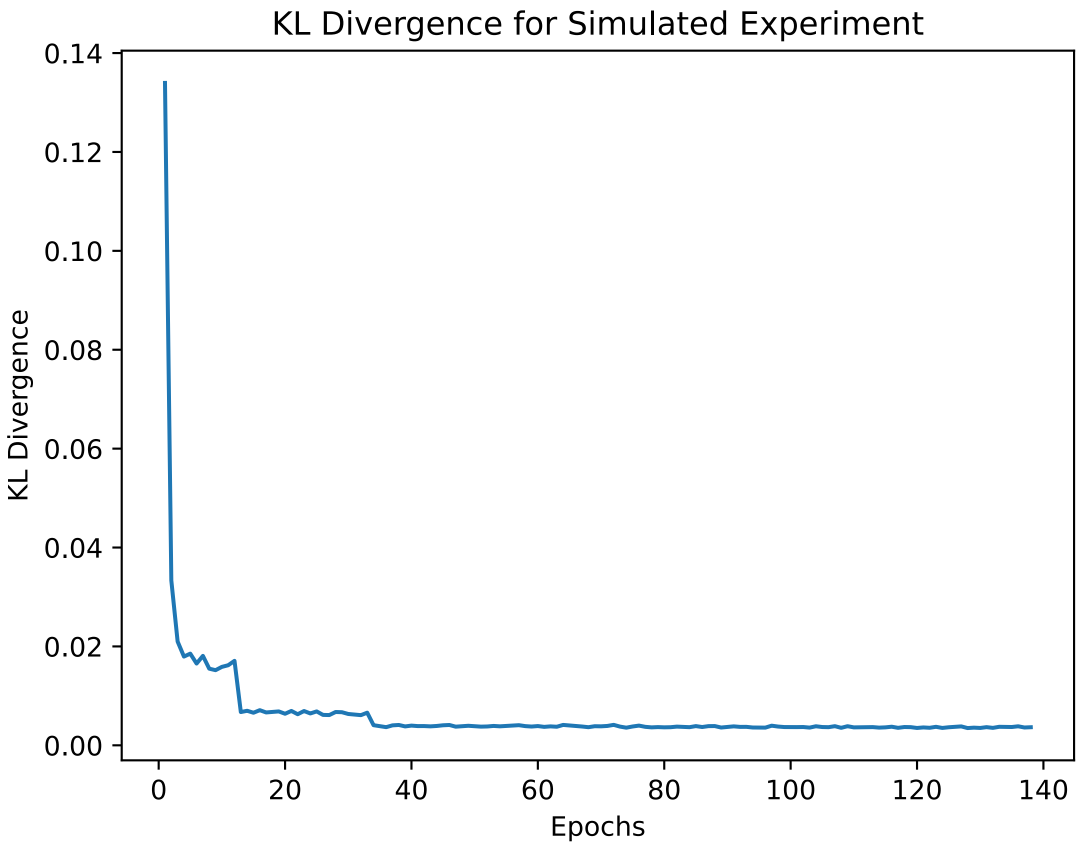


Figure 2.3: KL Divergence for the simulated distance-and-bearing sensor model. During training, the Kullback-Leibler (KL) divergence between the MDN predicted distributions and the simulated true distributions decreased as the epoch increased. As the predicted distributions get closer to the true distribution, the KL divergence score goes closer towards zero.

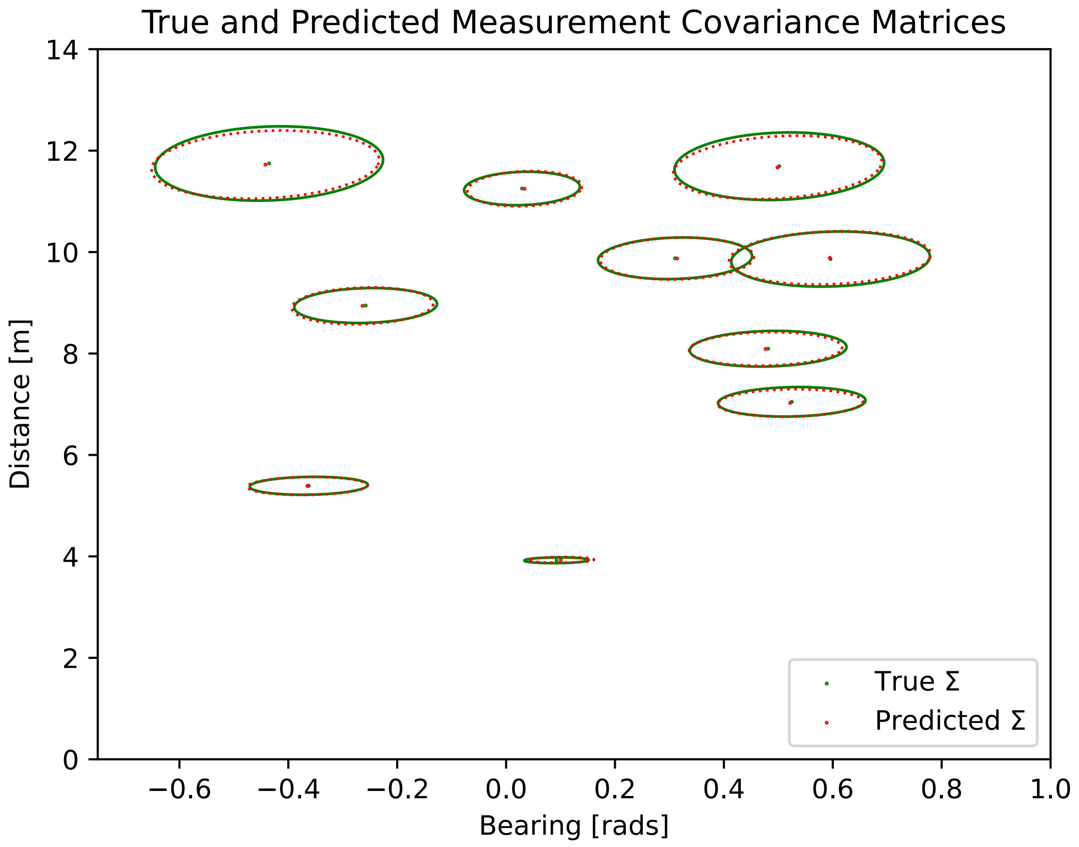


Figure 2.4: True and predicted measurement covariance matrices for the simulated distance-and-bearing sensor model. As the true state-dependent, measurement distributions $\mathcal{N}(\mathbf{z}_i^* + \boldsymbol{\delta}_i^*, \mathbf{Q}_i^*)$ (green dots and solid ellipses) varies, the mixture density network accurately predicted corresponding distributions $(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ (red dots and dotted ellipses). We graphed the covariances at three (3) standard deviations.

overlap the true distributions, implying the MDN learned to predict the expected measurement (which includes measurement bias) and noise accurately.

2.7 Real-World Experiment I: Distance-and-Bearing Sensor Model

2.7.1 Dataset

We used the University of Toronto's Institute for Aerospace Studies (UTIAS) multi-robot cooperative localization and mapping (MR.CLAM) dataset [25] to compare the performance of our proposed method with an *a priori* measurement model. The MR.CLAM dataset was collected during nine separate experiments in a static 15m x 8m indoor, open-space environment. Each experiment had five iRobot Creates and 15 static landmarks, and the landmarks were randomly placed throughout each experiment. All five robots arbitrarily and simultaneously moved throughout a course. The duration of each experiment ranged from 15 minutes to 70 minutes. The robots controlled their movements using velocity commands, which were issued at a rate of 67 Hz. Each robot used its own onboard, monocular vision system to 1) identify each landmark via barcode identifiers (the horizontal striped patterns) and 2) measure the range and bearing of each landmark. In addition, a motion capture system, which was composed of ten Vicon cameras, tracked the global position of each landmark and the global pose of each robot during each experiment.

The MR.CLAM dataset contains timestamped logs that were recorded by the motion capture system and the five iRobot Create robots. The motion capture system recorded timestamped logs of the global poses of each robot and the global position of each landmark during each experiment. In addition, each robot recorded timestamped logs of its odometry commands and measurements. Each odometry command is composed of a linear velocity and an angular velocity. Each robot measurement comprises a range, a bearing, and a signature. The signature is a numeric identifier that specifies which landmark or robot was observed. Although robots observed both landmarks and other robots, we only used landmark-observed measurements for our purposes. Here, each landmark barcode, the black horizontal lines along the upper half of the landmark cylinders, encodes the signature of a landmark, which is used for data association.

The motion capture system and each robot recorded their logs independently of each other. Therefore, the logs of each robot and the motion capture have different timestamps.

2.7.2 Experimental Setup

2.7.2.1 Training Datasets

We divided the nine MR.CLAM experiments (courses) into a training set, a validation set, and a testing set. We randomly selected courses 2, 3, 8, and 9 for training, courses 5 and 6 for validation, and courses 1, 4, and 7 for testing. During training, validation, and testing, only measurements and ground-truth states for a particular robot were used to train, validate, and test its measurement models.

The combined state $\boldsymbol{\lambda}$, the input to each MDN, was defined as

$$\boldsymbol{\lambda}_i = \begin{bmatrix} \lambda_{x,i} \\ \lambda_{y,i} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}^{-1} \begin{bmatrix} m_x \\ m_y \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix}, \tag{2.20}$$

where $\boldsymbol{\lambda}_i$ represents the spatial relationship between a robot and an observed landmark. From initial investigations, we found that this combined state representation helped the MDNs learn a range-and-bearing SDSMM for each robot's camera sensor. This representation is advantageous because the learned measurement models are not limited to any specific MR.CLAM experiment setup.

In (2.20), $\mathbf{x}_i = [x_i, y_i, \theta_i]^\top$ denotes the true global robot pose and $\mathbf{m}_{x,y} = [m_x, m_y]^\top$ denotes the ground-truth global position of an observed landmark. The robot pose and landmark position are defined in a global coordinate frame. The ground-truth robot poses and landmark positions came from the logs of the motion capture system.

The target of the SDSMMs was the range-and-bearing measurements $\mathbf{z}_i$, which were gathered by each robot's camera.

2.7.2.2 State-Dependent Sensor Measurement Model

We trained five MDNs using the MR.CLAM dataset (one MDN per robot). Each MDN predicted parameters for a multivariate range-and-bearing Gaussian distribution with a full covariance matrix. Let $g(\boldsymbol{\lambda}; \mathbf{w})$ denote an MDN that predicts parameters $\Theta = [\ \boldsymbol{\mu}, \boldsymbol{\Sigma}\ ]$ for a range-and-bearing measurement probability distribution. We used (2.8) and (2.10) to compute the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, respectively. Each robot has its own MDN because each robot had a different measurement bias and noise.

All MDNs had the same architecture. The architecture was a multi-layer perceptron (MLP) and was composed of five layers. Each of the first four layers had 35 units and used a ReLU activation function [33]. In the fifth layer, the number of units was five (5): two (2) units correspond to the mean, and three (3) units correspond to the full covariance matrix.

During training, we used the Adam optimizer with a learning rate of 10^{-3}, but the rest of the optimizer parameters remained at their recommended values [20, 43]. We used the negative log-likelihood loss (2.13) to train each network for 30 epochs. The networks were regularized using an L2 norm with a weight of 10^{-3}. We decreased the learning rate by a factor of 0.2 whenever the loss plateaued for more than five epochs. We used cross-validation to choose the network architecture, learning rate, number of epochs, amount of regularization, and learning rate decay. PyTorch 1.6.0 [35] was used to implement our MDNs. All MDNs were trained using Intel® AI DevCloud, which has a cluster of Intel Xeon processors.

2.7.2.3 Baseline

We used a traditional range-and-bearing sensor measurement model to compare with our SDSMM. We defined the *a priori* measurement model as

$$\mathbf{z}_i = h(\mathbf{x}_i, \mathbf{m}) + \boldsymbol{\sigma}, \tag{2.21}$$

where $h(\mathbf{x}_i, \mathbf{m})$ is the measurement function and $\boldsymbol{\sigma}$ is the measurement noise. The measurement function was defined as

$$\mathbf{z}_i = \begin{bmatrix} \rho_i \\ \phi_i \end{bmatrix} = \begin{bmatrix} \sqrt{(x_i - m_x)^2 + (y_i - m_y)^2} \\ \text{atan2}(y_i - m_y, x_i - m_x) - \theta_i \end{bmatrix}, \tag{2.22}$$

where $\mathbf{x}_i = [x_i, y_i, \theta_i]^\top$ is the global robot pose and $\mathbf{m} = [m_x, m_y]^\top$ is the global position of the observed landmark at the moment the sensor measurement $\mathbf{z}_i$ was observed.

The measurement noise $\boldsymbol{\sigma}$ was fixed, which is the typical for traditional measurement models. We assumed the measurement noise came from a zero-mean Gaussian distribution: $\boldsymbol{\sigma} \sim \mathcal{N}(0, \mathbf{Q})$. $\mathbf{Q}$ denotes the covariance of the measurement noise and was computed as $\mathbf{Q} = \mathbf{Cov}(\Delta\mathbf{Z})$ for each robot. Here, $\Delta\mathbf{Z}$ is the difference between the observed sensor measurements and the outputs of the measurement function (2.22). We obtained the measurement function outputs by computing $h(\mathbf{x}_i, \mathbf{m})$ for each timestamp i.

2.7.3 Results and Discussion

In the results below, we used log-likelihood and measurement error to evaluate the learning experiments. Each measurement error was the difference between a predicted measurement and observed sensor measurement. The predicted measurements came from either an *a priori* measurement model or an SDSMM. The observed sensor measurements came from the dataset described in Section 2.7.2.1.

Table 2.2: Test set log-likelihood means for *a priori* models and SDSMMs. The best mean is emphasized for each robot.

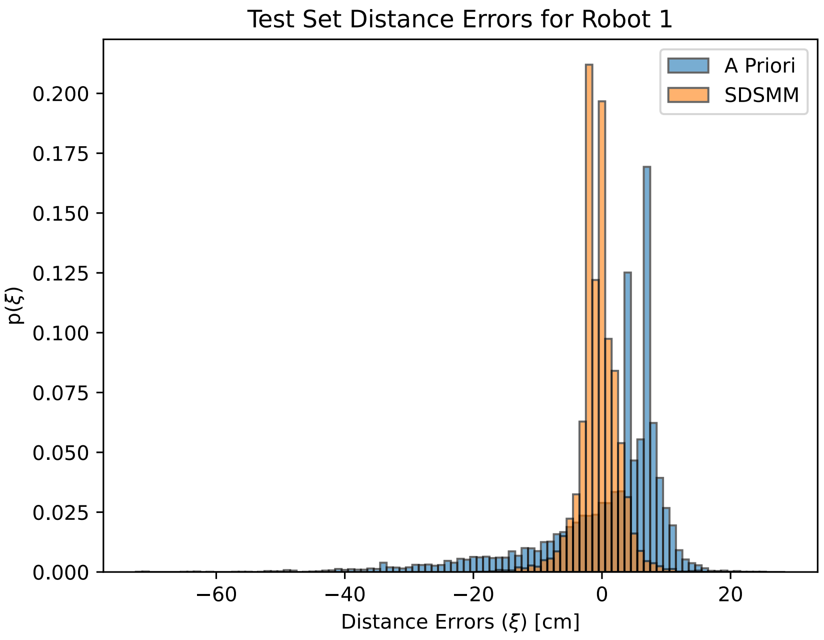
	A Priori	SDSMM
R1	3.07	**4.81**
R2	2.70	**5.44**
R3	2.30	**5.33**
R4	1.83	**5.31**
R5	3.33	**5.27**

Table 2.3: Test set distance error mean and standard deviation for the *a priori* models and SDSMMs. The best mean and standard deviation is emphasized for each robot. The values below are in centimeters.

	A Priori	SDSMM
R1	0.5 ± 10.5	$\mathbf{-0.5} \pm 3.1$
R2	-0.2 ± 14.4	$\mathbf{-0.2} \pm 2.7$
R3	-4.5 ± 13.5	$\mathbf{-0.3} \pm 2.6$
R4	-1.3 ± 15.3	$\mathbf{-0.2} \pm 2.7$
R5	-3.8 ± 14.4	$\mathbf{-0.7} \pm 3.0$

Table 2.4: Test set bearing error mean and standard deviation for the *a priori* models and SDSMMs. The best mean and standard deviation is emphasized for each robot. The values below are in degrees.

	A Priori	SDSMM
R1	$\mathbf{-0.1} \pm 1.4$	-0.2 ± 1.3
R2	$\mathbf{0.1} \pm 0.8$	0.5 ± 0.7
R3	-0.5 ± 0.7	$\mathbf{0.2} \pm 0.7$
R4	-0.7 ± 0.9	$\mathbf{-0.2} \pm 1.0$
R5	$-0.2 \pm 0.7 *$	$0.2 \pm 0.7 *$



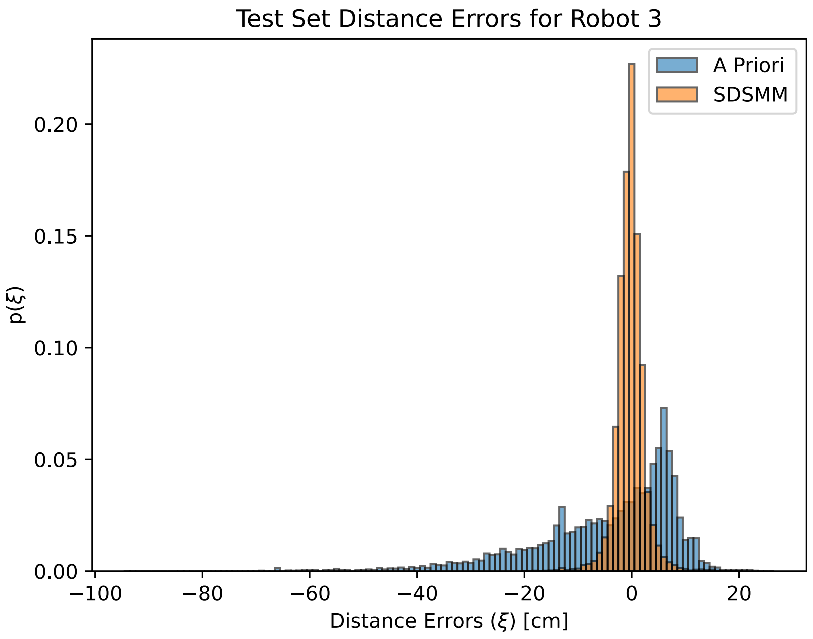
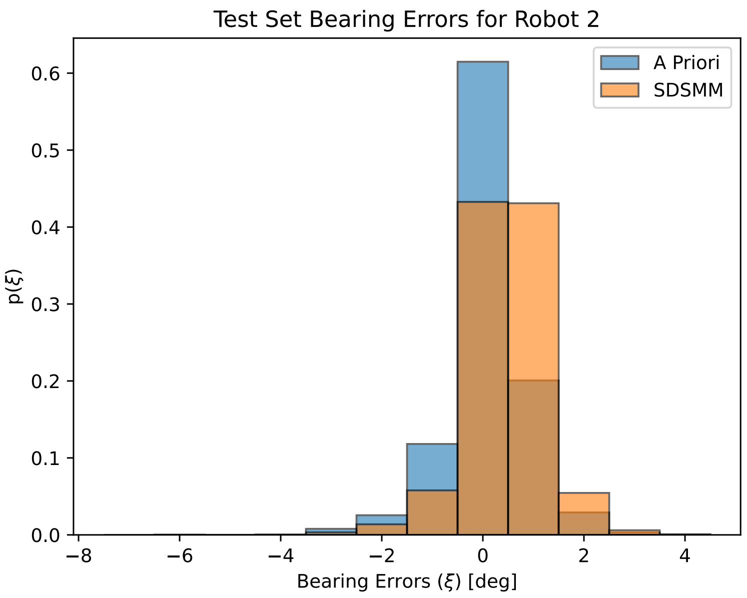


Figure 2.5: Test set distance errors for robots 1 and 3 using the *apriori* models and SDSMMs. The distance measurements had variable amounts of bias for all courses. Since an SDSMM can model bias, SDSMM had less distance prediction error than its *a priori* counterpart. The top and bottom graphs represent the *a priori* models with the lowest and highest error means and standard deviations, respectively.



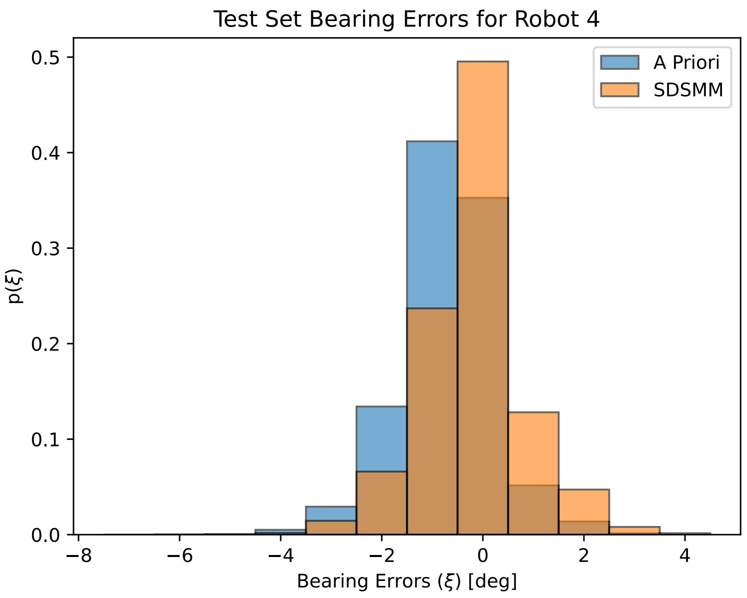


Figure 2.6: Test set bearing errors for robots 2 and 4 using the *apriori* models and SDSMMs. Since the bearing measurements had a minimal amount of bias, both models achieved similar prediction errors. The top and bottom graphs represent the *a priori* models with the lowest and highest error means and standard deviations, respectively.

Generally, the training results showed that the SDSMMs predicted distribution parameters that fit the data better than *a priori* models. The mean log-likelihoods for all SDSMMs were greater than their *a priori* counterpart (Table 2.2). SDSMMs had as much or, in most cases, less mean distance error than *a priori* models. The standard deviation of the distance errors were also one magnitude lower than the *a priori* models (Table 2.3 and Figure 2.5). Therefore, the SDSMM distance error distributions were centered near zero. However, the *a priori* distance error distributions resembled skewed distributions with modes relatively further away from zero. Unlike distance, the prediction errors of both models for bearing measurements were similar (Table 2.4 and Fig. 2.6). Both models had similar performance because the bias in bearing measurements was small.

Specifically, the training results showed that RDO models predicted distributions that fit the data better than baseline models. The mean log-likelihoods for all RDO models were greater than their *a priori* counterpart (Table 2.2). RDO models had as much or, in most cases, less mean distance bias than *a priori* models. The standard deviation of the distance biases were also one magnitude lower than the *a priori* models (Table 2.3 and Figure 2.5). Unlike distance, the prediction errors of both models for bearing measurements were similar (Table 2.4 and Figure 2.6).

2.8 Real-World Experiment II: Pole Detection Sensor Model

This section presents a state-dependent sensor measurement model that learned the measurement error for a LiDAR-based pole detection sensor model. Specifically, we used the LiDAR-based pole extractor and mapping module proposed in Schaefer *et. al* [47, 46] as our baseline. We refer to the baseline as LPD (LiDAR-based Pole Detector). In our experiments, we combined the LPD and our SDSMM. This combination is referred to as LPD+SDSMM. The SDSMM dynamically compensates for the state-dependent bias and noise of LPD. We also evaluated LPD and our proposed LPD+SDSMM using the University of Michigan's

North Campus Long-Term Vision and LiDAR Dataset [6] to determine how our proposed method performs across various times of day, changes in scenery, and weather conditions.

2.8.1 Dataset

We used the University of Michigan's North Campus Long-Term Vision and LiDAR Dataset [6] to compare the performance of our proposed method with a LiDAR-based sensor model. The dataset was collected at the University of Michigan over the course of 27 sessions. All sessions were collected within 15 months, were gathered bi-weekly, and spanned across all four seasons, various times of day (morning through the evening), different weather conditions, and dynamic environments. In total, the dataset contains 34.9 hours of data covering 147.4 kilometers of robot trajectory. During each session, a human manually drove a Segway robot indoors and outdoors, collecting data using a suite of sensors. The sensors were a Ladybug3 omnidirectional camera, a Velodyne HDL-32E 3D LiDAR, two Hokuyo planar LiDARs, and an inertial measurement unit (IMU), a single-axis fiber optic gyro (FOG), a consumer-grade global positioning system (GPS), and a real-time kinematic (RTK) GPS.

The authors also generated ground-truth robot poses for all 27 sessions using simultaneous localization and mapping (SLAM).

2.8.2 Experimental Setup

2.8.2.1 Baseline

The method proposed in Schaefer *et. al* [47] is composed of a LiDAR-based pole extractor, a mapping module, and a localization module. The input to the pole extractor is a set of registered 3D LiDAR scans, while the output is the position, width, and score of a pole. The set of registered 3D LiDAR scans is a collection of scans over a short distance; the distance used for the NCLT dataset was 1.5 meters. A pole position is represented by a 2D Cartesian point.

The pole extractor performs three sequential steps to produce its output. The extractor first builds a 3D occupancy map of the scanned space. Next, the extractor applies a pole feature detector for every voxel in the occupancy map to find potential poles. Finally, the extractor regresses the pole map to a collection of pole positions, widths, and scores.

The mapping module uses the pole extractor to build a global map using all the scans in a dataset. Since building a map for a large space may be memory intensive, the authors build smaller maps over short trajectories. The smaller maps are then aggregated into one global feature-based map. In some instances, the adjacent small maps overlap, causing a pole landmark to appear multiple times. In such cases, the authors compute a weighted position and width of a pole. The weights correspond to the pole scores, which are outputted by the pole extractor.

2.8.2.2 Training Datasets

We performed three experiments using the 27 sessions from the NCLT Dataset [6]. The experiments were called “clear days”, “1Day”, and “9Days.” The purpose of each experiment was to measure how well this LiDAR-based SDSMM generalizes to the other sessions, all of which had varying weather and sky conditions, times of day, and (indoor or outdoor) trajectories. For the “clear days” experiment, we selected sessions 2012-02-02, 2012-02-04, 2012-02-05, 2012-03-17 for training and sessions 2012-03-25 and 2012-11-17 for testing. These six (6) sessions represented days where there was no snow or foliage, the skies were clear, and the data was collected during the morning, mid-day, and afternoon.

Together, the 1Day and 9Days experiments represented scenarios in which we used sequential, non-consecutive days to collect data to train a model. For 1Day, we selected session 2012-01-08 for training, which is the first session in the dataset. For the 9Days experiment, we fine-tuned the 1Day model using sessions 2012-01-15 through 2012-03-17 for training. The testing set for both experiments was session 2012-03-25. These training sessions had various weather conditions (visible or no snow), time of day, vegetation (visible or no foliage), and sky conditions.

The training and testing sets were comprised of combined states and pole position measurement errors. We modified the pole extractor and mapping module from Schaefer *et al.* [47] to generate the combined states and measurement errors for training and testing. Here, we defined the combined state as the observed parameters for a pole in a local map: $\boldsymbol{\lambda}_i = \{\lambda_x, \lambda_y, \lambda_a, \lambda_s\}_i$. The pole position measurement error was defined as the difference between pole positions in the global and local maps: $\boldsymbol{\xi}_i = \boldsymbol{\lambda}^G_{i,[x,y]} - \boldsymbol{\lambda}_{i,[x,y]}$. Since the poles do not have identifiers, we followed Schaefer *et al.* [47] in using a KD-tree algorithm [30] to associate observed poles (in local maps) with poles in the global map. We also ensured that the pole positions $\boldsymbol{\lambda}_{i,[x,y]}$ and $\boldsymbol{\lambda}^G_{i,[x,y]}$ were in the same coordinate frame during training.

2.8.2.3 State-Dependent Sensor Measurement Model

We trained one MDN using the data created in Section 2.8.2.2. The MDN predicted parameters for a bi-variate Gaussian distribution with a full covariance matrix. Let $g(\boldsymbol{\lambda}; \mathbf{w})$ denote the MDN that predicts parameters $\Theta = [\ \boldsymbol{\mu}, \boldsymbol{\Sigma}\ ]$. Unlike the distance-and-bearing model presented in Section 2.7, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ represent pole position bias and noise, respectively.

The architecture for this MDN was an MLP that was composed of five layers. Each of the first four layers had 50 units and used the ReLU activation function [33]. Like the range-and-bearing model, the fifth layer had five (5) output units: two (2) units corresponded to the mean, and three (3) units corresponded to the full covariance matrix.

We trained the MDNs using the Adam optimizer [20, 43] with a learning rate of 10^{-3}. We used the negative log-likelihood loss (2.3) to train each MDN. The "clear days", "1Day", and "9Days" models trained for 28, 27, and 15 epochs, respectively. The MDNs were regularized using an L2 norm with a weight of 5×10^{-3}. Cross-validation was used to choose the network architecture, learning rate, number of epochs, and amount of regularization. We implemented

Table 2.5: Test set log-likelihood summary for the LPD and SDSMM-augmented LPD models. The summary states the 1st quantile, median, 3rd quantile, and maximum value for each model. These values were derived from the test set. The best log-likelihoods are emphasized.

	LPD	LPD-CD	LPD-1Day	LPD-9Days
1st Quantile	-2.28	-0.37	-0.35	$\mathbf{-0.24}$
Median	-2.26	0.32	$\mathbf{0.73}$	0.43
3rd Quantile	-2.25	1.01	$\mathbf{1.33}$	1.04

Table 2.6: Landmark position error mean and standard deviation (in centimeters) for the LPD and SDSMM-augmented LPD models. These values were derived from the test set.

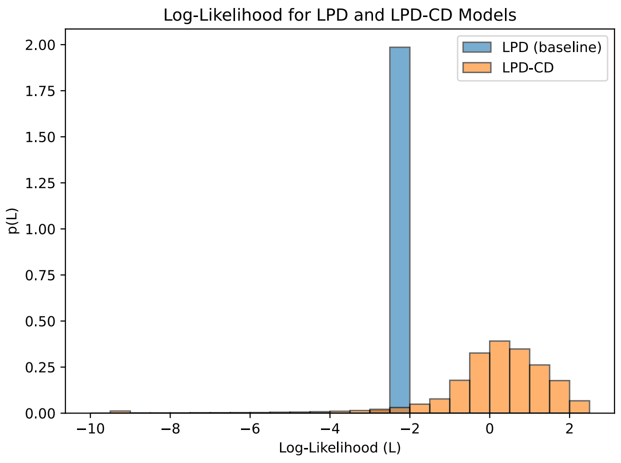
	LPD	LPD-CD	LPD-1Day	LPD-9Days
X Error Mean $\pm$ Stdv.	-0.2 ± 20.0	-1.3 ± 20.1	1.5 ± 20.9	0.7 ± 21.0
Y Error Mean $\pm$ Stdv.	0.3 ± 20.1	0.5 ± 23.9	-0.7 ± 24.0	-1.1 ± 24.5

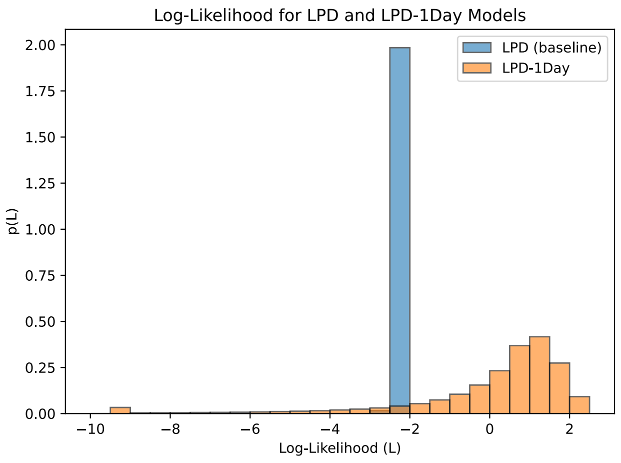
the MDN using PyTorch [35] and trained the network using an AMD Ryzen 7 3800X 8-Core Processor.

2.8.3 Results and Discussion

Like in Section 2.7.3, we used the log-likelihood and expected measurement error to compare the performance of the LPD and SDSMM-augmented LPD models. Since we do not know the true measurement error, log-likelihood helped us determine how well the output position distributions fit the data. We used the expected measurement error to measure the difference between the ground-truth landmark positions and the expected measurements (from either measurement model). The landmark locations came from the global map generated using the NCLT dataset and the LPD baseline. The results below were generated using the data from the test set.

We observed five findings from the learning results. First, SDSMM-augmented LPD models produced measurement distributions that fit the test set better than the standalone LPD. Figure 2.7 shows that the LPD log-likelihoods peaked between -3 and -2. Although





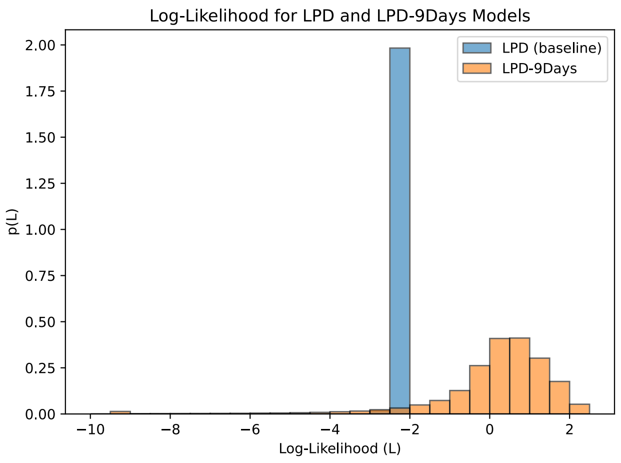


Figure 2.7: Test set log-likelihood distributions for the LPD model and SDSMM-augmented LPD models. Higher log-likelihoods imply a better fit to the test dataset.

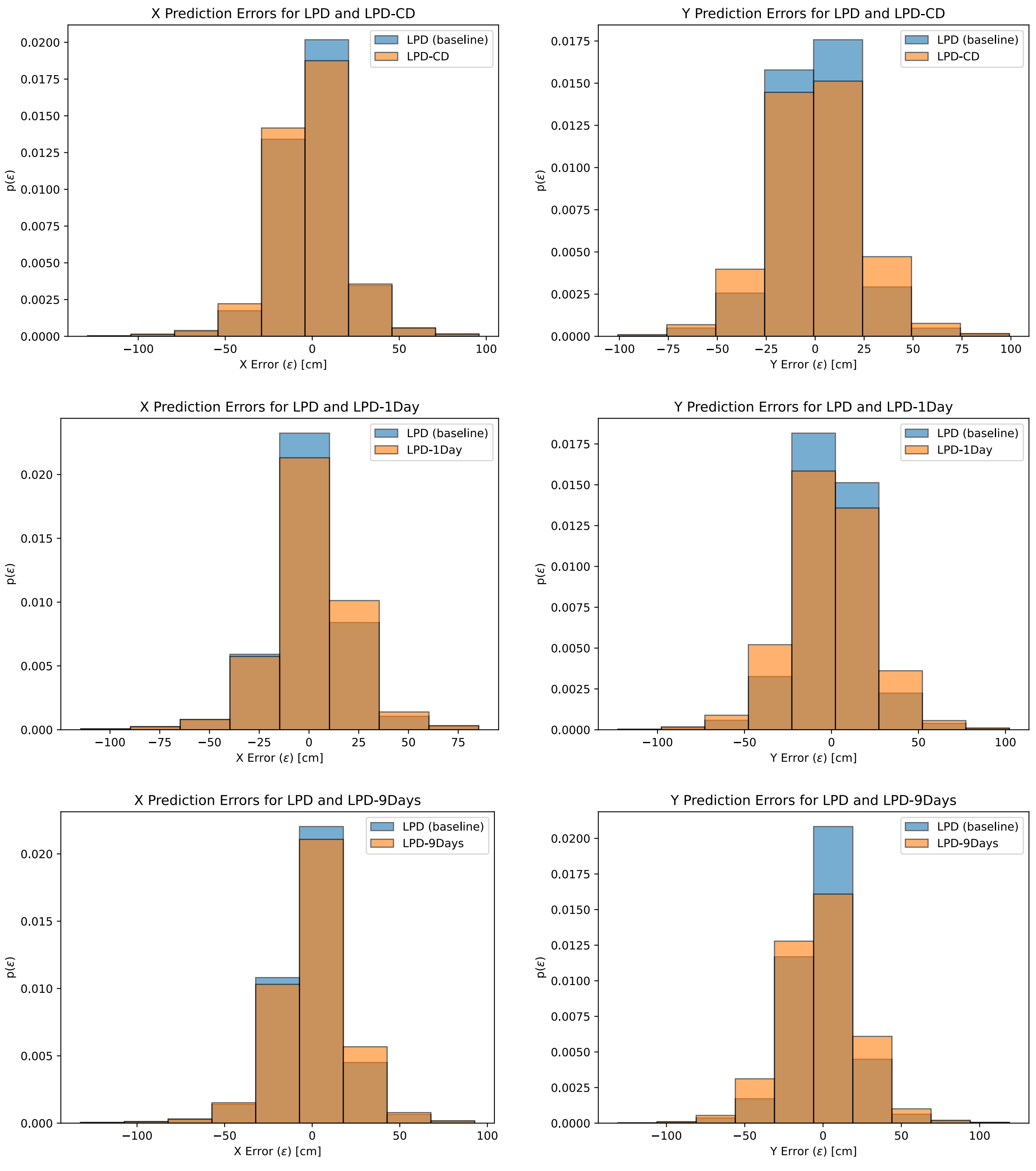


Figure 2.8: Test set prediction errors for the (x,y) output of the LPD and SDSMM-augmented LPD models. This figure compares the expected landmark position errors for the LPD and SDSMM-augmented LPD models for the entire test set. Both models seem to have similar expected measurement errors for the x and y measurements. The measurement errors are in centimeters.

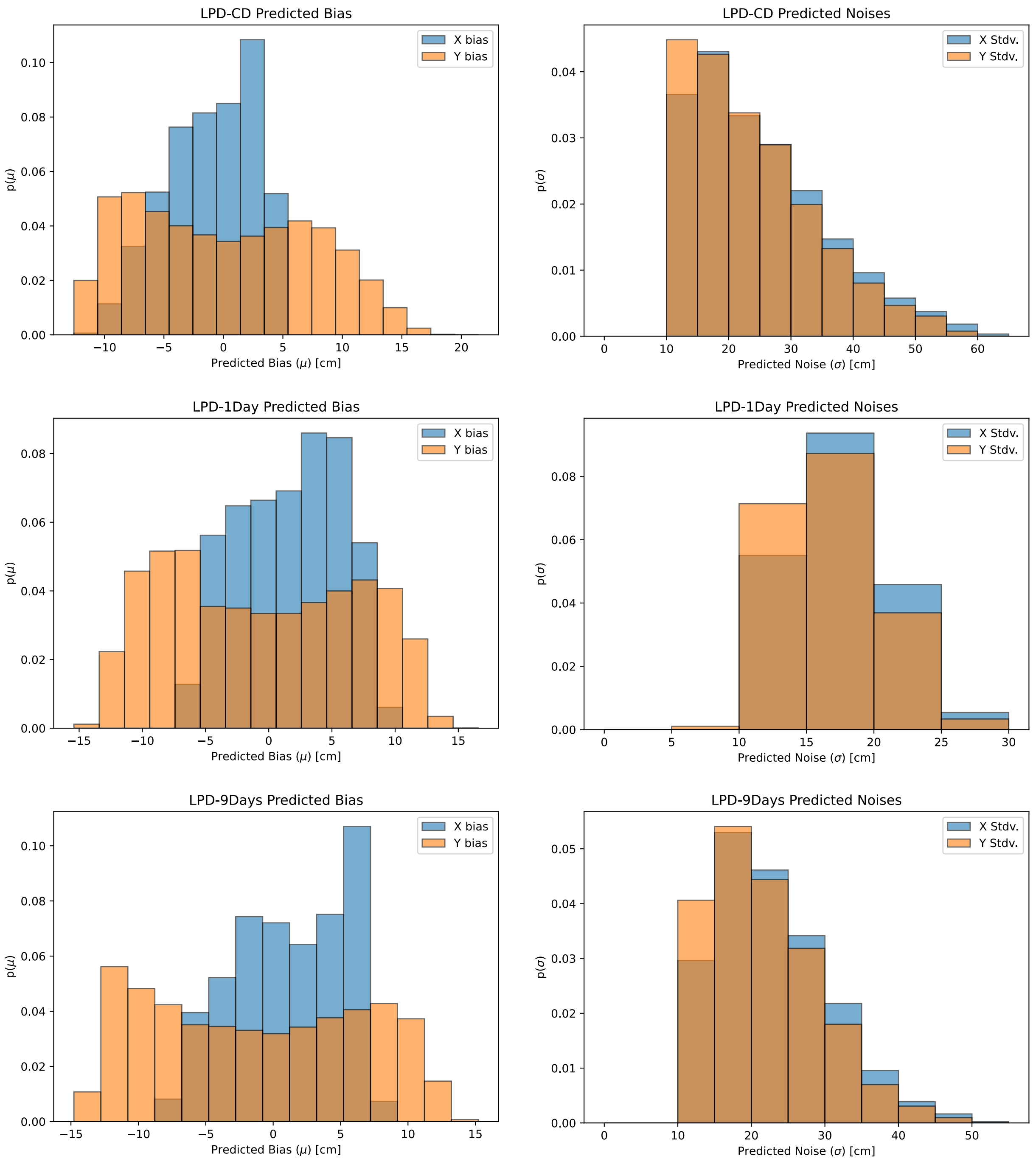


Figure 2.9: Test set SDSMM predicted bias and noise for (x,y) output. This figure displays the SDSMM predicted measurement bias and noise for the entire test set. The range and frequency of the predicted biases and noises appear similar for the x and y directions. Many of the predicted measurement noises are also smaller than the $\sigma^2 = 150$-centimeter measurement noise that Schaefer *et al.* [47] in their implementation.

the SDSMM-augmented LPD log-likelihoods do not peak as much as LPD, we observed that most SDSMM log-likelihoods were higher than LPD. Furthermore, Table 2.5 quantitatively showed that at least 75% of the LPD+SDSMM log-likelihoods were higher than all of the LPD log-likelihoods. We acknowledge that there is a cluster of log-likelihoods between -10 and -9 for the SDSMM-augmented LPD models. This cluster is NaNs that arose from numerical instabilities, which were caused by near-zero Gaussian probabilities. To ensure these NaNs did not affect the log-likelihood distribution summary (Table 2.5) and figure (Figure 2.7), we set the NaNs to be ln(0.0001), which roughly equates to -9.21.

Our second observation was that the position error for all four models varied as much as 1.5 meters (Figure 2.8). However, whether the high position errors are due to the LiDAR itself or the entire LPD pipeline is unclear. One possible reason for the higher magnitude errors is mismatch during landmark data association. Currently, the LPD baseline does not have a method for uniquely identifying pole landmarks. A second possible reason for high magnitude errors is the quality of the pole extractions. A third reason may be due to errors in the aggregated point clouds.

Our third observation was that the SDSMM predicted measurement noises were smaller than the fixed, isotropic measurement noise of $\sigma^2 = 1.5$ (or $\sigma = 1.2$) meters, which was used in [47] (Figure 2.9). Although our dynamic measurement noise ranged from 10 centimeters to 70 centimeters across the three models, we also observed that the frequency of the measurement noises decreased as their magnitudes increased, implying that the fixed measurement noise in [47] might be overestimated.

Our fourth observation showed that the expected landmark position errors for all four models were comparable. The largest difference amongst the mean landmark position errors amongst the models were less 1.8 centimeters, respectively (Table 2.6). The standard deviation of the position errors also differed less than 4.5 centimeters amongst the models. Figure 2.8 illustrates the position error distributions for the X and Y directions. The top graph in

Figure 2.9 provides insight as to why the position error for both models are similar: most predicted measurement biases lie between -10 and +10 centimeters.

Our final observation showed that the log-likelihoods, predicted biases, and predicted noises for all three models were comparable. Quantitatively, we saw that the three models had similar log-likelihoods and landmark position errors (Tables 2.5 and 2.6, respectively). We also saw that the predicted bias distribution were similar in shape and range (Figure 2.9 left). The frequency of the predicted noises decreased as the noise magnitude increased (Figure 2.9 right). However, the range for the LPD-1Day model was roughly half of the ranges for the other two models.

Chapter 3: Learning State-Dependent Sensor Measurement Models with Limited Real Data

3.1 Introduction

This dissertation seeks to train a model to dynamically and accurately predict state-dependent measurement probability distributions, which quantify measurement bias and noise[5]. Accurately quantifying measurement bias and noise is essential in fundamental robotics applications such as localization and mapping, where a robot uses biased and noisy measurements from one or multiple sensors to infer its state or the state of objects within the environment (such as the locations of landmarks). In Chapter 2, we developed a framework that learns state-dependent sensor measurements models (SDSMMs), which quantify state-dependent measurement bias and noise dynamically. Through simulation, we also demonstrated that an SDSMM predicts state-dependent measurement probability distributions accurately.

Since we presently rely on neural networks to implement SDSMMs, our framework assumes we can use sizeable datasets to train a model. However, gathering sizeable datasets requires multiple resources and tasks, which can be expensive or time-consuming. For example, consider an autonomous car that uses a suite of sensors (such as GPS, cameras, and LiDAR) to map its surrounding area and localize within it. For each sensor on the vehicle, we must collect a sizeable dataset to learn an accurate sensor model. We must also gather additional datasets to learn novel models whenever we obtain additional vehicles or when sensors receive damage, degrade, or are replaced, all of which are inevitable. Even though many vehicles will use sensors from the same manufacturer, we assume each sensor has its

[5]A portion of this chapter was published in [59, 60].

own bias and noise characteristics due to manufacturing differences or differences in the software used to process raw measurements. Therefore, one sensor model may not work out of the box for multiple sensors from the same manufacturer.

The scenario above illustrates that collecting large datasets is challenging, especially as robots become ubiquitous. On the other hand, collecting smaller datasets might be more feasible since collecting smaller datasets might require relatively fewer resources. Unfortunately, neural networks may yield poor performance when trained with limited data. For example, when training with a small dataset, the network might predict accurate outputs for the data in the small training set. However, the learned SDSMM may be inaccurate when applied to data outside of the small training dataset since, for example, the network may not have learned a general mapping between the input states and predicted measurement bias and noise.

Although the discussion above focuses on neural networks and autonomous cars, we emphasize to the reader that this discussion is not limited to those cases. Our discussion also applies to other sensor modeling frameworks that require sizeable training datasets. Furthermore, the discussion applies to other autonomous vehicles (such as aerial, aquatic, and other terrestrial vehicles) and their sensors.

In this chapter, we extend our framework to train SDSMMs when we have limited sensor data, which builds upon our work in [59, 60]. To extend the framework, we leverage transfer learning, which has demonstrated success in various applications where data is limited [34, 56, 50]. The motivation for employing transfer learning is that we are interested in transferring knowledge from a related source task (with source data) to improve the performance of our model on a target task (with target data) [56]. Typically, we learn a related source task using a large dataset that is more accessible or easier to gather compared to the dataset for the target task. Here, we leverage inaccurate physical sensor models to generate arbitrarily large training datasets since such models are generally ubiquitous and describe the measurement generation process. Therefore, this dissertation treats learning an SDSMM for an inaccurate physical model of our real sensor as a source task. By extension, we treat learning an SDSMM for our real sensor as the target task. We also explore methods for regularizing SDSMMs while training on small datasets to reduce model overfit.

In summary, this chapter:

- Learns SDSMMs with limited real data, which addresses one difficulty of collecting sizeable datasets.
- Explores methods for regularizing conditional density estimators such as mixture density networks.

3.2 Generating Artificial Datasets

The data generator creates artificial data for pre-training SDSMMs. The generator is composed of two main components. The first component, a basic measurement model, describes a rudimentary relationship between a combined state $\boldsymbol{\lambda}$ and a sensor measurement $\mathbf{z}$. The second component, a basic error model, describes a relationship between a combined state $\boldsymbol{\lambda}$ and a measurement error $\boldsymbol{\xi}$, which contains bias $\boldsymbol{\delta}$ and noise $\boldsymbol{\sigma}$.

Before discussing the data generator further, it is essential to explain why we cannot use the artificial training dataset alone to learn a sensor measurement model. By design, the generated artificial data is a crude representation of the state-dependent measurement bias and noise for a sensor. Therefore, the learned bias and noise will be crude approximations of the actual bias and noise of the sensor. This design choice is practical for two reasons. First, we typically do not know how measurement bias and noise correlate with states. As a result, we cannot specify accurate bias and noise models for a sensor. Second, we only seek to learn a mapping from the input state space to the output measurement space.

3.2.1 Basic Measurement Model

A basic measurement model expresses a relationship between a combined state and an expected sensor measurement. Given a combined state $\boldsymbol{\lambda}$, a basic measurement model outputs an expected value $\mathbf{z}^*$. The expected value is unbiased and noise-free. Formally, we define a basic measurement model as a function $h_{\mathbf{z}^*}$ that defines a relationship $h_{\mathbf{z}^*}\colon \boldsymbol{\Lambda} \rightarrow \mathbf{Z}^*$, where $\boldsymbol{\Lambda}$ is the set of valid combined states and $\mathbf{Z}^*$ is the set of valid unbiased, noise-free expected measurements. Therefore, the basic measurement model is formulated as:

$$\mathbf{z}^* = h_{\mathbf{z}^*}(\boldsymbol{\lambda}), \tag{3.1}$$

where $\mathbf{z}^* \in \mathbf{Z}^*$ and $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$. We can derive a basic measurement model using an *a priori*, deterministic physical model or heuristics. For example, we can use the physics-based principles of a (software or hardware) range sensor to derive the basic model. Such models include the Euclidean distance between a sensor and an observed object (such as in [51]) or the time of flight for a laser rangefinder.

3.2.2 Basic Bias and Noise Models

The basic bias model expresses a relationship between an expected measurement bias $\boldsymbol{\delta}$ and a combined state $\boldsymbol{\lambda}$. Likewise, a basic noise model expresses a relationship between an expected measurement noise $\boldsymbol{\sigma}$ and a combined state $\boldsymbol{\lambda}$. These models describe the measurement bias and noise that a sensor would observe at a given combined state. Formally, the bias and noise models are defined as functions $h_{\boldsymbol{\delta}}$ and $h_{\boldsymbol{\sigma}}$ that define the relationships $h_{\boldsymbol{\delta}}\colon \boldsymbol{\Lambda} \rightarrow \boldsymbol{\Delta}$ and $h_{\boldsymbol{\sigma}}\colon \boldsymbol{\Lambda} \rightarrow \boldsymbol{\Sigma}$, respectively. Here, $\boldsymbol{\Delta}$ is the set of valid measurement biases, while $\boldsymbol{\Sigma}$ is the set of all valid measurement noises. We formulate the bias and noise models as

$$\boldsymbol{\delta} = h_{\boldsymbol{\delta}}(\boldsymbol{\lambda}), \tag{3.2}$$

$$\boldsymbol{\sigma} = h_{\boldsymbol{\sigma}}(\boldsymbol{\lambda}). \tag{3.3}$$

The outputs of each function can be additive or multiplicative. Like a basic measurement model, the bias and noise models can be defined using *a priori* knowledge or heuristics. However, in simple cases, we can use the manufacturer-reported bias and noise of a sensor to define the respective models.

3.2.3 Generating an Artificial Dataset with Basic Models

We generate an artificial dataset using a two-step process. In the first step, we begin by sampling across the set of artificial combined states $\boldsymbol{\Lambda}_a$ using the probability distribution $p_{\boldsymbol{\Lambda}_a}(\boldsymbol{\lambda}_a)$. Let $\boldsymbol{\Lambda}'_a \subset \boldsymbol{\Lambda}_a$ denote the set of sampled, artificial combined states according to the distribution $p_{\boldsymbol{\Lambda}_a}(\boldsymbol{\lambda}_a)$. Then we use the basic measurement model $h_{\mathbf{z}^*}$ to compute a set of unbiased, noise-free artificial sensor measurements $\mathbf{Z}^{*\prime}_a$ that correspond to each artificial combined state. That is, we compute the following set: $\{\ \mathbf{z}^*_a \in \mathbf{Z}^{*\prime}_a \mid \mathbf{z}^*_a = h_{\mathbf{z}^*_a}(\boldsymbol{\lambda}_a),\ \boldsymbol{\lambda}_a \in \boldsymbol{\Lambda}'_a\ \}$. When generating artificial combined states and unbiased, noise-free measurements, we must ensure that each $\boldsymbol{\lambda}_a \in \boldsymbol{\Lambda}_a$ is an observable state. Likewise, each $\mathbf{z}^*_a \in \mathbf{Z}^*_a$ must be within the field of view of our sensor. That is, the measurement must be observable.

The next step is to corrupt the unbiased, noise-free measurements with bias and noise. We first generate a set of biases $\boldsymbol{\Delta}'_a \subset \boldsymbol{\Delta}_a$ and noises $\boldsymbol{\Sigma}'_a \subset \boldsymbol{\Sigma}_a$ using the bias and noise models, respectively. Similar to the basic measurement model, we compute the following: $\{\ \boldsymbol{\delta}_a \in \boldsymbol{\Delta}'_a \mid \boldsymbol{\delta}_a = h_{\boldsymbol{\delta}_a}(\boldsymbol{\lambda}_a)\ \}$ and $\{\ \boldsymbol{\sigma}_a \in \boldsymbol{\Sigma}'_a \mid \boldsymbol{\sigma}_a = h_{\boldsymbol{\sigma}_a}(\boldsymbol{\lambda}_a)\ \}$. Afterward, we either add or multiple the biases and noises to the unbiased, noise-free measurements to create the set of artificial measurements that will be used to pre-train the SDSMM. For example, we can compute a set of artificial measurements $\mathbf{Z}_a$ as $\{\ \mathbf{z}_a \in \mathbf{Z}'_a \mid \mathbf{z}_a = \mathbf{z}^*_a + \boldsymbol{\delta}_a + \boldsymbol{\sigma}_a\ \}$, where $\boldsymbol{\sigma}_a \sim \mathcal{N}(0, \boldsymbol{\Sigma}_a)$.

3.3 Learning with a Large Artificial Dataset

3.3.1 Pre-training State-Dependent Sensor Measurement Models

After we generate the artificial data, we use it to pre-train an SDSMM. The goal of pre-training is to learn a related source task such that we can transfer the knowledge to learn our target task. In this dissertation, pre-training helps the SDSMM 1) learn to map from the input state space to the output measurement space and 2) potentially model how input states affect measurement bias and noise. Let $g(\boldsymbol{\lambda}; \mathbf{w}_0)$ denote an SDSMM, where $\mathbf{w}_0$ is a set of randomly initialized network parameters. Let $\boldsymbol{\mathcal{D}}_a = \{(\mathbf{z}_{a,i}, \boldsymbol{\lambda}_{a,i})\}_{i=1}^{I}$ denote an artificial dataset (Section 3.2) that contains I tuples, where each tuple contains an artificial measurement $\mathbf{z}_{a,i}$ and its corresponding combined state $\boldsymbol{\lambda}_{a,i}$.

Given $\boldsymbol{\mathcal{D}}_a$, we learn a set of network parameters $\mathbf{w}_a$ that minimizes the negative log-likelihood loss $\mathcal{E}(\boldsymbol{\mathcal{D}}_a, \mathbf{w})$ using (2.13). After pre-training, the SDSMM learns to predict parameters for state-dependent distributions that are similar to a physical model of the sensor.

3.3.2 Evaluating Pre-trained Models

Since we designed the measurement generation process for the artificial dataset, we know the true probability distribution for any given state. Therefore, we can use statistical measures to quantify the divergence between the true and predicted distributions at a given combined state. The first statistical measure is Kullback-Leibler (KL) divergence [24, 23], which was proposed by Soloman Kullback and Richard Leibler. Mathematically, KL divergence (also known as relative entropy) is calculated as

$$KL(P \mid\mid Q) = -\sum_{x \in X} P(x) \cdot \log\left(\frac{Q(x)}{P(x)}\right). \tag{3.4}$$

Here, $\mid\mid$ denotes divergence, x denotes an event, and $P(x)$ and $Q(x)$ represent the true and approximate probabilities of x occurring. A KL divergence score is zero when the probabilities $P(x)$ and $Q(x)$ are identical. However, if one probability is large and the other is relatively small, the KL divergence score is large. Furthermore, KL divergence is asymmetrical, meaning a score of $KL(P \mid\mid Q)$ is not equal to a score of $KL(Q \mid\mid P)$.

The second statistical measure is Jensen-Shannon (JS) Divergence, proposed by Johan Jensen and Claude Shannon. Unlike KL divergence, the JS divergence is symmetrical. Therefore, a score of $JS(P \mid\mid Q)$ is equal to a score of $JS(Q \mid\mid P)$. A JS divergence score has an interval of $[0, \infty]$. However, the score lies within the interval $[0, 1]$ if $\log_2(\cdot)$ is used. A normalized score of 0 indicates identical distributions, while 1 indicates maximally different distributions. To calculate a JS divergence score, we use the following equation:

$$JS(P \mid\mid Q) = \frac{1}{2} \cdot KL(Q \mid\mid G) + \frac{1}{2} \cdot KL(P \mid\mid G), \tag{3.5}$$

where $KL(\cdot)$ refers to (3.4) and $G = \frac{1}{2} \cdot (P + Q)$.

To evaluate a pre-trained SDSMM using KL or JS divergence, we use the true distributions used to generate the artificial dataset as $P(x)$ and the predicted distributions from the SDSMM as the approximate distribution $Q(x)$.

3.4 Learning with a Limited Real Dataset

3.4.1 Fine-tuning State-Dependent Sensor Measurement Models

We now use a smaller, yet real dataset to fine-tune the pre-trained SDSMM. In Section 3.3, we pre-trained the SDSMM so that the model learns a mapping from the input space to the measurement space using the artificial dataset. Here, the goal of fine-tuning is to leverage the learned knowledge from the source task and learn to predict parameters for distribution for the real sensor. Let $g(\boldsymbol{\lambda}; \mathbf{w}_A)$ denote a pre-trained SDSMM, where $\mathbf{w}_A$ is a set of parameters that were optimized for $\boldsymbol{\mathcal{D}}_A$ (our artificial dataset). Let $\boldsymbol{\mathcal{D}}_R = \{(\mathbf{z}_j, \boldsymbol{\lambda}_j)\}_{j=1}^{J}$ denote our small real dataset with J tuples, where each tuple contains a real sensor measurement $\mathbf{z}_j$ that was observed at the ground-truth combined state $\boldsymbol{\lambda}_j$. Here, $I \gg J$ where $I = |\boldsymbol{\mathcal{D}}_A|$, $J = |\boldsymbol{\mathcal{D}}_R|$, and $|\boldsymbol{\mathcal{D}}|$ is the cardinality of a set $\boldsymbol{\mathcal{D}}$. We also assume the elements of $\boldsymbol{\mathcal{D}}_R$ are independently and identically distributed.

Given $\boldsymbol{\mathcal{D}}_R$ and the network parameters $\mathbf{w}_A$, we learn a novel set of network parameters $\mathbf{w}_R$ that minimizes the negative log-likelihood loss $\mathcal{E}(\boldsymbol{\mathcal{D}}_R, \mathbf{w})$ using (2.13). Once fine-tuned, the novel network parameters $\mathbf{w}_R$ are optimized for $\boldsymbol{\mathcal{D}}_R$, and the fine-tuned SDSMM predicts distributions for our real sensor.

3.4.2 Evaluating Fine-tuned Models

Unlike our artificial dataset, we generally do not know the true measurement bias and noise for each combined state. Therefore, we must approximate the measurement bias and noise by gathering samples of measurements at each combined state. However, gathering samples of measurements at each state can be difficult because the real world is dynamic. For example, while a camera-based sensor gathers measurements for a particular state, the ambient lighting may change due to clouds and dynamic obstacles (such as people or vehicles) that may obstruct the view of landmarks. The problem is exacerbated when we can only collect small numbers of data for training an SDSMM. As a result, we cannot employ statistical measures such as KL or JS divergence as mentioned in Section 3.3.2.

Since we typically will not have the true distributions for a real sensor, one alternative mechanism for evaluating an SDSMM is examining model fit. For example, we can employ statistical measures such as negative log-likelihood to evaluate the predicted distributions of a model. For each combined state, the goal is to quantify how well the predicted distribution fits the corresponding measurement. We can also use mean-squared error (MSE) to compare the predicted distribution means and sensor measurements directly. An MSE informs us how close the regressed mean is to the measurement. For example, assuming our model predicts distribution means (which include measurement bias) and the real measurements are biased and noisy, an MSE informs us how close the predicted mean is to a biased measurement.

Although the negative log-likelihood and MSE help us evaluate model fit, such metrics are most helpful in comparing the relative performance amongst multiple models. Therefore, we can also use SDSMMs to solve localization and mapping problems in addition to the metrics mentioned above. Assuming we have the true combined states (including robot poses and landmark positions), we can first use SDSMMs in either problem. Then we can evaluate average (localization or mapping) error or the errors over time as the robot localizes or maps the environment. If an SDSMM has a lower localization or mapping error than other methods, we can assume that the SDSMM predicts more accurate distributions for the sensor.

3.5 Regularizing State-Dependent Measurement Models

One of the most common issues in machine learning is model overfit. For neural networks, overfit occurs when the training loss decreases, yet the validation loss stagnates or increases. In such a case, the network tends to learn the data and the noise in the data well, resulting in poor generalization with unseen data [2]. Overfitting can occur when a neural network has a high (memory) capacity or learns with a small number of data [2]. Therefore, we seek to lessen model overfit, especially when fine-tuning a network on a small training dataset.

There are multiple methods for reducing model overfit. One popular method is increasing the number of training data. However, in this context, we cannot gather additional real training data due to resource restrictions.

A second popular method for reducing model overfit is called regularization. According to Goodfellow *et al.*, regularization is "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." We generally add a weighted penalty term to the loss function as a form of regularization in machine learning. Common penalties include $L1$, $L2$ (or weight decay), and $L1 + L2$ regularization. For neural networks in particular, other regularization methods include early stopping [14], dropout, shakeout [18], whiteout [26], and adding noise to training data [3, 14, 44].

In this dissertation, we chose Rothfuss *et al.* [44] to regularize our models because their method is easy to implement and works specifically for conditional density estimation methods like MDNs. When fine-tuning an MDN, we follow Rothfuss *et al.* [44] in perturbing the input and target variables of a training set. Specifically, we add noise to the true combined states and the sensor measurements in the *real* training set $\boldsymbol{\mathcal{D}}_R = \{(\mathbf{z}_j, \boldsymbol{\lambda}_j)\}_{j=1}^{J}$. Rothfuss *et al.* [44] is used since the work focused on regularizing conditional density estimators (such as an MDN). In the context of our work, perturbing the states has three useful properties. First, perturbing states grow a training set artificially, which can improve model generalization [2]. Second, perturbing a combined state $\boldsymbol{\lambda}_j$ implies that its neighbors have similar amounts of measurement bias and noise. Third, in addition to regularization, we can also view perturbation as data augmentation.

We use two separate sources to add noise to a limited real dataset. The first source is the empirically measured or manufacturer-reported noise of sensors used to collect true combined states. For example, high-accuracy, ground truth sensors (such as a visual positioning system) generally capture ground truth states (of the robot and its environment). Although such sensors are highly accurate, their measurements still contain noise. The second source is the empirical or manufacturer-reported noise for a real sensor (that is, the sensor we seek to model).

During training, each variable in the training set receives a random amount of noise sampled from a zero-mean Gaussian distribution. We use the following equations to perturb the variables:

$$\widetilde{\boldsymbol{\lambda}}_j = \boldsymbol{\lambda}_j + \mathcal{N}(0, \boldsymbol{\Sigma}_t), \tag{3.6}$$

$$\widetilde{\mathbf{z}}_j = \mathbf{z}_j + \mathcal{N}(0, \boldsymbol{\Sigma}_s). \tag{3.7}$$

Here, $\widetilde{\boldsymbol{\lambda}}_j$ and $\widetilde{\mathbf{z}}_j$ are the j-th perturbed combined state and sensor measurement. $\boldsymbol{\Sigma}_t$ and $\boldsymbol{\Sigma}_s$ are the measurement covariance noises for the sensors used to collect true combined states and the sensor we are modelling, respectively. Therefore, $\widetilde{\boldsymbol{\lambda}}_j$ and $\widetilde{\mathbf{z}}_j$ become the new input and target of the network during training. However, we use the original (unperturbed) values for the validation set.

We emphasize that introducing error into artificial measurements and introducing error into real combined states have different purposes. Introducing error into artificial measurements produces measurements that are, to some extent, close to those of a real sensor. However, introducing noise into real combined states incorporates the noise of ground truth sensors and arbitrarily creates neighboring combined states, which may not exist in the real dataset.

3.6 Real-World Experiment

We performed three sets of learning experiments using the MR.CLAM dataset [25] described in Section 2.7.1. We referred to the experiments as Fine-tune with Limited Real Data (FLRD), Real Data Only (RDO), Limited Real Data Only (LRDO), and Artificial Data Only (ADO). All learning experiments used the MDN architecture described in Section 2.7.2.2. However, the hyper-parameters for each set of experiments were different. The hyper-parameters are described in the appropriate sections. We randomly selected courses 2, 3, 5, 6, 8, and 9 for training.

3.6.1 State-Dependent Sensor Measurement Models

3.6.1.1 Fine-tune with Limited Real Data (FLRD)

FLRD evaluated the method proposed in this chapter. To generate the artificial dataset, we used a traditional landmark-based distance-and-bearing measurement model. Our basic measurement model $h_{\mathbf{z}^*}(\boldsymbol{\lambda})$ was defined as the distance-and-bearing measurement model (2.22) described in Section 2.7.2.3. We randomly generated 2×10^5 unbiased, noise-free measurements using a uniform distribution. The uniform distribution was defined as $\mathcal{U}(0.15, 12)$ for range and $\mathcal{U}(-0.5236, 0.5236)$ for bearing. The units for range and bearing were meters and radians, respectively. Afterward, we used the inverse of the range-and-bearing model to

Table 3.1: Dataset sizes for all RDO experiments.

	Training Set Size	Testing Set Size
Robot 1	25,331	10,326
Robot 2	26,251	10,186
Robot 3	30,777	12,921
Robot 4	13,554	7,314
Robot 5	35,803	15,049

calculate the corresponding combined states:

$$\boldsymbol{\lambda}_i = \begin{bmatrix} \lambda_x \\ \lambda_y \end{bmatrix} = \begin{bmatrix} \rho_{A,i} \cdot \cos(\phi_{A,i}) \\ \rho_{A,i} \cdot \sin(\phi_{A,i}) \end{bmatrix}. \tag{3.8}$$

We defined the basic noise model $h_{\boldsymbol{\sigma}}(\boldsymbol{\lambda})$ as a zero-mean Gaussian with a fixed, diagonal covariance matrix $\mathbf{Q}_A$:

$$\mathbf{Q}_A = \begin{bmatrix} \boldsymbol{\sigma}^2_{A,\rho} & 0 \\ 0 & \boldsymbol{\sigma}^2_{A,\phi} \end{bmatrix}. \tag{3.9}$$

The range and bearing standard deviations were defined as $\boldsymbol{\sigma}_{A,\rho} = 10^{-4}$ and $\boldsymbol{\sigma}_{A,\phi} = 3.0 \times 10^{-8}$, respectively. Next, we sampled the noisy artificial measurements from the distribution $\mathbf{z}_{A,i} \sim \mathcal{N}(\mathbf{z}^*_{A,i}, \mathbf{Q}_A)$. Finally, the noisy artificial measurements $\mathbf{z}_{A,i}$ and the combined states $\boldsymbol{\lambda}_{A,i}$ were aggregated into the dataset $\boldsymbol{\mathcal{D}}_A = \{(\mathbf{z}_{A,i}, \boldsymbol{\lambda}_{A,i})\}_{i=1}^{I}$, which was used to pre-train an MDN.

For each real robot MR.CLAM dataset, we used the bootstrap method [12, 10] to create six datasets with differing sample sizes. The sample sizes were "all", 10^4, 5.0×10^3, 2.5×10^3, 5.0×10^2, and 10^2. "All" fine-tuned the pre-trained model with all available training data for a particular robot (Table 3.1). The "all" sample size provided an upper bound performance for the other sample sizes. The remaining sample sizes were used to simulate "limited" real data. For each robot, we randomly sampled each sample size 100 times to determine average performances [11]. In total, we created 3,000 real robot datasets $\boldsymbol{\mathcal{D}}_R = \{(\mathbf{z}_i, \boldsymbol{\lambda}_i)\}_{j=1}^{J}$, which resulted in fine-tuning 3,000 MDNs.

3.6.1.2 Real Data Only (RDO)

RDO represents networks that we were trained on *all* available real data. Table 3.1 quantifies the dataset sizes for each network. These networks use the method described in Chapter 2. Each RDO model was trained on data from a particular robot, creating five individual models. We compare RDO and FLRD to determine if FLRD can achieve similar performance to RDO but with fewer data.

3.6.1.3 Limited Real Data Only (LRDO)

LRDO served as an ablation study of our proposed method. We trained these networks using our simulated limited real data only. However, these networks were not pre-trained. We compare LRDO and FLRD to determine if pre-training is beneficial for learning accurate SDSMMs with limited real data.

3.6.1.4 Artificial Data Only (ADO)

ADO represents our pre-trained network. Therefore, this network was not fine-tuned with any real data. ADO represents a learned *a priori* range-and-bearing model since we trained this network with the artificial data generated using an *a priori* range-and-bearing model.

3.6.2 Real-World Results and Discussion

We compared RDO, LRDO, and ADO against FLRD to evaluate the proposed transfer learning method (FLRD). Our first comparison (Section 3.6.2.1) examined the performance amongst FLRD, RDO, and ADO as we decreased the number of available real data. Our

Table 3.2: Test set log-likelihood means for RDO and ADO models. A higher log-likelihood implies a better fit to the test data. For each robot, the best log-likelihood is emphasized for the RDO, FLRD, and ADO models.

	RDO	ADO
R1	**4.81**	−14.55
R2	5.44	−13.89
R3	5.33	−13.70
R4	**5.31**	−15.86
R5	5.27	−12.82

Table 3.3: Test set log-likelihood means and standard deviations for FLRD models. A higher log-likelihood implies a better fit to the test data. For each robot, the best mean is emphasized for the RDO, FLRD, and ADO models.

	Real Data Sample Sizes					
	All Real Data	10,000	5,000	2,500	500	100
R1	4.75 ± 0.05	4.75 ± 0.04	4.72 ± 0.06	4.68 ± 0.06	4.41 ± 0.05	3.85 ± 0.30
R2	$\mathbf{5.49} \pm 0.02$	5.42 ± 0.04	5.46 ± 0.03	5.31 ± 0.06	4.88 ± 0.20	3.94 ± 0.29
R3	$\mathbf{5.55} \pm 0.02$	5.54 ± 0.02	5.52 ± 0.03	5.46 ± 0.03	5.16 ± 0.06	4.00 ± 0.41
R4	5.25 ± 0.05	5.28 ± 0.03	5.01 ± 0.34	4.36 ± 0.42	4.67 ± 0.33	3.48 ± 1.00
R5	$\mathbf{5.57} \pm 0.03$	5.56 ± 0.03	5.51 ± 0.03	5.49 ± 0.04	5.19 ± 0.06	4.10 ± 0.36

second comparison (Section 3.6.2.2) examined the performance between FLRD and LRDO to determine if there is a benefit of transfer learning in this context.

We also used Rothfuss *et. al* [44] to regularize the FLRD models during fine-tuning. However, noise regularization did not improve our learning results for this experiment. Therefore, we did not include those results.

3.6.2.1 Comparing FLRD, RDO, and ADO

The results demonstrated that FLRD performed comparably to RDO. For sample sizes as small as 2,500 (∼19% of the training data), the results showed that RDO was either lower than FLRD or within three standard deviations of an FLRD mean log-likelihood (compare "RDO" in Table 3.2 with Table 3.3). For robots 2, 3, and 5, FLRD had higher

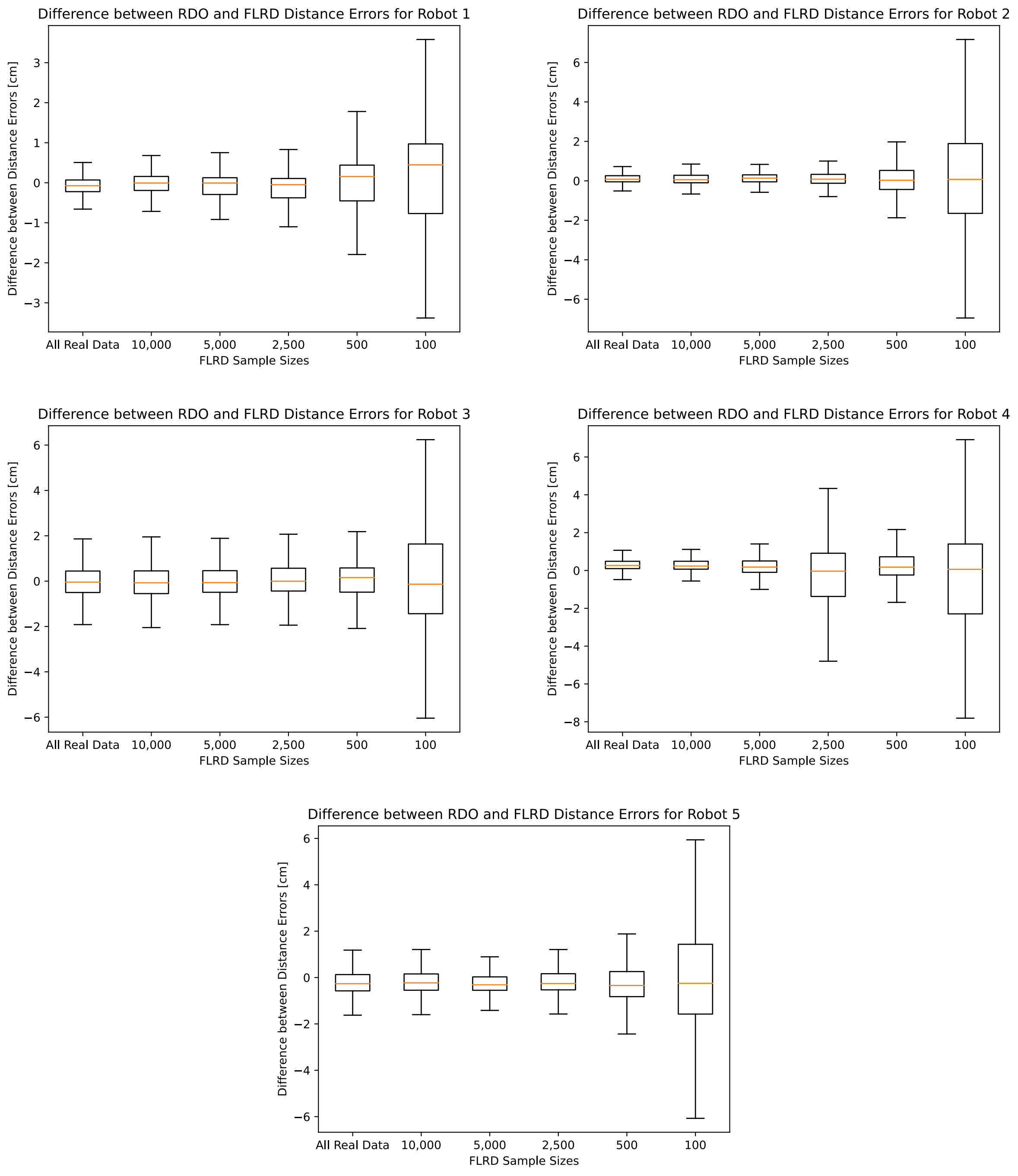


Figure 3.1: Difference between RDO and FLRD distance errors. This figure illustrates how close the predicted distance errors are between FLRD and RDO models. Intuitively, each graph shows that the difference between the FLRD and RDO errors increase as the FLRD sample sizes decrease. All distance values are expressed in centimeters.

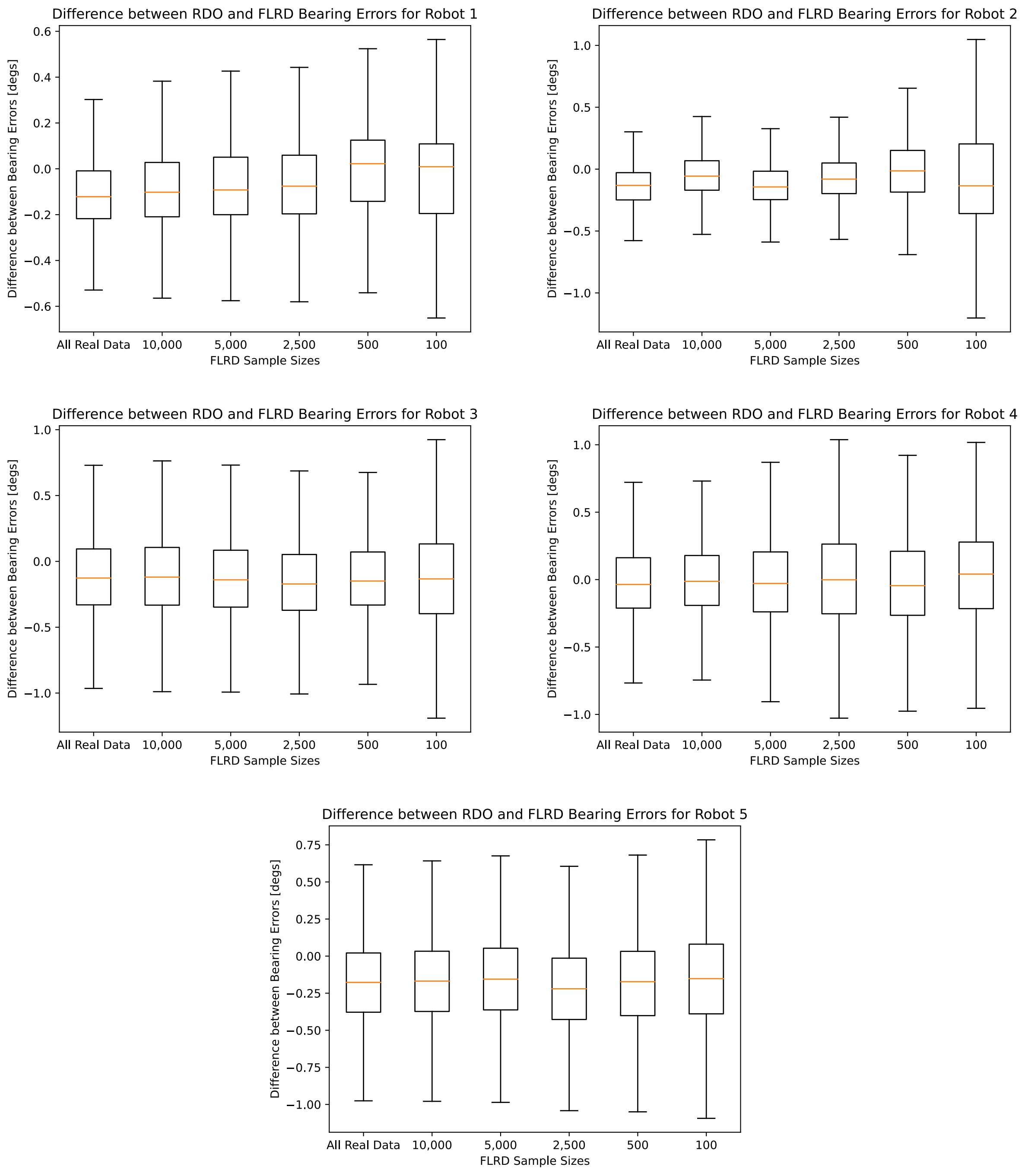


Figure 3.2: Difference between RDO and FLRD bearing errors. This figure illustrates how close the predicted bearing errors are between FLRD and RDO models. Generally, each graph shows that the differences between the FLRD and RDO errors vary less than distance errors as the FLRD sample sizes decrease. All bearing values are expressed in degrees.

Table 3.4: Distance error means and standard deviations (in centimeters) for RDO and ADO models. The best mean is emphasized for each robot.

	RDO	ADO
R1	$-0.5 \pm 2.9 *$	$0.2 \pm 10.5 *$
R2	$-\mathbf{0.4} \pm 2.5$	-0.5 ± 14.4
R3	$-\mathbf{0.3} \pm 2.5$	-4.8 ± 13.5
R4	$-\mathbf{0.5} \pm 2.8$	-1.5 ± 15.2
R5	$-\mathbf{0.5} \pm 3.1$	-4.1 ± 14.3

Table 3.5: Bearing error means and standard deviations (in degrees) for RDO and ADO models. The best mean is emphasized for each robot.

	RDO	ADO
R1	$-0.2 \pm 1.2 *$	$-0.0 \pm 1.4 *$
R2	$0.5 \pm 0.7 *$	$0.2 \pm 0.8 *$
R3	$0.2 \pm 0.8 *$	$-0.4 \pm 0.7 *$
R4	$-\mathbf{0.1} \pm 1.0$	-1.0 ± 1.6
R5	$\mathbf{0.2} \pm 0.7$	-0.2 ± 0.8

log-likelihoods than RDO. However, the higher log-likelihoods seem to be a by-product of fine-tuning (compare Tables 3.3 and 3.6). We also observed that the mean log-likelihoods for FLRD decreased as the sample size approached 100. For example, the lower log-likelihoods implied that FLRD predicted measurement probability distributions that had a worse fit than RDO at sample size 100.

Compared to ADO (Table 3.2), we observed that FLRD (Table 3.3) has significantly higher log-likelihoods. This observation holds at sample size 100. This result is also natural. Although the simulated and real data represent distance-and-bearing measurements with the same predictors, the measurement bias and noise are different. Therefore, the pre-trained model predicted measurement distributions that were different from the real data.

We also evaluated the errors between the RDO and FLRD predicted means. The goal of this evaluation was to observe how close the errors were from models. To perform this evaluation, we first computed the prediction errors for all RDO and FLRD models using the test set. Here, the prediction error is the difference between a sensor measurement and an MDN predicted mean. Next, for FLRD models with the same sample size, we calculated a mean prediction error across all 100 models. Finally, we computed the difference between the RDO errors and the mean FLRD errors.

The difference between the RDO and mean FLRD errors are depicted in figures 3.1 and 3.2. A box plot and an orange line represent the inter-quartile range and median of the error differences for each figure, respectively. The whiskers represent the lowest and highest error difference within the interval $[Q1 - 1.5 \times (Q3 - Q1), Q3 + 1.5 \times (Q3 - Q1)]$, respectively. For robots 1, 2, 3, and 5, Figure 3.1 shows that FLRD and RDO produce similar distance errors for sample sizes as few as 2,500. The figures illustrate that most error differences for these robots were as large as ± 3 centimeters. The distance errors for robot 4 were larger than the other robots for sample size 2,500, growing as high as about ± 5 centimeters. Generally speaking, the distance error differences for all robots grew at sample sizes 500 and 100. Compared to the larger sample sizes, FLRD models had fewer examples to learn the measurement bias, causing the models to predict less accurate measurement biases.

On the other hand, the differences between the RDO and FLRD bearing errors remained consistent across all sample sizes. Since bearing measurements had little bias, the sample sizes had little effect.

3.6.2.2 Comparing FLRD and LRDO

In most cases, the results showed that FLRD had similar or slightly better log-likelihoods than LRDO. Compared to LRDO, FLRD models produced higher log-likelihoods for 23 out of 25 cases; the two shaded cells in Table 3.6 show the cases where LRDO had higher log-likelihoods than FLRD. Therefore, tables 3.3 and 3.6 show that FLRD predicted distributions had a better fit than LRDO.

Like with the FLRD models, we computed the errors between the RDO and LRDO predicted measurements. Then we compared the LRDO errors (Figures 3.3, 3.4, and 3.5)

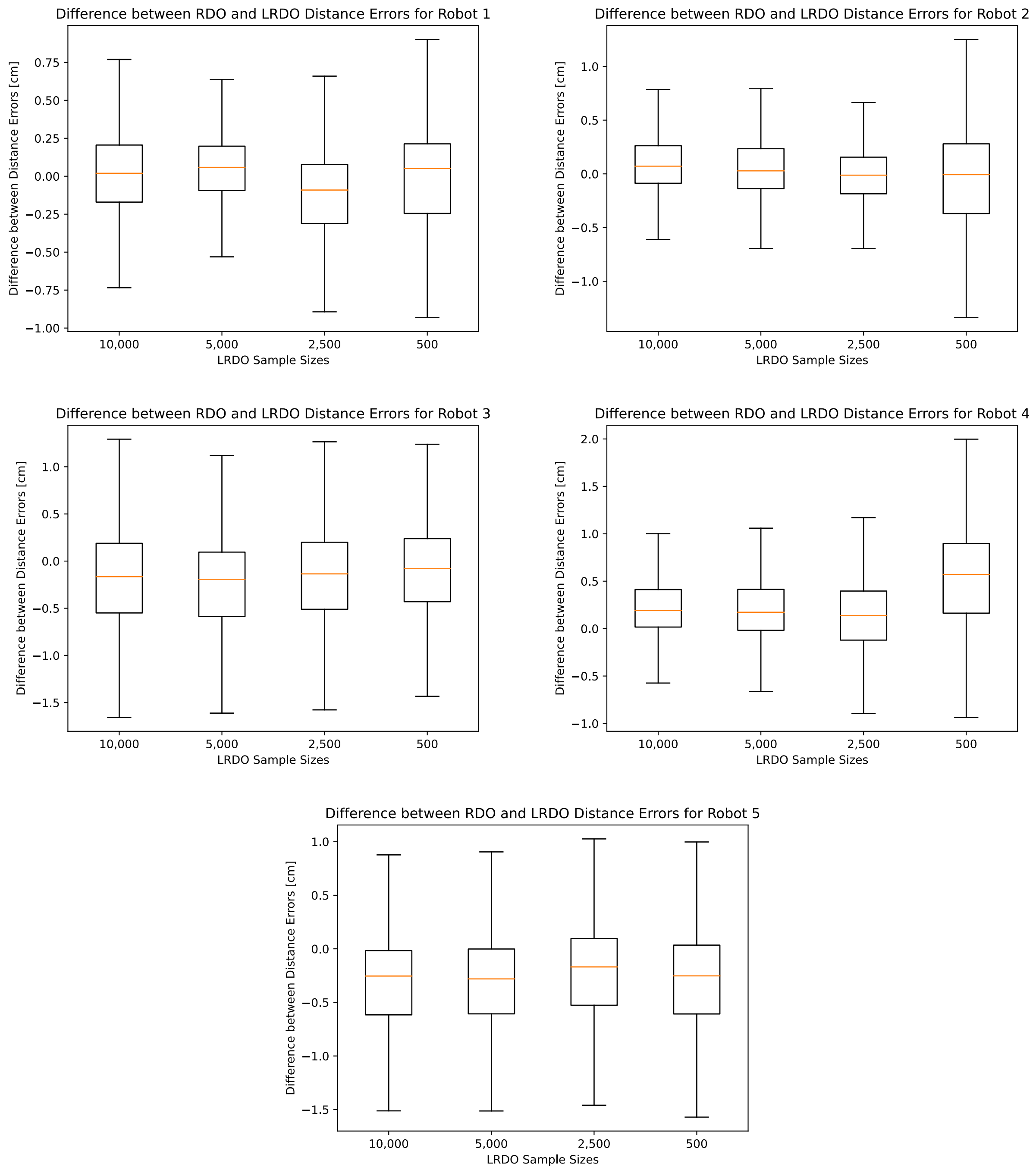


Figure 3.3: Difference between RDO and LRDO distance errors. This figure illustrates how close the predicted distance errors are between LRDO and RDO models. Due to the range of the LRDO errors at sample size 100, we only included sample sizes 10,000 down to 500 in this figure. All distance values are expressed in centimeters.

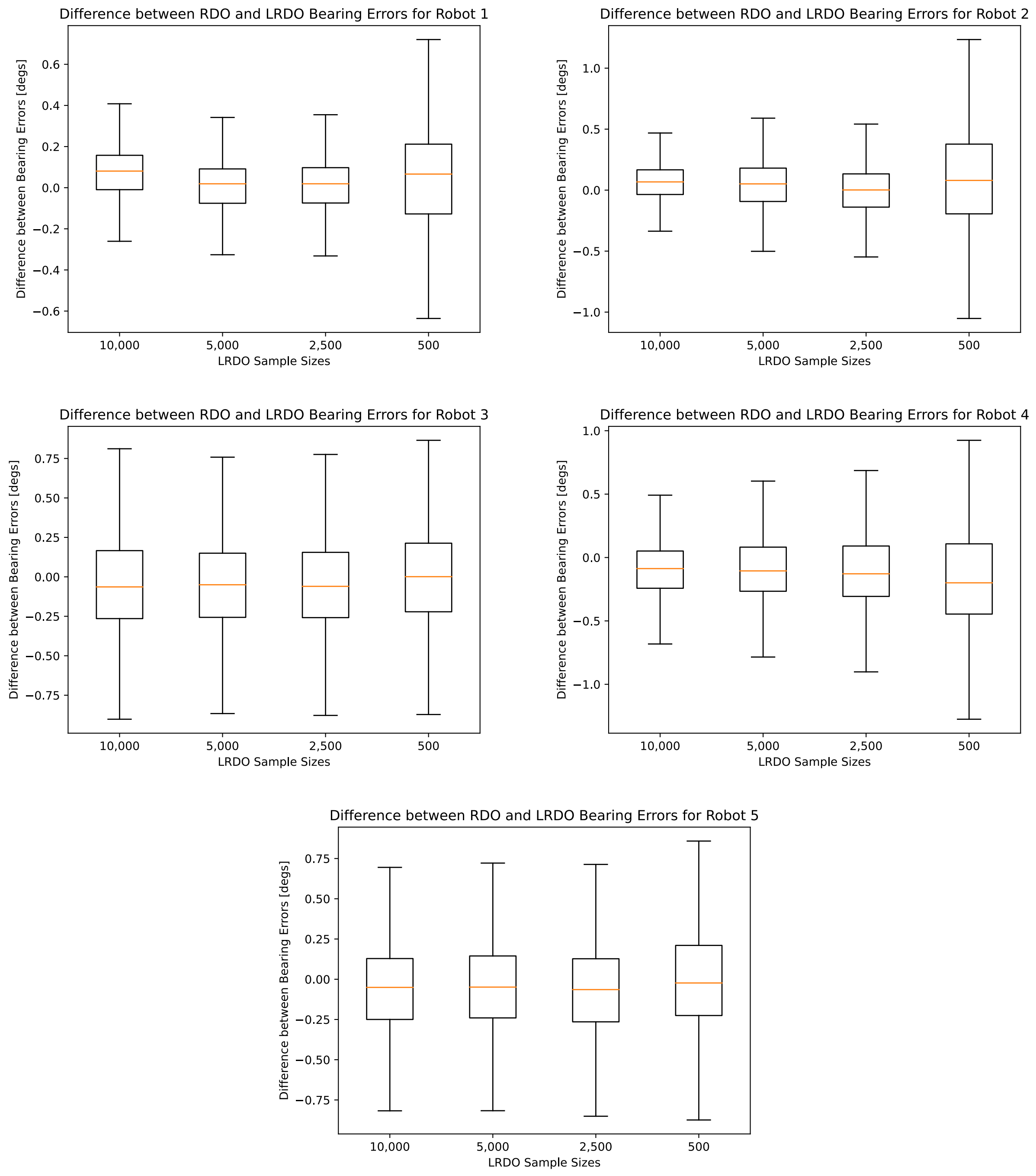


Figure 3.4: Difference between RDO and LRDO bearing errors. This figure illustrates how close the predicted distance errors are between LRDO and RDO models. Intuitively, each graph shows that the difference between the FLRD and RDO errors increase as the FLRD sample sizes decrease. All distance values are expressed in centimeters.

Table 3.6: Test set log-likelihood means and standard deviations for LRDO models. A higher log-likelihood implies a better fit to the test data. A shaded result signifies that LRDO had a higher mean log-likelihood than FLRD for the same robot and sample size.

	Sample Sizes				
	$10,000$	$5,000$	$2,500$	500	100
R1	4.54 ± 0.12	4.72 ± 0.04	4.17 ± 0.44	4.11 ± 0.27	0.81 ± 0.85
R2	5.38 ± 0.04	5.12 ± 0.07	4.82 ± 0.29	3.85 ± 0.75	0.43 ± 0.55
R3	5.32 ± 0.06	5.39 ± 0.05	5.34 ± 0.08	4.63 ± 0.28	0.41 ± 0.68
R4	5.17 ± 0.08	5.07 ± 0.13	5.01 ± 0.07	4.40 ± 0.32	0.41 ± 1.39
R5	5.49 ± 0.04	5.45 ± 0.06	5.39 ± 0.06	4.47 ± 0.47	0.53 ± 0.60

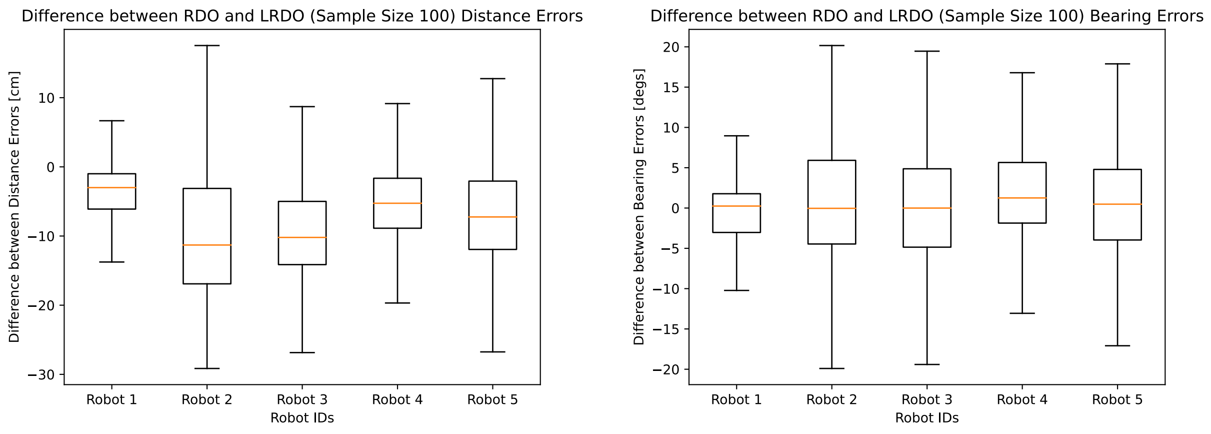


Figure 3.5: Difference between RDO and LRDO distance and bearing errors at sample size 100. At the smallest sample size (100), the LRDO models produced distributions that varied significantly from their RDO counterparts.

with the FLRD errors (Figures 3.1 and 3.2). For sample size 10,000 down to 500, we observed that the range of the LRDO errors were comparable to their FLRD counterparts.

At sample size 100, the results showed a noticeable difference in performance (compare the last columns of Tables 3.3 and 3.6). For sample size 100, LRDO models had significantly lower mean log-likelihoods and higher log-likelihood variances compared to their FLRD counterparts. The range of the LRDO errors were also significantly larger than their FLRD counterparts (see Figures 3.5, 3.1, and 3.2). These observations were not surprising: we observed a similar trend with the log-likelihoods in Tables 3.3 and 3.6.

Chapter 4: Using State-Dependent Sensor Measurement Models for Robot Localization

4.1 Introduction

The previous chapters introduced state-dependent sensor measurement models (SDSMMs), described methods for implementing and learning them, and evaluated them using simulated and real-world datasets. Our discussion now pivots to applying these models to solve a fundamental problem in robotics: localization. Localization is the process of inferring the state of a robot (such as the robot's pose) given the map of an environment [51]. In this chapter, we apply SDSMMs to localization because localization is a fundamental perceptual problem in robotics and most robotics tasks require information about the pose of the robot [51]. For example, localization allows a robot to express its state relative to other coordinate systems (such as a global map), determine the locations of objects within its environment, plan trajectories to reach goal positions, and map its surroundings.

The rest of this chapter is organized as follows[6]. In Section 4.2, we describe how to integrate our state estimator-agnostic SDSMM framework with three state estimators: the Extended Kalman Filter (Section 4.2.2), the Particle Filter (Section 4.2.3), and the Extended Kalman Particle Filter (Section 4.2.4). Next, Section 4.3 evaluates distance-and-bearing sensor models using the Extended Kalman Filter and the Extended Kalman Particle Filter. Afterward, Section 4.4 evaluates our SDSMM-augmented pole detection sensor model using the Particle Filter.

[6]A portion of this chapter was published in [58, 59, 60].

4.2 Integrating State-Dependent Sensor Measurement Models

4.2.1 State Estimator, Environment, and Robot Assumptions

The following subsections describe how to integrate an SDSMM with three state estimation algorithms: the Extended Kalman Filter (EKF), the Particle Filter (PF), and the Extended Kalman Particle Filter (EKPF). Although the literature proposes variations of these algorithms, we focus on their baseline implementations since such implementations are commonly known. We discuss the EKF and PF due to their popularity. The EKPF is discussed for two reasons. First, the EKPF offers a richer state distribution than a standalone EKF. Second, the EKPF also mitigates the sample degeneracy problem of a standalone PF [53]. For detailed descriptions and derivations of the EKPF, we invite readers to review [53]. We implement the SDSMM using an MDN. Due to the Gaussian assumption of the EKF and EKPF, we restrict the MDN to predict parameters for a Gaussian distribution. Let $\boldsymbol{g}(\boldsymbol{\lambda}; \mathbf{w})$ represent an MDN that outputs a mean and covariance matrix for a Gaussian distribution. The mean $\boldsymbol{\mu}$ represents either a measurement bias or an expected measurement (which inherently incorporates measurement bias). The covariance matrix $\boldsymbol{\Sigma}$ represents the covariance of the measurement noise. During localization, the MDN uses a propagated (*a priori*), sampled, or observed combined state since the robot does not know the ground-truth state.

We assume the environment is a planar surface, contains landmarks, and has at least one mobile robot. Such environments include a warehouse, an urban area, or a building. We forego discussing any data association methods for brevity because they were not a focus of this dissertation.

Each mobile robot is a land-based vehicle that navigates on the planar surface of our environment. A robot has a pre-built map of landmarks. To infer its pose, the robot measures the position of a landmark. Such measurements include 2D polar measurements (distance and bearing) or 2D Cartesian measurements.

4.2.2 The Extended Kalman Filter

An EKF is governed by two models: a state transition model and a sensor measurement model. The EKF state is represented by the parameters $\{\bar{\mathbf{x}}, \mathbf{P}\}$, which are the mean and covariance of a Gaussian distribution. During localization, the EKF first uses the state transition model to propagate the previous posterior state distribution $\{\bar{\mathbf{x}}_{t-1}, \mathbf{P}_{t-1}\}$ to produce $\{\bar{\mathbf{x}}_{t|t-1}, \mathbf{P}_{t|t-1}\}$, the propagated state distribution. The state transition model is defined as $\bar{\mathbf{x}}_{t|t-1} = f(\bar{\mathbf{x}}_{t-1}, \mathbf{u}_t) + \boldsymbol{\nu}$. $f(\cdot)$ is the nonlinear state transition function that propagates the system from previous state $\bar{\mathbf{x}}_{t-1}$ to a propagated state $\bar{\mathbf{x}}_{t|t-1}$. $\mathbf{u}_t$ is the current control command to the system. $\boldsymbol{\nu}$ is the state transition noise that is sampled from a zero-mean Gaussian distribution $\boldsymbol{\nu} \sim \mathcal{N}(0, \mathbf{R})$, where $\mathbf{R}$ is a fixed covariance of the state transition noise. The state propagation equations are defined as:

$$\bar{\mathbf{x}}_{t|t-1} = f(\bar{\mathbf{x}}_{t-1}, \mathbf{u}_t) \tag{4.1}$$

$$\mathbf{P}_{t|t-1} = \mathbf{F}_t \mathbf{P}_{t-1} \mathbf{F}_t + \mathbf{R}. \tag{4.2}$$

Here, $\bar{\mathbf{x}}_{t|t-1}$ and $\mathbf{P}_{t|t-1}$ are the propagated state and propagated state error covariance. $\mathbf{F}_t$ is the Jacobian matrix of the state transition function.

After the EKF propagates the state, the EKF then uses the measurement model to incorporate a sensor measurement $\mathbf{z}_t$ and approximate the next posterior distribution $\{\bar{\mathbf{x}}_t, \mathbf{P}_t\}$. Traditionally, the sensor measurement model is defined as $\mathbf{z}_t = h(\bar{\mathbf{x}}_{t|t-1}, \mathbf{m}) + \boldsymbol{\sigma}$. $h(\cdot)$ is the nonlinear measurement function that maps from a predicted state $\bar{\mathbf{x}}_{t|t-1}$ and an observed landmark $\mathbf{m}$ to the sensor measurement $\mathbf{z}_t$. $\boldsymbol{\sigma}$ is the measurement noise that is sampled from the distribution $\boldsymbol{\sigma} \sim \mathcal{N}(0, \mathbf{Q})$, where $\mathbf{Q}$ is a fixed covariance of the measurement noise. Therefore, the following equations incorporate the traditional sensor model:

$$\mathbf{K}_t = \mathbf{P}_{t|t-1} \mathbf{H}_t^\top \left(\mathbf{H}_t \mathbf{P}_{t|t-1} \mathbf{H}_t^\top + \mathbf{Q}_t \right)^{-1}, \tag{4.3}$$

$$\bar{\mathbf{x}}_t = \bar{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - h(\bar{\mathbf{x}}_{t|t-1}, \mathbf{m})), \tag{4.4}$$

$$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\,\mathbf{P}_{t|t-1}, \tag{4.5}$$

where $\mathbf{K}_t$ is the Kalman gain, $\mathbf{P}_{t|t-1}$ is the predicted state error covariance, and $\mathbf{I} \in \mathbb{R}^{3\times 3}$ is an identity matrix. $\mathbf{H}_t$ is a measurement Jacobian matrix of the measurement function.

To integrate an SDSMM with an EKF, we use the SDSMM to predict an expected measurement and a covariance of the measurement noise. Specifically, given a propagated combined state $\bar{\boldsymbol{\lambda}}_t$, the SDSMM predicts an expected measurement and a covariance of the measurement noise. Therefore, the following set of equations use an SDSMM to update the propagated state distribution:

$$\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{G}_t^\top \left(\mathbf{G}_t\mathbf{P}_{t|t-1}\mathbf{G}_t^\top + \bar{\boldsymbol{\Sigma}}_t\right)^{-1}, \tag{4.6}$$

$$\bar{\mathbf{x}}_t = \bar{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \bar{\boldsymbol{\mu}}_t), \tag{4.7}$$

$$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{G}_t)\,\mathbf{P}_{t|t-1}. \tag{4.8}$$

In (4.6) and (4.7), $\bar{\boldsymbol{\mu}}_t$ and $\bar{\boldsymbol{\Sigma}}_t$ represent the state-dependent expected measurement (which incorporates measurement bias) and the state-dependent covariance of the measurement noise, respectively. $\bar{\boldsymbol{\mu}}_t$ and $\bar{\boldsymbol{\Sigma}}_t$ are also the Gaussian parameter outputs of the MDN given a propagated combined state: $g(\bar{\boldsymbol{\lambda}}_{t|t-1}; \mathbf{w})$. The propagated combined state $\bar{\boldsymbol{\lambda}}_{t|t-1}$ is assumed to be derived from $\bar{\mathbf{x}}_{t|t-1}$. Theoretically, the measurement noise in (4.6) violates an assumption of the Kalman filter and its variants, where it is assumed that the noise is state-*independent*. However, we find that our formulation works in practice. Finally, in (4.6) and (4.8), $\mathbf{G}_t$ is a measurement Jacobian matrix and is defined as

$$\mathbf{G}_t = \frac{\partial \bar{\boldsymbol{\mu}}_t}{\partial \bar{\mathbf{x}}_{t|t-1}}. \tag{4.9}$$

This derivative is computed using the MDN output units for the mean $\bar{\boldsymbol{\mu}}_t$, the MDN itself, and the function that derives $\bar{\boldsymbol{\lambda}}_{t|t-1}$ from $\bar{\mathbf{x}}_{t|t-1}$. We can compute $\mathbf{G}_t$ using frameworks that provide auto-differentiation (such as [31, 35]).

4.2.3 The Particle Filter

The Particle Filter (PF) is governed by a state transition distribution and a sensor measurement distribution. The PF uses a next state distribution to propagate the prior particle states $\{\bar{\mathbf{x}}_{t-1}^{(i)}\}_{i=1}^{M}$ to produce $\{\bar{\mathbf{x}}_{t}^{(i)}\}_{i=1}^{M}$, the predicted particle states. Here, M is the particles in our system. The next state distribution is defined as $\bar{\mathbf{x}}_t^{(i)} \sim p(\bar{\mathbf{x}}_t^{(i)} \mid \bar{\mathbf{x}}_{t-1}^{(i)}, \mathbf{u}_t)$. $\mathbf{u}_t$ is the control command to the system. $\bar{\mathbf{x}}_t^{(i)}$ and $\bar{\mathbf{x}}_{t-1}^{(i)}$ are the propagated and previous states of the i-th particle, respectively. Generally, the PF assumes that one can sample from the next state transition distribution to propagate the particle states.

Afterward, the PF computes an importance factor w_t to incorporate a measurement into the system. The importance factor is computed as $w_t^{(i)} = p(\mathbf{z}_t \mid \bar{\mathbf{x}}_t^{(i)}, \mathbf{m})$, where $p(\cdot)$ is the measurement probability, $\mathbf{z}_t$ is the sensor measurement, $\bar{\mathbf{x}}_t^{(i)}$ is the i-th particle state, and $\mathbf{m}$ is an observed landmark. We propose two ways to integrate an SDSMM with a PF. The choose of either method depends on what the SDSMM mean represents. For simplicity, we restrict the measurement probability to a Gaussian distribution

$$w_t^{(i)} = p(\mathbf{z}_t \mid \mathbf{v}_t^{(i)}, \bar{\boldsymbol{\Sigma}}_t^{(i)}) = \frac{1}{(2\pi)^{k/2} \cdot \left|\bar{\boldsymbol{\Sigma}}_t^{(i)}\right|^{1/2}} \exp\left\{ \left(\mathbf{v}_t^{(i)}\right)^{\top} \cdot \left[\ \bar{\boldsymbol{\Sigma}}_t^{(i)}\ \right]^{-1} \cdot \left(\mathbf{v}_t^{(i)}\right) \right\}, \tag{4.10}$$

where k is the dimension of the sensor measurement, and $\mathbf{v}_t^{(i)}$ and $\bar{\boldsymbol{\Sigma}}_t^{(i)}$ are the innovation and the state-dependent measurement noise for the i-th particle, respectively. If we trained an SDSMM to output the expected measurement as the mean, we compute the innovation as:

$$\mathbf{v}_t^{(i)} = \mathbf{z}_t - \bar{\boldsymbol{\mu}}_t^{(i)}, \tag{4.11}$$

where $\bar{\boldsymbol{\mu}}_t^{(i)}$ represents the state-dependent expected measurement. Conversely, if we trained an SDSMM to output the measurement bias as the mean, we compute the innovation as:

$$\mathbf{v}_t^{(i)} = (\mathbf{z}_t + \bar{\boldsymbol{\mu}}_t^{(i)}) - h(\bar{\mathbf{x}}_t^{(i)}, \mathbf{m}), \tag{4.12}$$

where $\bar{\boldsymbol{\mu}}_t^{(i)}$ represents the state-dependent measurement bias, $h(\cdot)$ is a measurement function, and $\mathbf{m}$ is an observed landmark. In (4.10), (4.11), and (4.12), the mean $\bar{\boldsymbol{\mu}}_t^{(i)}$ and the noise $\bar{\boldsymbol{\Sigma}}_t^{(i)}$ are outputs of the SDSMM given the t-th propagated combined state $\bar{\boldsymbol{\lambda}}_t^{(i)}$. The t-th propagated combined state is partially derived from the propagated state of the t-th particle. Unlike with the EKF, the PF does not compute Jacobian matrices for its measurement model.

4.2.4 The Extended Kalman Particle Filter

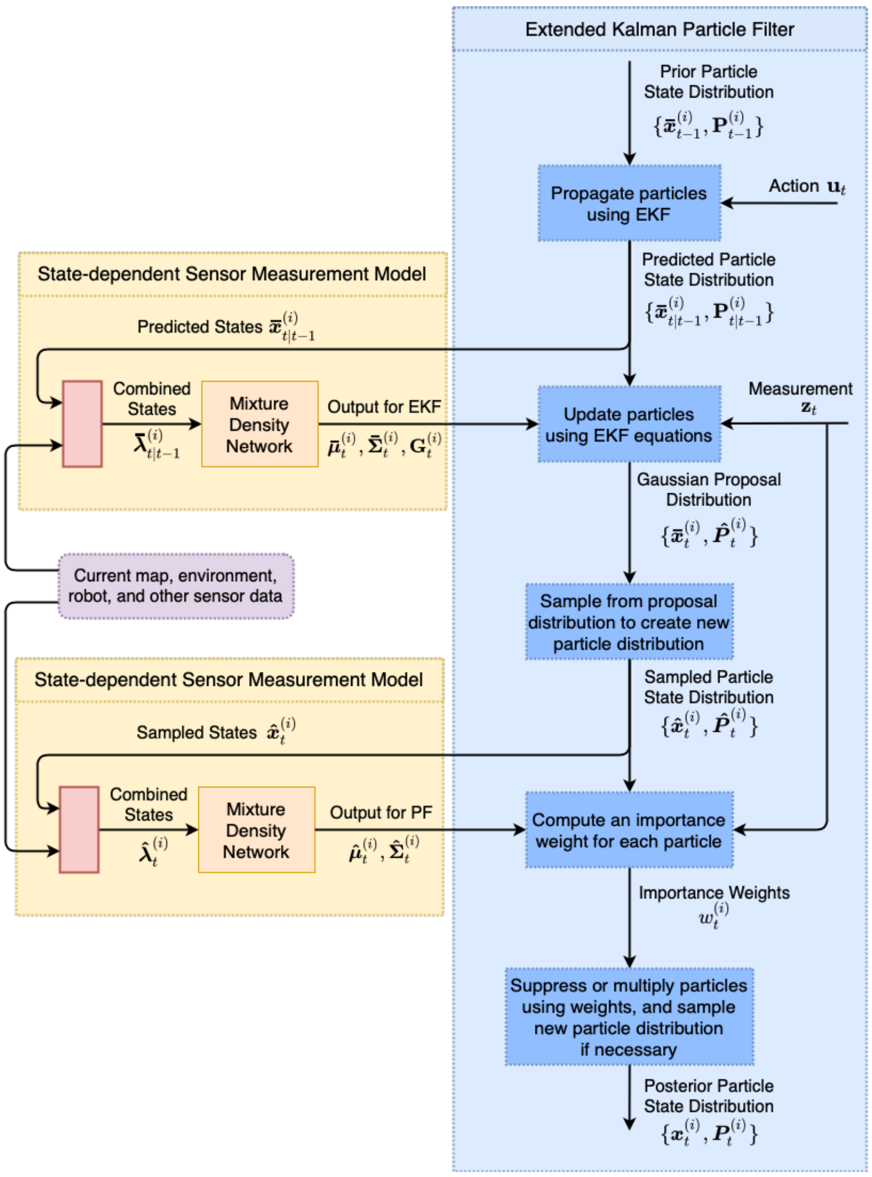
An Extended Kalman Particle Filter (EKPF) combines two popular state estimation algorithms: an EKF and a PF. Like the PF, the EKPF uses a collection of particles to represent the state distribution of the system. Each particle state within the EKPF is represented as the mean and covariance of a Gaussian distribution. We use the following notation to represent the system: $\{w^{(i)}, \bar{\mathbf{x}}^{(i)}, \mathbf{P}^{(i)}\}_{i=1}^{M}$, where M is the number of particles in the system. Figure 4.1 illustrates how an EKPF uses an SDSMM.

The EKPF begins by propagating the previous posterior state distribution $\{\bar{\mathbf{x}}_{t-1}^{(i)}, \mathbf{P}_{t-1}^{(i)}\}$, which produces the predicted state distribution $\{\bar{\mathbf{x}}_{t|t-1}^{(i)}, \mathbf{P}_{t|t-1}^{(i)}\}$. The EKPF uses the state transition equations from the EKF to propagate the particle state distribution:

$$\bar{\mathbf{x}}_{t|t-1}^{(i)} = f(\bar{\mathbf{x}}_{t-1}^{(i)}, \mathbf{u}_t), \tag{4.13}$$

$$\mathbf{P}_{t|t-1}^{(i)} = \mathbf{F}_t^{(i)} \mathbf{P}_{t-1}^{(i)} \mathbf{F}_t^{(i)\top} + \mathbf{R}. \tag{4.14}$$

Like the EKF, $\mathbf{u}_t$ is the control command to the system, $\mathbf{F}_t^{(i)}$ is the Jacobian matrix of the state transition model, and $\mathbf{R}$ is the state transition noise. When a sensor measurement $\mathbf{z}_t$



Extended Kalman Particle Filter
Prior Particle State Distribution $\{\bar{x}_{t-1}^{(i)}, \mathbf{P}_{t-1}^{(i)}\}$
Propagate particles using EKF
Action $\mathbf{u}_t$
Predicted Particle State Distribution $\{\bar{x}_{t|t-1}^{(i)}, \mathbf{P}_{t|t-1}^{(i)}\}$
State-dependent Sensor Measurement Model
Predicted States $\bar{x}_{t|t-1}^{(i)}$
Combined States $\bar{\lambda}_{t|t-1}^{(i)}$
Mixture Density Network
Output for EKF $\bar{\mu}_t^{(i)}, \bar{\Sigma}_t^{(i)}, \mathbf{G}_t^{(i)}$
Update particles using EKF equations
Measurement $\mathbf{z}_t$
Gaussian Proposal Distribution $\{\bar{x}_t^{(i)}, \hat{P}_t^{(i)}\}$
Current map, environment, robot, and other sensor data
Sample from proposal distribution to create new particle distribution
Sampled Particle State Distribution $\{\hat{x}_t^{(i)}, \hat{P}_t^{(i)}\}$
State-dependent Sensor Measurement Model
Sampled States $\hat{x}_t^{(i)}$
Combined States $\hat{\lambda}_t^{(i)}$
Mixture Density Network
Output for PF $\hat{\mu}_t^{(i)}, \hat{\Sigma}_t^{(i)}$
Compute an importance weight for each particle
Importance Weights $w_t^{(i)}$
Suppress or multiply particles using weights, and sample new particle distribution if necessary
Posterior Particle State Distribution $\{x_t^{(i)}, P_t^{(i)}\}$

Figure 4.1: Integrating the Extended Kalman Particle Filter and a State-Dependent Sensor Measurement Model. This figure illustrates how an Extended Kalman Particle Filter uses a state-dependent sensor measurement model (SDSMM). The SDSMM predicts a state-dependent measurement distribution $(\boldsymbol{\mu}_t^{(i)}, \boldsymbol{\Sigma}_t^{(i)})$ given the current estimated robot state, map, environment, robot, and sensor data. The model predicts the appropriate distribution parameters for the Extended Kalman Filter and the Particle Filter (importance weighting). The SDSMM can also be used to compute measurement Jacobian matrices $\mathbf{G}_t^{(i)}$ for the EKF. In the pipeline, we only use one SDSMM to compute the appropriate measurement distributions. However, we depicted the SDSMM twice to show the data flow into and out of the model.

is observed, the following EKF equations update each particle state:

$$\mathbf{S}_t^{(i)} = \mathbf{G}_t^{(i)} \mathbf{P}_{t|t-1}^{(i)} (\mathbf{G}_t^{(i)})^\top + \bar{\boldsymbol{\Sigma}}_t^{(i)} \tag{4.15}$$

$$\mathbf{K}_t^{(i)} = \mathbf{P}_{t|t-1}^{(i)} (\mathbf{G}_t^{(i)})^\top (\mathbf{S}_t^{(i)})^{-1}, \tag{4.16}$$

$$\bar{\mathbf{x}}_t^{(i)} = \bar{\mathbf{x}}_{t|t-1}^{(i)} + \mathbf{K}_t^{(i)} \left(\mathbf{z}_t - \bar{\boldsymbol{\mu}}_t^{(i)} \right), \tag{4.17}$$

$$\hat{\mathbf{P}}_t^{(i)} = \left(\mathbf{I} - \mathbf{K}_t^{(i)} \mathbf{G}_t^{(i)} \right) \mathbf{P}_{t|t-1}^{(i)}, \tag{4.18}$$

producing a Gaussian proposal distribution $\{\bar{\mathbf{x}}_t^{(i)}, \hat{\mathbf{P}}_t^{(i)}\}$. Here, a state error $\hat{\mathbf{P}}_t^{(i)}$ that is propagated along with its corresponding particle. $\bar{\boldsymbol{\mu}}_t^{(i)}$, $\bar{\boldsymbol{\Sigma}}_t^{(i)}$, and $\mathbf{G}_t^{(i)}$ are the state-dependent expected measurement, measurement noise, and measurement Jacobian for the i-th particle, respectively. Like in Section 4.2.2, the measurement noise in (4.15) violates a theoretical assumption, where it is assumed that the noise is state-*independent*. However, we find that this formulation works in practice too.

Once the Gaussian proposal distribution is computed, the EKPF samples from the proposal distribution

$$\hat{\mathbf{x}}_t^{(i)} \sim q(\mathbf{x}_t^{(i)} | \mathbf{x}_{0:t-1}^{(i)}, \mathbf{z}_{1:t}) = \mathcal{N}(\bar{\mathbf{x}}_t^{(i)}, \hat{\mathbf{P}}_t^{(i)}), \tag{4.19}$$

to create a new particle set $\{\hat{\mathbf{x}}_t^{(i)}, \hat{\mathbf{P}}_t^{(i)}\}$. Afterward, we compute an importance weight $w_t^{(i)}$ for each particle:

$$w_t^{(i)} \propto \frac{p(\mathbf{z}_t \mid \hat{\boldsymbol{\lambda}}_t^{(i)}) \cdot p(\hat{\mathbf{x}}_t^{(i)} \mid \mathbf{x}_{t-1}^{(i)})}{q(\hat{\mathbf{x}}_t^{(i)} \mid \mathbf{x}_{0:t-1}^{(i)}, \mathbf{z}_{1:t})}. \tag{4.20}$$

Here, $p(\hat{\mathbf{x}}_t^{(i)} \mid \mathbf{x}_{t-1}^{(i)})$ is the transition prior, $\hat{\mathbf{x}}_t^{(i)}$ is the sampled particle state from (4.19), and $\mathbf{x}_{t-1}^{(i)}$ is the posterior state from the previous time step $(t-1)$. We do not define the transition prior since it is independent of a sensor measurement. The proposal distribution, $q(\hat{\mathbf{x}}_t^{(i)} \mid \mathbf{x}_{0:t-1}^{(i)}, \mathbf{z}_{1:t})$, equates to $\mathcal{N}(\hat{\mathbf{x}}_t^{(i)} \mid \bar{\mathbf{x}}_t^{(i)}, \hat{\mathbf{P}}_t^{(i)})$ due to the EKF Gaussian assumption (see [53] for details). Due to the use of the EKF, we assume the measurement probability can be represented with a Gaussian distribution. Therefore, The state-dependent measurement probability $p(\mathbf{z}_t|\hat{\boldsymbol{\lambda}}_t^{(i)})$ is computed as

$$p(\mathbf{z}_t \mid \hat{\boldsymbol{\mu}}_t^{(i)}, \hat{\boldsymbol{\Sigma}}_t^{(i)}) = \frac{1}{(2\pi)^{k/2} \cdot \left|\hat{\boldsymbol{\Sigma}}_t^{(i)}\right|^{1/2}} \exp\left\{\left(\mathbf{z}_t - \hat{\boldsymbol{\mu}}_t^{(i)}\right)^\top \cdot \left[\ \hat{\boldsymbol{\Sigma}}_t^{(i)}\ \right]^{-1} \cdot \left(\mathbf{z}_t - \hat{\boldsymbol{\mu}}_t^{(i)}\right)\right\}, \quad (4.21)$$

where $\hat{\boldsymbol{\mu}}_t^{(i)}$ and $\hat{\boldsymbol{\Sigma}}_t^{(i)}$ are the Gaussian parameter outputs of the MDN, given the state $\hat{\boldsymbol{\lambda}}_t^{(i)}$ as input. Finally, $\hat{\boldsymbol{\lambda}}_t^{(i)}$ is derived from the sampled state $\hat{\mathbf{x}}_t^{(i)}$ in (4.19).

4.3 Real-World Experiments I: A Multi-Robot Localization Study

4.3.1 Experimental Setup

We ran numerous localization experiments using five camera-based range-and-bearing sensors from the MR.CLAM dataset. We learned four types of measurement models for each camera sensor. The measurement models were baseline, RDO, FLRD, and LRDO. The baseline model was an *a priori* measurement model. This model computed an unbiased experiment measurement and used a fixed covariance of the measurement noise. RDO, FLRD, and LRDO represented SDSMMs that were trained techniques mentioned in this dissertation. RDO learned from all real data, FLRD used transfer learning to learn from an abundance of artificial and limited real data, and LRDO learned from limited real data only. We encourage our readers to review Section 3.6 to determine the learning details for each model.

We used the Extended Kalman Filter and the Extended Kalman Particle Filter as our state estimators. Our EKPF used 500 particles to approximate the state. We determined the number of particles through cross-validation using the baseline measurement model. All localization experiments used courses 4 and 7. For the EKF, we defined the state transition model $\bar{\mathbf{x}}_{t|t-1} = f(\bar{\mathbf{x}}_{t-1}, \mathbf{u}_t)$ as follows:

$$\bar{\mathbf{x}}_t = \begin{bmatrix} \bar{x}_{t|t-1} \\ \bar{y}_{t|t-1} \\ \bar{\theta}_{t|t-1} \end{bmatrix} = \begin{bmatrix} \bar{x}_{t-1} + v_t \cos(\bar{\theta}_{t-1})\Delta t \\ \bar{y}_{t-1} + v_t \sin(\bar{\theta}_{t-1})\Delta t \\ \bar{\theta}_{t-1} + \omega_t \Delta t \end{bmatrix}, \tag{4.22}$$

where $\bar{\mathbf{x}}_{t-1} = [\bar{x}_{t-1}, \bar{y}_{t-1}, \bar{\theta}_{t-1}]^\top$ is the previous state, $\mathbf{u}_t = [v_t, \omega_t]^\top$ is the linear and angular odometry commands to the EKF, and Δt is the incremental time between two odometry commands $\mathbf{u}_{t-1}$ and $\mathbf{u}_t$. We defined the Jacobian of the state transition model $\mathbf{F}_t$ as

$$\mathbf{F}_t = \begin{bmatrix} 1 & 0 & -v_t \sin(\theta_{t-1})\Delta t \\ 0 & 1 & v_t \cos(\theta_{t-1})\Delta t \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.23}$$

We used similar formulations for the EKPF state transition model. However, in the case of the EKPF, we computed an individual state variable for each particle.

Both state estimators also employed the state-dependent equations and variables described in Sections 4.2.2 and 4.2.4. Since we do not have access to the ground-truth states during localization, the baseline and state-dependent measurement models used a predicted, propagated, or updated state as input. We used (2.20) to compute a combined state.

4.3.2 Results and Discussion

4.3.2.1 Localization Metrics

We used the root-mean-squared error (RMSE) and mean absolute error (MAE) to evaluate the localization experiments. The RMSE and MAE were computed between the ground-truth and estimated states. Each state $\bar{\mathbf{x}}_t = [x_t, y_t, \theta_t]^\top$ was represented as a 2D Cartesian position $[x_t, y_t]^\top$ and a 1D heading θ_t. The ground-truth states came from a roof-mounted Vicon system, while the estimated states came from the state estimators. For the EKF, we used the output estimated states directly. To derive the estimated states for the EKPF, we computed the mean of the particles at each time step: $\bar{\mathbf{x}}_t = \sum_{i=1}^{M} \left(w_t^{(i)} \cdot \bar{\mathbf{x}}_t^{(i)} \right)$. At time t, let $\bar{\mathbf{x}}_t^G$ denote a ground-truth state and $\bar{\mathbf{x}}_t$ denote an estimated state. The position RMSE E_{PE} was calculated as

$$E_{PE} = \sqrt{\frac{1}{|T|} \left(\sum_{t=1}^{|T|} \left\| \bar{\mathbf{x}}_{t,[x,y]}^G - \bar{\mathbf{x}}_{t,[x,y]} \right\|_2^2 \right)}, \tag{4.24}$$

where t is a time step, $|T|$ is the number of time steps for a particular robot and course, and $\bar{\mathbf{x}}_{t,[x,y]}^G$ and $\bar{\mathbf{x}}_{t,[x,y]}$ are the 2D positions for the ground-truth and estimated states. We computed the heading MAE E_{HE} was computed as

$$E_{HE} = \sqrt{\frac{1}{|T|} \left(\sum_{t=1}^{|T|} \left| \theta_t^G - \theta_t \right| \right)}, \tag{4.25}$$

where θ_t^G and θ_t are the ground-truth and estimated robot headings.

Since FLRD and LRDO experiments used multiple models per robot, we computed a mean and variance of the errors to determine the performance of each model. To describe how we computed the error means and standard deviations, let us briefly consider the estimated states from one robot, one course, and an FLRD measurement model. Let $\bar{\mathbf{x}}_{t,s,i}$ represent the estimated robot state at time step t. The estimated robot state was produced by a state estimator that used an FLRD model. We fine-tuned the model with the i-th bootstrapped dataset, which had a sample size of s. Here, $1 \leq i \leq 100$ and $s = \{all, 10^4, 5 \times 10^3, 2.5 \times 10^3, 5 \times 10^2, 1 \times 10^2\}$. To compute the error statistics, we first computed the position RMSE $E_{PE,s,i}$ and heading MAE $E_{HE,s,i}$ for the s-th sample size and i-th bootstrapped dataset using (4.24) and (4.25), respectively. Then we computed the position and heading error means and standard deviations for a given sample size s. The position

Table 4.1: Course 4 EKF position errors (in centimeters) for the baseline, ADO, and RDO models. The best EKF position error is emphasized amongst the baseline, ADO, RDO, and FLRD models for each robot. An asterisk "$*$" denotes the best error for each robot in the following table.

	Baseline	ADO	RDO
R1	6.8	4.8	**4.3**$*$
R2	8.4	9.9	**4.6**$*$
R3	10.4	6.1	5.9$*$
R4	11.6	9.6	9.1$*$
R5	9.2	8.9	4.6$*$

error mean and standard deviation were computed as:

$$E^{\mu}_{PE,s} = \frac{1}{100}\sum_{i=1}^{100} E_{PE,s,i}, \tag{4.26}$$

$$E^{\sigma}_{PE,s} = \sqrt{\frac{1}{100}\sum_{i=1}^{100}\left(E_{PE,s,i} - E^{\mu}_{PE,s}\right)^2}. \tag{4.27}$$

For a given sample size s, the heading error mean $E^{\mu}_{HE,s}$ and standard deviation $E^{\sigma}_{HE,s}$ were computed using the same equations.

For each combination of a robot, course, and SDSMM trained using bootstrap (FLRD or LRDO), and dataset sample size, we computed a position error mean and standard deviation and a heading error mean and standard deviation.

4.3.2.2 Comparing the Baseline, ADO, and RDO

We compared the baseline, ADO, and RDO to measure the difference in performance amongst the three models. Through our results, we observed three characteristics when we compared the position errors for the three models. Our first observation showed that RDO is superior to the baseline and ADO models, no matter what state estimator is used. Tables 4.1 and 4.2 show that RDO reduced the EKF position error by 21.3% to 50% and 10.4%

Table 4.2: Course 7 EKF position errors (in centimeters) for the baseline, ADO, and RDO models. The best EKF position error is emphasized amongst the baseline, ADO, RDO, and FLRD models for each robot. An asterisk “$*$” denotes the best error for each robot in the following table.

	Baseline	ADO	RDO
R1	12.9	40.8	**8.7**$*$
R2	12.0	13.7	**6.4**$*$
R3	16.2	21.6	**11.7**$*$
R4	15.5	30.4	**12.2**$*$
R5	12.2	18.6	**7.9**$*$

Table 4.3: Course 4 EKPF position errors (in centimeters) for the baseline, ADO, and RDO models. The best EKPF position error is emphasized amongst the baseline, ADO, RDO, and FLRD models for each robot. An asterisk “$*$” denotes the best error for each robot in the following table.

	Baseline	ADO	RDO
R1	5.9	5.3	**3.6**$*$
R2	8.4	11.8	**4.2**$*$
R3	9.1	6.4	**4.0**$*$
R4	13.6	10.5	**5.7**$*$
R5	7.7	9.4	**3.5**$*$

Table 4.4: Course 7 EKPF position errors (in centimeters) for the baseline, ADO, and RDO models. The best EKPF position error is emphasized amongst the baseline, ADO, RDO, and FLRD models for each robot. An asterisk “$*$” denotes the best error for each robot in the following table.

	Baseline	ADO	RDO
R1	13.5	27.2	**7.2**$*$
R2	10.8	15.7	**5.4**$*$
R3	16.5	22.2	**10.8**$*$
R4	19.0	33.5	**9.4**$*$
R5	12.2	21.1	**6.7**$*$

to 78.7% when compared to their baseline and ADO counterparts, respectively. Likewise, tables 4.3 and 4.4 show that RDO reduced the EKPF position error by 34.5% to 58% and 32.1% to 73.2% when compared to their baseline and ADO counterparts, respectively.

Our second observation highlighted the importance of accurate measurement noise by comparing the baseline and ADO. For course 4 and both state estimators (tables 4.1 and 4.3), ADO outperformed the baseline for three out of five robots: robots 1, 3, and 4. We note that ADO for robot 5 also outperformed its baseline counterpart for the EKF on course 4. However, for the same robot and course, ADO had a higher position error than the baseline for the EKPF.

When we examined the ADO position errors for course 7, we observed that the ADO errors were significantly higher than their baseline counterparts. Furthermore, the percent increase between the baseline and ADO models was not affected by the state estimator. The position errors between the baseline and ADO increased by 14.2% to 216.3% for the EKF and 34.5% to 101.5% for the EKPF. Since the baseline and ADO models output similar expected measurements, the performance difference between the baseline and ADO is likely due to the measurement noise. Specifically, the ADO predicted measurement noise is smaller than the fixed measurement noise used by the baseline. The smaller measurement noise made both state estimators weigh the inaccurate ADO measurements more, which increased the localization error.

Compared to the EKF, the third observation showed that the RDO position errors consistently improved when we used the EKPF, which is a more expressive state estimator. However, the baseline and ADO position errors were not consistently better when the EKPF was used. The EKPF baseline had better position errors than the EKF baseline for three robots on course 4 and one robot on course 7. For ADO, the results were relatively worse than the baseline. An EKPF ADO outperformed its EKF counterpart in only one case.

Table 4.5: Course 4 EKF position errors (in centimeters) for FLRD models. The best error is emphasized amongst baseline, ADO, RDO, and FLRD models. An asterisk “$*$” denotes the best error(s) for each robot in the following table.

	Real Data Sample Sizes					
	All Real Data	10,000	5,000	2,500	500	100
R1	4.5 ± 0.2	4.3 ± 0.2 $*$	4.3 ± 0.2 $*$	4.3 ± 0.2 $*$	4.5 ± 0.2	4.9 ± 0.5
R2	6.3 ± 5.4	5.5 ± 0.4	5.7 ± 0.5	5.2 ± 0.2 $*$	5.5 ± 0.2	5.7 ± 0.2
R3	**5.5** ± 0.1 $*$	**5.5** ± 0.1 $*$	5.6 ± 0.1	5.8 ± 0.1	6.5 ± 0.1	6.7 ± 0.2
R4	8.4 ± 0.2	8.5 ± 0.2	8.5 ± 0.2	8.5 ± 0.2	8.7 ± 0.2	**8.3** ± 0.3 $*$
R5	**4.6** ± 0.1 $*$	4.6 ± 0.2	4.8 ± 0.1	4.8 ± 0.1	5.1 ± 0.2	4.7 ± 0.2

Table 4.6: Course 7 EKF position errors (in centimeters) for FLRD models. The best error is emphasized amongst baseline, ADO, RDO, and FLRD models. An asterisk “$*$” denotes the best error(s) for each robot in the following table.

	Real Data Sample Sizes					
	All Real Data	10,000	5,000	2,500	500	100
R1	9.1 ± 0.2	9.1 ± 0.2	9.2 ± 0.2	9.4 ± 0.3	10.3 ± 0.4	10.3 ± 0.6
R2	6.7 ± 0.5	6.8 ± 0.2	6.8 ± 0.2	6.9 ± 0.1	7.7 ± 0.2	8.7 ± 0.3
R3	21.6 ± 8.7	16.3 ± 3.5	15.2 ± 2.7	14.0 ± 2.0	**11.5** ± 0.2	13.7 ± 1.3
R4	11.0 ± 0.7	12.9 ± 19.0	10.7 ± 0.5	**10.3** ± 0.4	11.0 ± 2.4	10.3 ± 0.5
R5	8.5 ± 1.1	7.8 ± 0.5	**7.7** ± 0.1	7.7 ± 0.2	7.8 ± 0.2	8.9 ± 0.6

Table 4.7: Course 4 EKPF position errors (in centimeters) for FLRD models. The best error is emphasized amongst baseline, ADO, RDO, and FLRD models. An asterisk “$*$” denotes the best error(s) for each robot in the following table.

	Real Data Sample Sizes					
	All Real Data	10,000	5,000	2,500	500	100
R1	3.7 ± 0.3	3.5 ± 0.3	3.4 ± 0.3	**3.3** ± 0.2	**3.3** ± 0.2	3.9 ± 0.8
R2	3.7 ± 0.7	3.5 ± 0.4	3.7 ± 0.5	**3.5** ± 0.3	3.7 ± 0.2	4.1 ± 0.3
R3	3.9 ± 0.1	3.9 ± 0.1	**3.8** ± 0.1	3.9 ± 0.1	4.2 ± 0.1	4.5 ± 0.3
R4	5.5 ± 0.2	**5.4** ± 0.2	5.8 ± 0.3	6.1 ± 0.3	5.9 ± 0.3	6.3 ± 0.4
R5	3.7 ± 0.8	3.5 ± 0.5	3.5 ± 0.4	**3.4** ± 0.4	3.7 ± 0.2	3.7 ± 0.3

Table 4.8: Course 7 EKPF position errors (in centimeters) for FLRD models. The best error is emphasized amongst baseline, ADO, RDO, and FLRD models. An asterisk "$*$" denotes the best error(s) for each robot in the following table.

	Real Data Sample Sizes					
	All Real Data	10,000	5,000	2,500	500	100
R1	7.6 ± 0.7	7.6 ± 0.7	7.6 ± 0.7	7.7 ± 0.7	8.0 ± 0.6	9.3 ± 1.3
R2	5.7 ± 0.3	5.8 ± 1.0	5.7 ± 0.2	5.7 ± 0.2	6.4 ± 0.4	7.3 ± 0.5
R3	19.9 ± 4.9	13.9 ± 3.9	12.6 ± 2.9	11.8 ± 1.9	**9.9** ± 0.3	11.9 ± 1.5
R4	**9.3** ± 1.1	9.5 ± 2.3	9.4 ± 0.5	9.7 ± 0.6	11.6 ± 7.4	10.1 ± 1.5
R5	6.8 ± 0.5	6.8 ± 0.4	7.1 ± 0.5	7.0 ± 0.5	7.1 ± 0.6	9.3 ± 0.8

4.3.2.3 Comparing RDO and FLRD

We compared RDO and FLRD to measure the similarity in performance between RDO and FLRD. With these results, we observed three findings. First, we observed that FLRD achieved position errors that were comparable to RDO in most cases. For example, the results for course 4 showed that FLRD generally had position errors that were either lower than or within 2 to 8 millimeters of their RDO counterpart (Tables 4.5 and 4.7). The worst-case difference between FLRD and RDO was 1.7 centimeters (sample size "all" for robot 2 in table 4.5). Here, one experiment diverged, causing the RMSE mean and standard deviation to increase.

We also observed that there was a greater difference between RDO and FLRD for course 7. Most RDO and FLRD results were comparable for robots 1, 2, 4, and 5. For sample sizes as small as 2,500, the difference between RDO and FLRD ranged from 3 and 7 millimeters. However, as the sample size approached 100, we observed larger differences between RDO and FLRD RMSEs. At a sample size of 100, the range in differences increased, spanning from 1 to 2.6 centimeters for both state estimators.

All FLRD experiments did not perform as well as RDO on course 7. Specifically, robot 3 performed noticeably worse on this course for sample sizes "all" down to 2,500. We do not believe that the higher errors are due to issues with training. Robot 3 had the second-highest

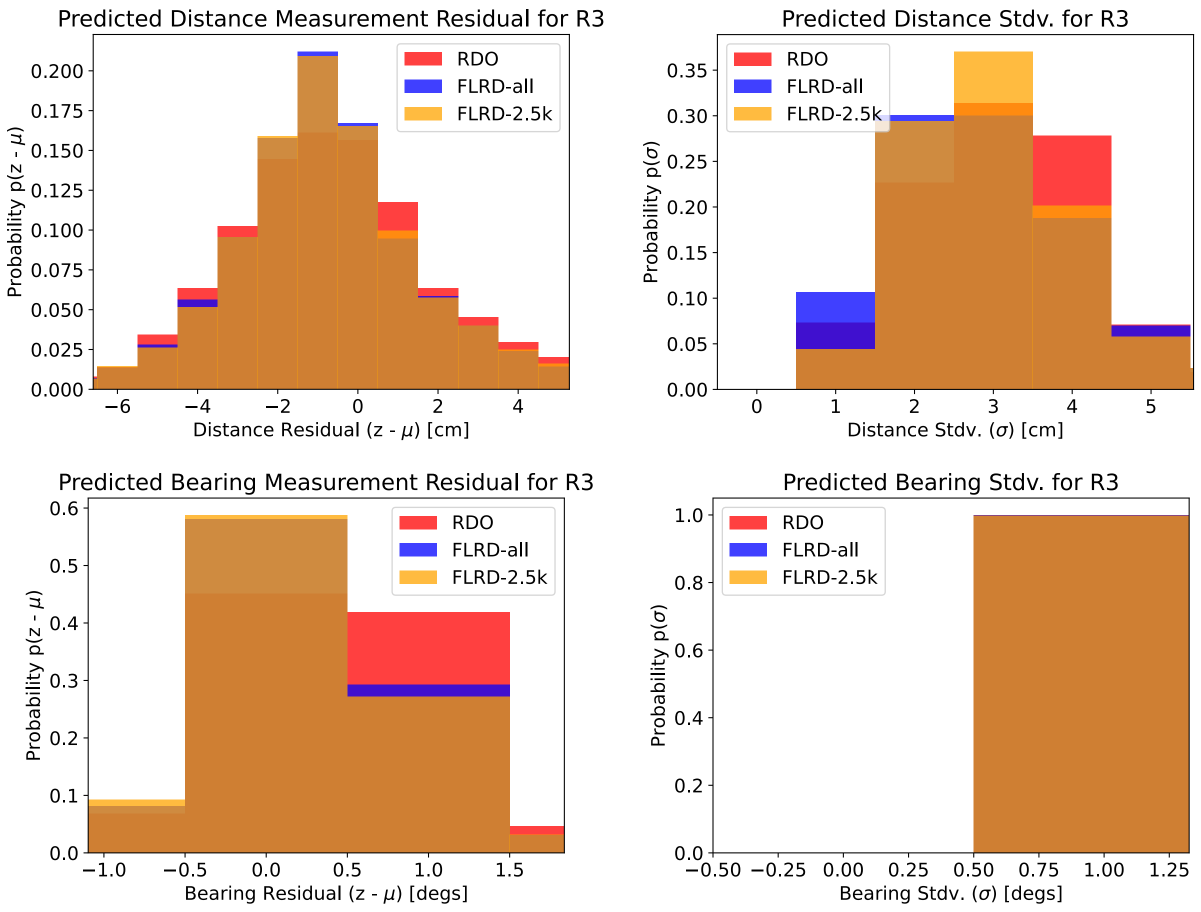


Figure 4.2: Robot 3 predicted distance and bearing residuals and noises for FLRD models. Compared to RDO at the same input states, FLRD predicted similar measurement residuals and noises from sample size “all” down to 2,500 for course 7. Therefore, we do not believe the FLRD localization results for robot 3 diverged due to training divergence. Here, z is a sensor measurement, while μ and σ are derived from the MDN outputs.

Table 4.9: Robot 3 position errors (in centimeters) for the ADO and RDO models on Course 1. The best error is emphasized amongst ADO, RDO, and FLRD models. An asterisk “$*$” denotes the best error(s) for each robot in the following table.

	ADO	RDO
EKF	9.5	5.7 $*$
EKPF	10.8	4.2 $*$

Table 4.10: Robot 3 course 1 position RMSEs (in centimeters) for RDO and FLRD models. The best error is emphasized amongst ADO, RDO, and FLRD models. An asterisk “$*$” denotes the best error(s) for each robot in the following table.

	Real Data Sample Sizes					
	All Real Data	$10,000$	$5,000$	$2,500$	500	100
EKF	**5.3** ± 0.1 $*$	5.3 ± 0.1	5.3 ± 0.1	5.4 ± 0.1	5.7 ± 0.1	6.0 ± 0.4
EKPF	**3.9** ± 0.1 $*$	3.9 ± 0.1	3.9 ± 0.1	4.0 ± 0.1	4.1 ± 0.2	6.6 ± 0.8

likelihood means and standard deviations of all five robots (Table 3.3 in Section 3.6.2.1). When evaluated at the same input states, FLRD predicted similar measurement residuals and noises as RDO at multiple sample sizes (Figure 4.2), indicating that FLRD should have achieved similar errors as RDO. For both filters, FLRD also performed comparably to RDO on course 4 (Tables 4.5 and 4.7) and on course 1 (Tables 4.9 and 4.10), which we used as an auxiliary test course.

We also noticed that robot 4 diverged once for sample size 10,000 (Table 4.6) and seven times for sample size 500 (Table 4.8), causing the RMSE means and standard deviations to increase. However, the remaining 99 and 93 results (for sizes 10,000 and 500, respectively) were comparable to RDO.

We believe the arrangement of the landmarks partially caused the greater differences for course 7. Landmarks were clustered together on opposite ends of the data collection workspace [25]. As a result, the robots might have had difficulty identifying landmarks or localizing since they were clustered instead of uniformly distributed.

Lastly, the results imply that combining an EKF with an SDSMM (RDO or FLRD) may be useful for devices with limited resources or real-time applications, even though this combination did not yield the lowest errors. When using an SDSMM, we observed that the EKF and EKPF achieved the second-lowest and smallest errors, respectively. However, due to the number of computations, some devices or real-time applications may not have the resources or time needed to use an EKPF with an SDSMM. On the other hand, when

Table 4.11: Course 4 EKF position errors (in centimeters) for LRDO models. A shaded cell means that LRDO performed better than FLRD for the same robot and sample size. A dagger (†) emphasizes that the EKF diverged with 10% or more of the LRDO models.

	Real Data Sample Sizes				
	10,000	5,000	2,500	500	100
R1	4.7 ± 0.4	4.3 ± 0.3	5.1 ± 0.8	5.4 ± 1.0	22.3 ± 38.5 †
R2	113.7 ± 928.8 †	22.5 ± 42.0 †	7.5 ± 20.2	10.6 ± 21.3	20.0 ± 16.8 †
R3	6.0 ± 0.1	5.9 ± 0.1	5.9 ± 0.1	6.7 ± 0.4	22.7 ± 26.3 †
R4	39.8 ± 263.7 †	9.0 ± 0.8	9.6 ± 7.8	9.0 ± 1.1	27.1 ± 34.1 †
R5	4.6 ± 0.2	4.6 ± 0.2	4.7 ± 0.2	6.6 ± 6.9	19.3 ± 12.9 †

Table 4.12: Course 7 EKF position errors (in centimeters) for LRDO models. A shaded cell means that LRDO performed better than FLRD for the same robot and sample size. A dagger (†) emphasizes that the EKF diverged with 10% or more of the LRDO models.

	Real Data Sample Sizes				
	10,000	5,000	2,500	500	100
R1	10.0 ± 2.4	9.4 ± 0.9	10.0 ± 1.1	11.0 ± 1.8	23.1 ± 10.9 †
R2	6.6 ± 0.5	7.6 ± 5.1	7.3 ± 0.3	8.8 ± 2.4	23.0 ± 14.4 †
R3	26.0 ± 42.8 †	23.9 ± 42.5 †	33.4 ± 42.3 †	35.8 ± 43.9 †	31.8 ± 22.3 †
R4	32.2 ± 64.2 †	37.2 ± 109.6 †	21.8 ± 34.3 †	38.1 ± 56.9 †	34.6 ± 48.1 †
R5	9.9 ± 15.3	8.2 ± 4.8	8.3 ± 2.6	8.5 ± 1.1	45.4 ± 48.0 †

appropriate, such devices or applications could use an EKF with an SDSMM to trade better accuracy for faster computations.

4.3.2.4 Comparing LRDO and FLRD

FLRD and LRDO were compared to measure the benefit of transfer learning (FLRD) versus learning with limited data only (LRDO). Generally, the results showed that FLRD performed as good or better than LRDO. LRDO had better mean position errors in 8 out of 100 cases (shaded in Tables 4.5, 4.7, 4.6, and 4.8). However, the performance gains in localization were negligible in six cases, ranging from 1 to 2 millimeters. The performance

Table 4.13: Course 4 EKPF position errors (in centimeters) for LRDO models. A shaded cell means that LRDO performed better than FLRD for the same robot and sample size. A dagger (†) emphasizes that the EKPF diverged with 10% or more of the LRDO models.

	Real Data Sample Sizes				
	10,000	5,000	2,500	500	100
R1	3.6 ± 0.4	3.4 ± 0.3	4.1 ± 0.8	4.4 ± 1.1	59.7 ± 50.2 †
R2	4.3 ± 4.7	4.5 ± 3.2	4.2 ± 3.0	9.4 ± 17.9 †	89.3 ± 81.9 †
R3	4.1 ± 0.1	4.0 ± 0.1	4.0 ± 0.1	4.6 ± 0.4	114.7 ± 125.5 †
R4	5.8 ± 0.3	6.0 ± 0.3	6.0 ± 0.3	6.5 ± 0.9	59.9 ± 40.2 †
R5	3.5 ± 0.6	3.6 ± 0.7	3.7 ± 0.7	4.2 ± 0.9	71.5 ± 65.0 †

Table 4.14: Course 7 EKPF position errors (in centimeters) for LRDO models. A shaded cell means that LRDO performed better than FLRD for the same robot and sample size. A dagger (†) emphasizes that the EKF diverged with 10% or more of the LRDO models.

	Real Data Sample Sizes				
	10,000	5,000	2,500	500	100
R1	8.2 ± 4.0	7.5 ± 0.7	8.2 ± 1.1	10.7 ± 13.9 †	87.1 ± 88.4 †
R2	6.8 ± 7.9	7.3 ± 9.7	6.2 ± 0.5	7.8 ± 2.4	90.0 ± 89.9 †
R3	15.6 ± 10.9	11.5 ± 3.8	17.7 ± 21.8 †	22.6 ± 22.0 †	152.1 ± 132.4 †
R4	13.8 ± 20.5 †	13.9 ± 28.0	11.8 ± 11.5	12.4 ± 13.4	63.0 ± 63.4 †
R5	7.5 ± 8.3	6.6 ± 0.6	6.8 ± 0.6	7.5 ± 2.7	45.4 ± 46.9 †

gains for the other two cases were 5 and 11 millimeters, which were at sample size 5,000 for robots 5 and 3 course 7 (respectively); see Table 4.14.

The results also showed a noticeable difference in performance at a sample size of 100, which demonstrated the benefit of the proposed method. At sample size 100, all LRDO localization experiments diverged for both filters (Tables 4.11, 4.13, 4.12, and 4.14). Aside from sample size 100, LRDO also diverged 16 other times for robots 1 through 4.

To understand why LRDO failed localization and FLRD did not fail localization, we compared the predicted measurement residuals and noises for all LRDO and FLRD models at sample size 100 and an RDO model. Qualitatively, Figure 4.3 shows that, at sample size 100, FLRD learned measurement distributions that were close to RDO. However, the LRDO predicted measurement residuals and noises varied significantly from RDO. We also observed that the distance measurements had a bias near -8 centimeters relative to the real sensor's distance measurements. From the figure, we conclude that FLRD learned reasonable measurement distributions at sample size 100 due to transfer learning. Pre-training potentially helped the model learn a relationship between the input states and the output *a priori* measurements. Fine-tuning then helped the model learn the measurement bias and noise for each sensor.

Since LRDO learned relatively poor measurement distributions, we believe the following occurred during localization. During an EKF update, inaccurately high measurement residuals caused over-corrections in the pose. Although high measurement noise reduces the confidence in measurements and, in turn, lessens over-corrections (via the Kalman gain), high noise also reduces relatively smaller (possibly more accurate) pose corrections. Inaccurately high residuals and noises also impacted measurement and transition probabilities, which in turn affected importance weighting. For example, over-corrections might have placed the sampled state $\hat{\mathbf{x}}_t$ far away from the previous state $\mathbf{x}_{t-1}$, producing near-zero transition probabilities. Inaccurately high measurement residuals or noises might have also produced low

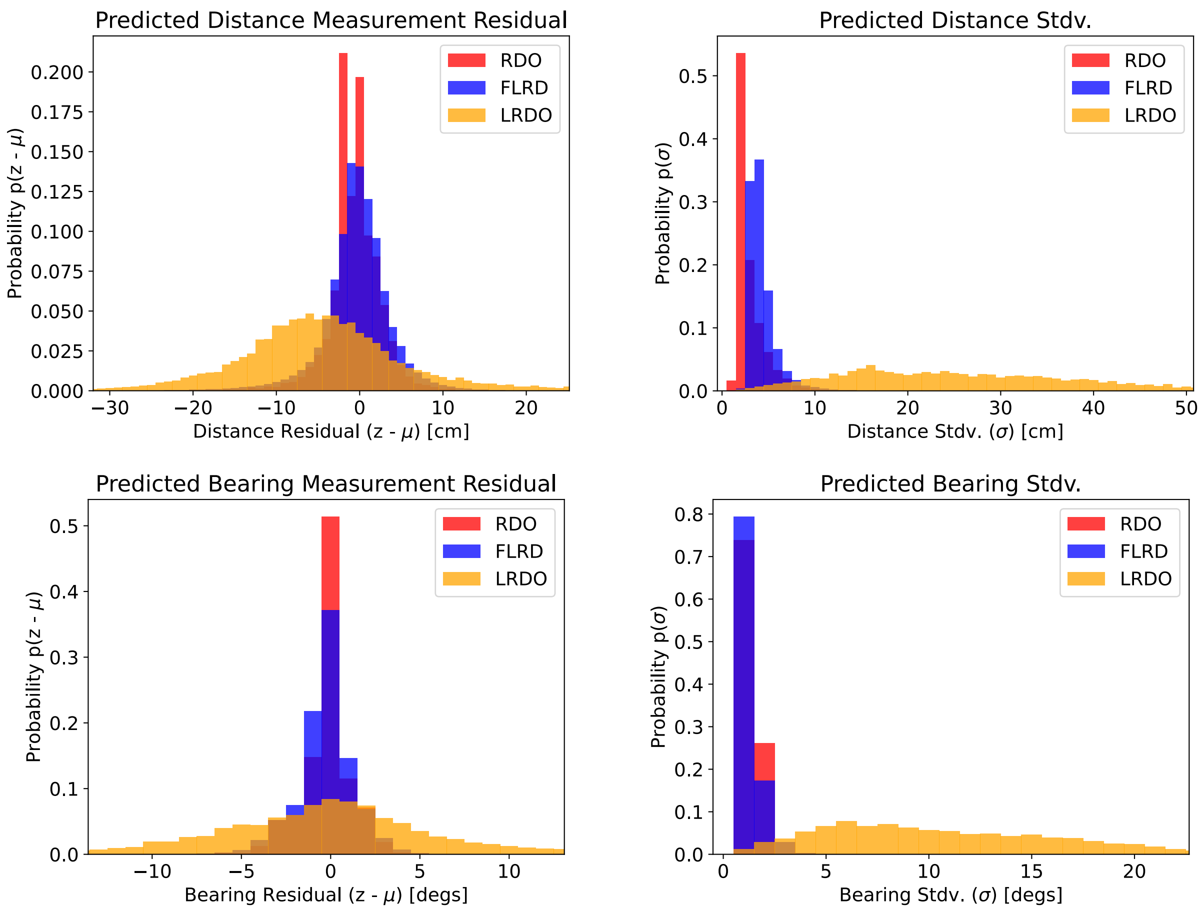


Figure 4.3: Comparing an RDO model with FLRD and LRDO models at sample size 100. The graphs illustrate the test set measurement residuals and noises for Robot 1. Each graph was created using the predictions from all 100 FLRD and LRDO models at sample size 100 and an RDO model. z is a sensor measurement. μ and σ are the MDN outputs. Compared to all LRDO models, the graphs illustrate that all FLRD models (the proposed approach) produced estimates closer to RDO (the prior approach). Specifically, the residuals $(z - \mu)$ and standard deviations (σ) vary significantly for LRDO. This figure shows that transfer learning helped FLRD achieve performance comparable to RDO with fewer data and produce more accurate estimates than LRDO at a small sample.

or near-zero measurement probabilities, causing particles to have similar probabilities and degrade the performance of the filter.

4.4 Real-World Experiments II: A Long-term Localization Study

4.4.1 Experimental Setup

We ran localization experiments using all sessions from the University of Michigan's NCLT dataset [6]. We used the pole-detection method and the Particle Filter (PF) proposed in Schaefer *et al.* [47] as our baseline. Schaefer *et al.* [47] defined the PF state transition model as a trivariate normal distribution with Gaussian motion noise $\mathbf{R}$: $\bar{\mathbf{x}}_t^{(i)} \sim \mathcal{N}(\bar{\mathbf{x}}_{t-1}^{(i)}, \mathbf{R})$. The measurement model in [47] was a pole-detection algorithm that used a Velodyne HDL-32E LiDAR sensor, which was mounted atop a Segway robot. Given a point cloud captured using the LiDAR sensor, the pole-detection method outputted four (4) parameters for each pole detected within the cloud. We refer to these four parameters as a pole measurement. Each pole measurement $\mathbf{z}_{t,k}$ was composed of the 2D Cartesian position of the pole relative to the Segway robot (z_x, z_y), the width of the pole z_w, and a pole score z_s: $\mathbf{z}_{t,k} = \{z_x, z_y, z_w, z_s\}$. Here, t referred to the timestamp of the LiDAR point cloud, while k referred to the index of a pole that was observed in the point cloud. The PF measurement probability was defined as

$$p(\mathbf{z}_{t,k} \mid \bar{\mathbf{x}}_t^{(i)}, \mathbf{m}_k) = \mathcal{N}\left(\left|\left| \mathit{transform}(\mathbf{z}_{t,k}^{[x,y]}, \bar{\mathbf{x}}_t^{(i)}) - \mathbf{m}_k \right|\right|, \sigma\right). \tag{4.28}$$

The function $\mathit{transform}(\mathbf{z}_{t,k}^{[x,y]}, \bar{\mathbf{x}}_t^{(i)})$ transformed the pole position $\mathbf{z}_{t,k}^{[x,y]}$ from the vehicle coordinate frame to world coordinate frame, outputting the expected pole position. $\mathbf{m}_k$ is the position of a pole in the global map that was associated with the pole observed in the measurement $\mathbf{z}_{t,k}$. All poles in the global map are defined in the world coordinate frame. Finally, σ is the isotropic pole position noise. In [47], the pole position noise was a fixed value of $\sigma^2 = 1.5$ meters. The authors used a KD-tree to perform data association given the expected pole position in the world coordinate frame. The KD-tree outputted a pole in the global map that was closest to the expected pole position for each input.

Our proposed measurement models combined the baseline measurement model with an SDSMM. We labeled the SDSMM-augmented LPDs as LPD-CD, LPD-1Day, and LPD-9Days. Each SDSMM represented an MDN that was trained under a particular scenario. LPD-CD, where "CD" means "clear days", was trained using data collected during clear weather and ample daylight. LPD-1Day was trained with data from the first session (2012-01-08). For LPD-9Days, we fine-tuned LPD-1Day using data from the following nine sessions (2012-01-15 through 2012-03-17). The training details for each model was described in Section 2.8. Each SDSMM learned to predict the state-dependent measurement bias and noise given a pole measurement from the baseline model. That is, a combined state $\boldsymbol{\lambda}_{t,k}$ for these experiments was a pole measurement $\mathbf{z}_{t,k}$: $\boldsymbol{\lambda}_{t,k} = \mathbf{z}_{t,k}$. We then used the predicted bias $\boldsymbol{\mu}_{t,k}$ (the MDN output mean) to augment the expected pole position from the baseline. Mathematically, our measurement probability was defined as

$$p(\mathbf{z}_{t,k} \mid \bar{\mathbf{x}}_t^{(i)}, \mathbf{m}_k) = \mathcal{N}\left(\left|\left| transform(\mathbf{z}_{t,k}^{[x,y]} + \boldsymbol{\mu}_{t,k}, \bar{\mathbf{x}}_t^{(i)}) - \mathbf{m}_k \right|\right|, \max(diag(\boldsymbol{\Sigma}_{t,k}))\right), \quad (4.29)$$

where $[\boldsymbol{\mu}_{t,k}, \boldsymbol{\Sigma}_{t,k}] = g(\boldsymbol{\lambda}_{t,k}; \mathbf{w})$ and $\max(diag(\boldsymbol{\Sigma}_{t,k}))$ extracts the maximum variance value along the main diagonal of the state-dependent measurement noise $\boldsymbol{\Sigma}_{t,k}$. We computed the measurement noise using this method to mimic the isotropic measurement noise in [47].

4.4.2 Results and Discussion

4.4.2.1 Localization Metrics

For each session, we computed a root-mean-squared error (RMSE) and a mean absolute error (MAE) for both the position and angle errors. The RMSE and MAE were computed between the ground truth and estimated states. The estimated states came from the Particle Filter. To derive the estimated states at time t, we followed Schaefer *et al.* [47] in computing the mean of the best 10% of the particles. The ground truth states came from the SLAM solution that Carlevaris-Bianco *et al.* [6] derived for the NCLT dataset.

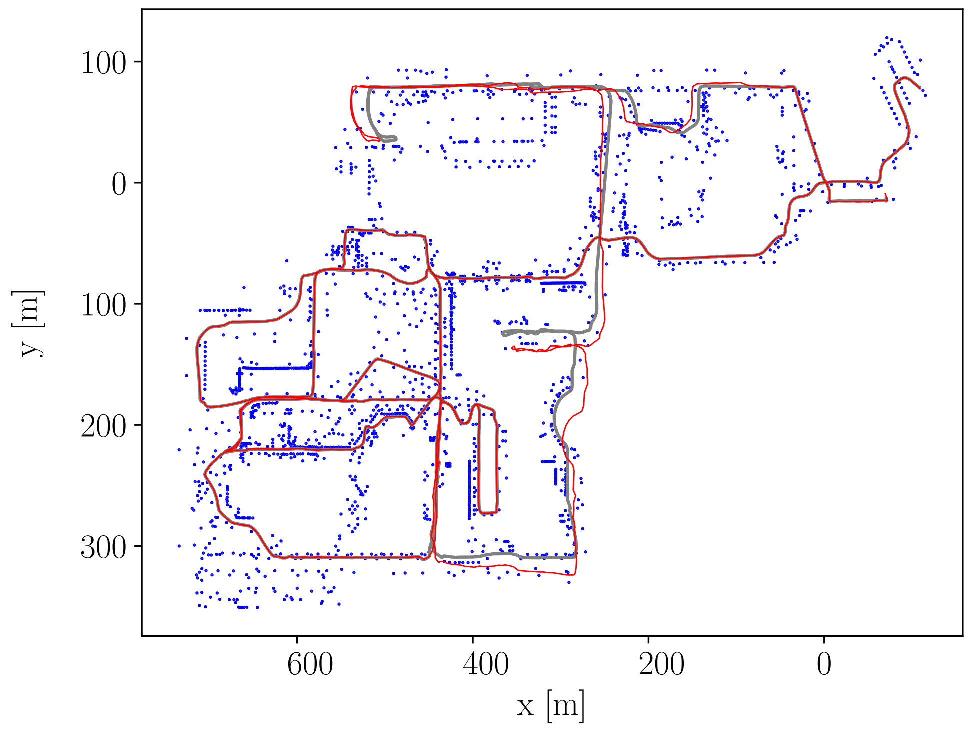
4.4.2.2 Comparing LPD and LPD-CD

We compared LPD (the baseline) and the SDSMM-augmented LPDs to measure the differences in performance from incorporating the measurement bias (LPD-CD$_{\boldsymbol{\mu}}$), the measurement noise (LPD-CD$_{\boldsymbol{\Sigma}}$), and both (LPD-CD with *no* subscript). Table 4.15 shows the position RMSEs for each session using the four measurement models. Through these results, we observed three findings. Like in [58], we first observed that employing the state-dependent bias, noise, or both generally reduced the localization position RMSEs (Tables 4.15 and 4.16). LPD-CD$_{\boldsymbol{\mu}}$ achieved lower position RMSEs than LPD (the baseline) in 9 out of 23 sessions. LPD-CD$_{\boldsymbol{\Sigma}}$ achieved lower position RMSEs than LPD in 20 out of 23 sessions, performing relatively better than LPD-CD$_{\boldsymbol{\mu}}$. Together, both models performed the best (in terms of position RMSEs) in 6 out of 23 sessions, five of which were achieved by LPD-CD$_{\boldsymbol{\Sigma}}$.

By comparing the results for LPD-CD$_{\boldsymbol{\mu}}$ and LPD-CD$_{\boldsymbol{\Sigma}}$ alone, our second observation was that predicting measurement noise produces lower position RMSEs than predicting measurement bias. This observation suggests that predicting an accurate measurement noise may be more important for this experiment, yielding more accurate localization. In 20 out of 23 sessions, LPD-CD$_{\boldsymbol{\Sigma}}$ outperformed LPD-CD$_{\boldsymbol{\mu}}$. Relative to LPD-CD$_{\boldsymbol{\mu}}$, LPD-CD$_{\boldsymbol{\Sigma}}$ decreased the position RMSEs by almost 20% on average. In the best case, LPD-CD$_{\boldsymbol{\Sigma}}$ decreased the position RMSE by as much as 98% (for session 2013-02-23). However, this trend does not hold true for all sessions. Specifically, LPD-CD$_{\boldsymbol{\mu}}$ outperformed LPD-CD$_{\boldsymbol{\Sigma}}$ in three sessions: 2012-01-15, 2012-05-26, and 2012-11-16. For sessions 2012-01-15 and 2012-05-26, the difference in position RMSEs were small: 1.1 centimeters for the former and 0.7 centimeters for the latter. For session 2012-11-16, we observed an almost 400% increase in the position RMSE for LPD-CD$_{\boldsymbol{\Sigma}}$.

Table 4.15: Testing set position RMSEs (in centimeters) for the baseline and SDSMM-augmented measurement models. The baseline was the standalone LPD, while the SDSMM-augmented measurement models were LPD-CD$_{\boldsymbol{\mu}}$, LPD-CD$_{\boldsymbol{\Sigma}}$, and LPD-CD. An asterisk "$*$" denotes the best error for each session.

Sessions	Measurement Models			
	LPD	LPD-CD$_{\boldsymbol{\mu}}$	LPD-CD$_{\boldsymbol{\Sigma}}$	LPD-CD
2012-01-08	17.8	18.3	17.2	15.8 $*$
2012-01-15	22.5	21.6 $*$	22.7	21.7
2012-01-22	22.2	21.6	19.9	19.0 $*$
2012-02-12	100.5	100.5	100.0 $*$	100.1
2012-02-18	22.1	21.9	18.6 $*$	18.8
2012-02-19	19.4	19.2	19.1	17.5 $*$
2012-03-25	26.2	26.6	24.3	24.1 $*$
2012-03-31	18.4	18.2	16.6	16.0 $*$
2012-04-29	25.1	25.1	22.5	22.2 $*$
2012-05-11	22.5	22.1	19.5	18.9 $*$
2012-05-26	21.7	21.0	21.7	20.9 $*$
2012-06-15	23.8	23.8	20.2	18.6 $*$
2012-08-04	34.0	32.4	24.5	23.4 $*$
2012-08-20	26.4	26.5	21.9	19.5 $*$
2012-09-28	31.1	31.9	22.4	21.7 $*$
2012-10-28	33.8	35.7	28.8 $*$	28.8 $*$
2012-11-04	45.6	46.9	27.2 $*$	29.1
2012-11-16	72.2	80.1	399.9	46.1 $*$
2012-11-17	37.7	38.3	35.3	32.7 $*$
2012-12-01	49.2	49.9	35.8	34.6 $*$
2013-01-10	27.8	28.6	17.6	16.8 $*$
2013-02-23	548.0	2905.6	55.0	54.3 $*$
2013-04-05	92.0	91.8	86.0 $*$	86.7



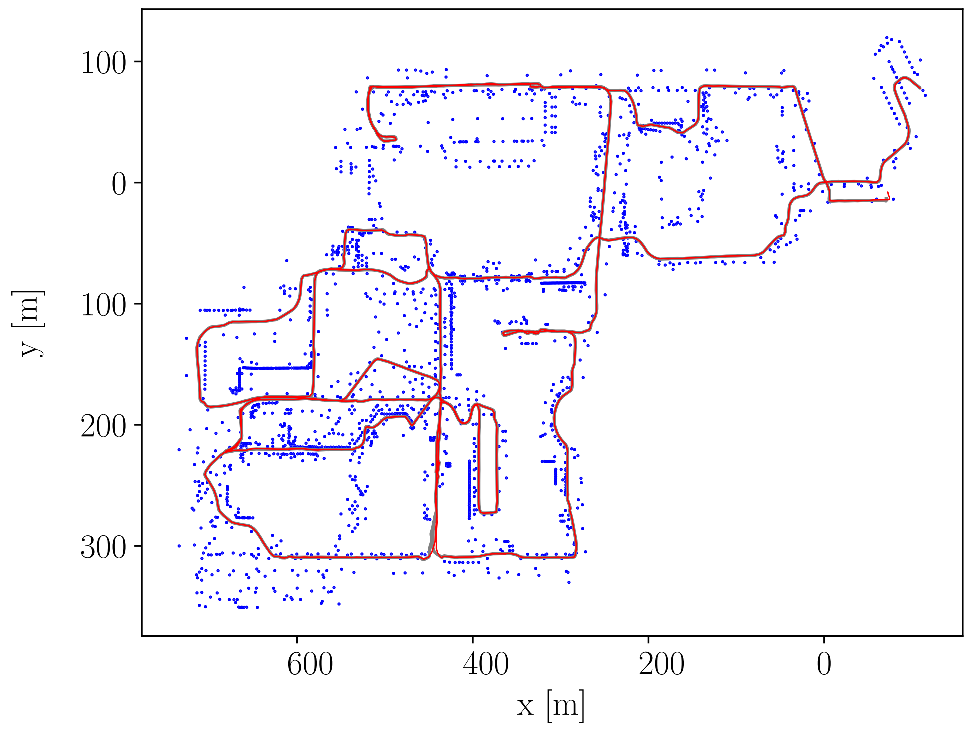


Figure 4.4: LPD and LPD-CD trajectories during session 2013-02-23. The LPD trajectory (top graph) drifted away from the ground truth in the middle and top of the graph. However, LPD-CD (the SDSMM-augmented model) trajectory (bottom graph) remained close to the ground truth. Estimated trajectories are red, ground-truth trajectories are gray, and landmarks are blue.

Table 4.16: Testing set angle RMSEs (in degrees) for the baseline and SDSMM-augmented measurement models. The baseline was a standalone LPD, while the SDSMM-augmented measurement models were LPD-CD$_{\boldsymbol{\mu}}$, LPD-CD$_{\boldsymbol{\Sigma}}$, and LPD-CD. An asterisk "$*$" denotes the best error for each session.

Sessions	Measurement Models			
	LPD	LPD-CD$_{\boldsymbol{\mu}}$	LPD-CD$_{\boldsymbol{\Sigma}}$	LPD-CD
2012-01-08	0.857	0.815	0.87	0.768 $*$
2012-01-15	0.999	0.975	1.017	0.949 $*$
2012-01-22	1.291	1.273 $*$	1.316	1.273 $*$
2012-02-12	1.04	1.005	1.059	0.994 $*$
2012-02-18	0.938	0.895	0.952	0.884 $*$
2012-02-19	0.944	0.909	0.956	0.889 $*$
2012-03-25	1.836	1.837	1.82	1.815 $*$
2012-03-31	0.973	0.953	0.99	0.933 $*$
2012-04-29	1.079	1.031	1.078	1.018 $*$
2012-05-11	0.998	0.96 $*$	1.045	0.973
2012-05-26	0.889	0.89	0.924	0.875 $*$
2012-06-15	0.879	0.857	0.904	0.834 $*$
2012-08-04	1.143	1.138	1.117	1.094 $*$
2012-08-20	0.941	0.928	0.934	0.882 $*$
2012-09-28	0.952	0.926	0.948	0.877 $*$
2012-10-28	0.919	0.897	0.924	0.851 $*$
2012-11-04	0.996	0.995	0.958	0.916 $*$
2012-11-16	2.031	2.075	2.43	1.983 $*$
2012-11-17	0.959	0.95	0.91	0.84 $*$
2012-12-01	0.93	0.943	0.89	0.825 $*$
2013-01-10	0.911	0.886	0.899	0.823 $*$
2013-02-23	1.769	6.205	0.88	0.819 $*$
2013-04-05	1.028	1.001	1.028	0.935 $*$

Our third observation was that LPD-CD (which used both bias and noise) achieved the lowest position RMSEs for 17 out of 23 sessions (or about 75% of the sessions). Compared to LPD, LPD-CD$_{\boldsymbol{\mu}}$, and LPD-CD$_{\boldsymbol{\Sigma}}$, LPD-CD reduced the position RMSE by 23.9%, 24.6%, and 9.7% on average (respectively) during the sessions where LPD-CD had the lowest position RMSE. Figure 4.4 depicts one session where LPD-CD outperformed LPD significantly. In cases where LPD-CD did not achieve the lowest position RMSE, the performance difference was sub-centimeter (for sessions 2012-01-15, 2012-02-18, 2012-10-28, and 2013-04-05) or less than two centimeters (for session 2012-11-04). Therefore, LPD-CD arguably achieved the lowest position RMSEs overall.

As for the angle RMSEs (Table 4.16), our fourth observation was that LPD-CD achieved the lowest errors for 22 out of 23 test sessions. However, aside from session 2013-02-23, there was a negligible, sub-degree performance difference amongst the four measurement models across all 23 test sessions. For session 2013-02-23, LPD and LPD-CD$_{\boldsymbol{\mu}}$ had larger angle RMSEs than the other two models.

4.4.2.3 Comparing LPD-CD, LPD-1Day, and LPD-9Days

We compared the LPD-CD, LPD-1Day, and LPD-9Days models to measure the differences in performance as the data collection conditions changed. Since Section 4.4.2.2 demonstrated that using both measurement bias and noise achieved the lowest errors, our discussion focused on such implementation for the three models. From our analysis, we identified three findings. Our first finding was all three models demonstrated improvement over the baseline LPD, especially during the last eleven days (Figure 4.5). During the last eleven days, the LPD-CD, LPD-1Day, and LPD-9Days reduced the position RMSE by a median of 10.6, 10.9, and 10.5 centimeters, respectively. In the extreme case, the three models reduced the position RMSE by 493 to 495 centimeters for session 2013-02-23.

Our second finding was that the three models achieved similar position RMSEs for most sessions. The median difference between the best and worst position RMSES for each session

Table 4.17: Testing set position RMSEs (in centimeters) for the LPD-CD, LPD-1Day, and LPD-9Days models. An asterisk “$*$” denotes the best error for each session. Training sessions (for any model) were omitted. Aside from LPD-1Day during session 2012-05-26, the three SDSMM-augmented models performed better than the baseline LPD.

Sessions	Measurement Models		
	LPD-CD	LPD-1Day	LPD-9Days
2012-03-25	24.1 $*$	25.4	24.4
2012-03-31	16.0 $*$	16.1	16.3
2012-04-29	22.2 $*$	22.4	22.4
2012-05-11	18.9 $*$	20.9	19.0
2012-05-26	20.9 $*$	39.3	21.1
2012-06-15	18.6 $*$	20.3	19.1
2012-08-04	23.4	23.2 $*$	23.6
2012-08-20	19.5	22.3	19.3 $*$
2012-09-28	21.7	21.5	19.6 $*$
2012-10-28	28.8	28.3	28.2 $*$
2012-11-04	29.1 $*$	30.3	32.8
2012-11-16	46.1	47.5	44.3 $*$
2012-11-17	32.7	32.6	32.0 $*$
2012-12-01	34.6	33.5 $*$	34.2
2013-01-10	16.8	16.1	16.0 $*$
2013-02-23	54.3	53.5 $*$	54.1
2013-04-05	86.7	85.5 $*$	86.3

Table 4.18: Testing set angle RMSEs (in degrees) for the LPD-CD, LPD-1Day, and LPD-9Days models. An asterisk "∗" denotes the best error for each session. Training sessions (for any model) were omitted.

Sessions	Measurement Models		
	LPD-CD	LPD-1Day	LPD-9Days
2012-03-25	1.815 ∗	1.821	1.823
2012-03-31	0.933 ∗	0.933 ∗	0.938
2012-04-29	1.018	1.015 ∗	1.016
2012-05-11	0.973 ∗	0.981	0.976
2012-05-26	0.875 ∗	0.939	0.878
2012-06-15	0.834 ∗	0.859	0.834 ∗
2012-08-04	1.094	1.075 ∗	1.085
2012-08-20	0.882 ∗	0.899	0.888
2012-09-28	0.877 ∗	0.882	0.884
2012-10-28	0.851	0.852	0.850 ∗
2012-11-04	0.916	0.913 ∗	0.919
2012-11-16	1.983	1.982	1.971 ∗
2012-11-17	0.840	0.826 ∗	0.837
2012-12-01	0.825	0.820 ∗	0.823
2013-01-10	0.823	0.832	0.818 ∗
2013-02-23	0.819 ∗	0.820	0.820
2013-04-05	0.935 ∗	0.938	0.937

Table 4.19: Differences in position RMSEs (in centimeters) for sessions that had the same or similar conditions to session 2012-05-26.

Sessions	Data Collection Conditions				Differences in
	Time	Sky	Foliage	Snow	Position RMSE
2012-05-26	Evening	Sunny	Yes	No	18.4
2012-08-20	Evening	Sunny	Yes	No	3.0
2012-09-28	Evening	Sunny	Yes	No	2.1
2012-02-18	Evening	Sunny	No	No	0.3
2012-11-16	Evening	Sunny	No	No	3.2
2012-12-01	Evening	Sunny	No	No	1.1

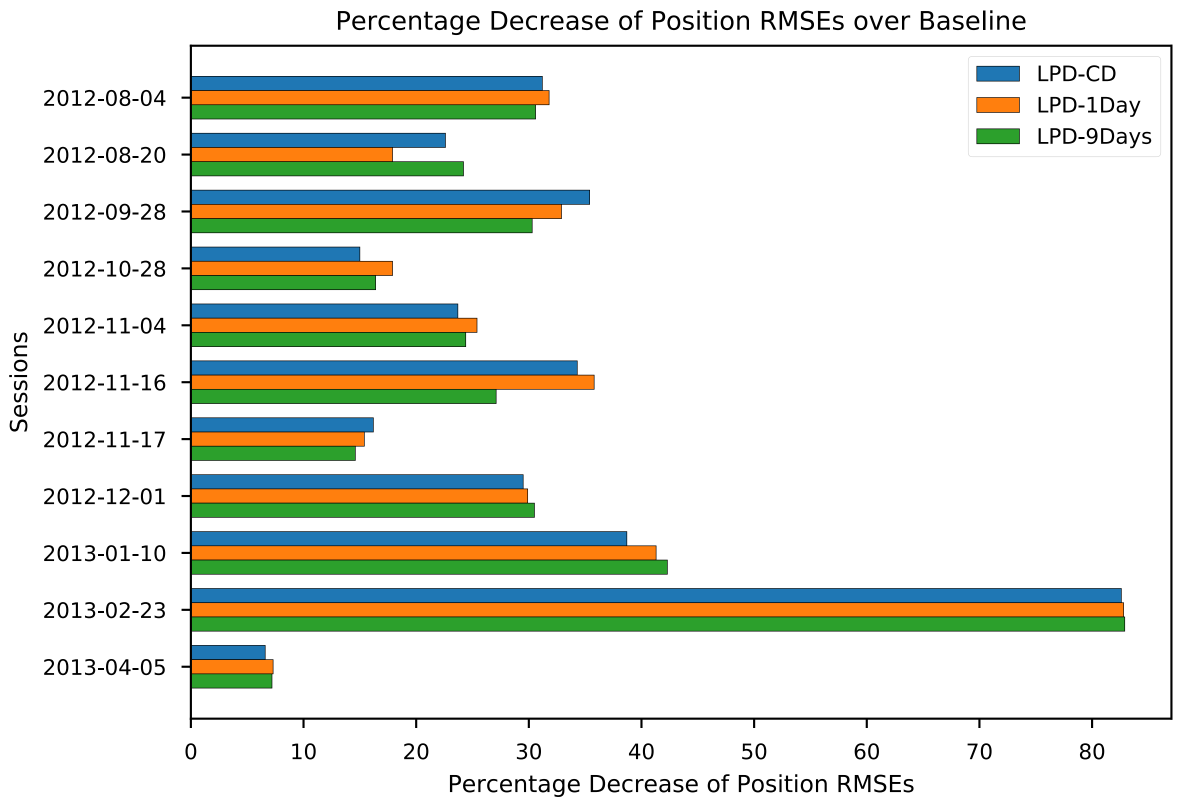
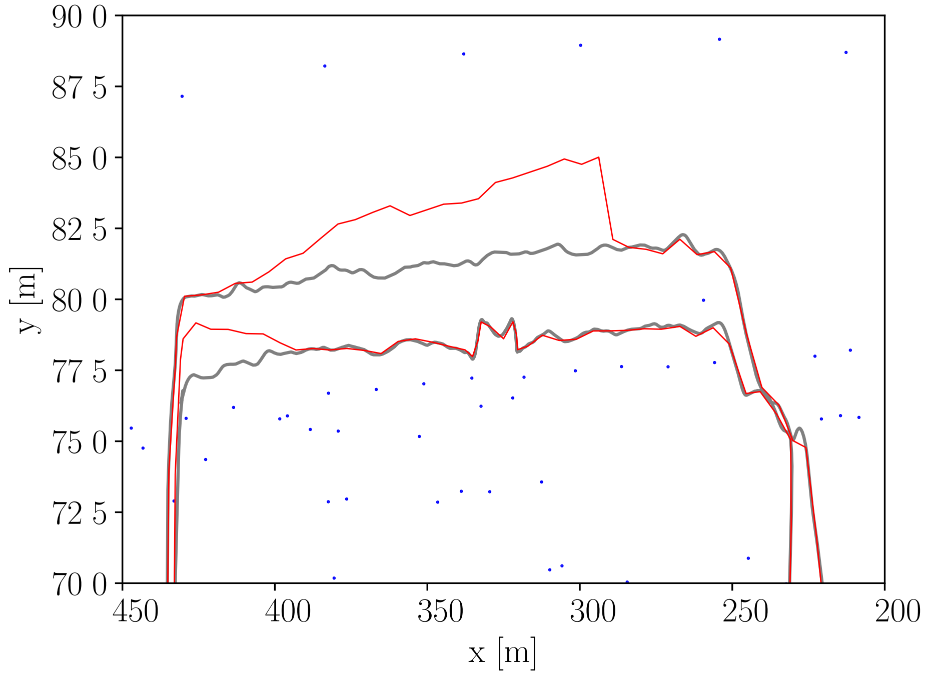


Figure 4.5: Percentage decrease of position RMSEs over baseline LPD for the last eleven sessions. This figure depicts how much the SDSMM-augmented LPD models (LPD-CD, LPD-1Day, and LPD-9Days) decreased the position RMSEs when compared to the baseline.



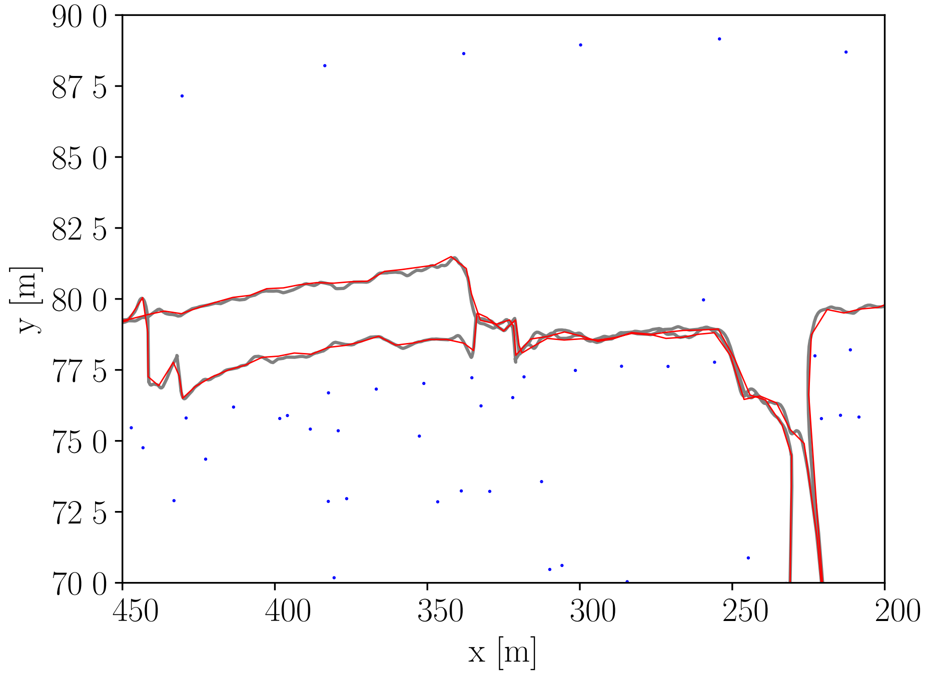


Figure 4.6: Differences in performance for LPD-1Day on days that had the same data collection conditions. This figure depicts the LDP-1Day trajectories for sessions 2012-05-26 (top) and 2012-08-20 (bottom). The bottom graph shows that the model performed better along a similar trajectory on different days with the same data collection conditions. In both graphs, the ground-truth trajectory is gray, the estimated trajectory is red, and the blue dots represent pole landmarks.

was 1.2 centimeters (Table 4.17), while the 75th percentile difference was 2.1 centimeters. There were three sessions where the difference in position RMSEs were greater than 2.1 centimeters: 2012-11-04, 2012-11-16, and 2012-05-26. For sessions 2012-11-04 and 2012-11-16, the worst and best performing models were not the same, which did not indicate a best or worst performing model. There was also a stark difference in performance for session 2012-05-26. The LPD-1Day model had a position RMSE that was almost $2\times$ greater than LPD-CD and LPD-9Days. However, we believe the stark difference in position RMSE is an outlier. We came to this conclusion by analyzing the differences in position RMSEs for five sessions that had the same or similar data collection conditions as session 2012-05-26. For the five sessions, we observed that the position RMSEs were comparable amongst LPD-1Day and the other two models (Table 4.19). We also examined the sessions (in Table 4.19) where the robot traversed similar trajectories in which the error occurred (for example, Figure 4.6). The results show that the robot trajectory was closer to the ground truth during the other sessions.

Our final finding was that all three models had comparable angle RMSEs (Table 4.18). Although LPD-CD achieved the lowest angle RMSE for most sessions, the performance differences amongst all three models were less than one degree.

Chapter 5: Conclusion and Future Work

This chapter concludes this dissertation by reviewing the discussions in each chapter, summarizing our experiments, and describing our findings (Section 5.1). Afterward, we discuss potential future work (Section 5.2).

5.1 Conclusion

This dissertation introduced a novel method called *state-dependent sensor measurement models* (SDSMM). An SDSMM learns to predict the measurement bias and noise of a sensor given the states of a robot and its environment. In Chapter 1, we described the motivation behind this dissertation. Then we discussed the related work, including methods for learning sensor measurement models and learning such models with limited data.

Chapter 2 discussed SDSMMs in detail and described how to implement SDSMMs using Mixture Density Networks (MDNs). Instead of using a traditional MDN with a common variance, we employed an MDN with a full covariance matrix, which increases the expressiveness of the MDN. We also analyzed how the dimension growth and time complexity of an MDN as we increase the number of components are in the distribution or the number of dimensions in the distribution.

We evaluated our proposed method using a simulated experiment and two real-world experiments. We quantitatively and qualitatively showed that the MDN learned to predict the true measurement bias and noise for the simulated experiment. The first real-world experiment discussed how to learn an SDSMM for a camera-based range-and-bearing sensor measurement model. We evaluated our proposed method against an *a priori* sensor model (the baseline) using an indoor multi-robot localization and mapping dataset. The results showed that our proposed method learned better expected measurements (which incorporated the measurement bias) and measurement noises compared to the *a priori* model. Particularly, our quantitative results showed that the error between the *a priori* expected measurement and real sensor measurement was $3.4\times$ to $5.7\times$ *larger* than the MDN errors.

Our second real-world experiment discussed how to learn an SDSMM for a LiDAR-based pole extraction sensor model. We used a state-of-the-art pole extractor (LPD) as the baseline. An SDSMM was trained to compensate for the measurement error in the baseline method. We then evaluated the baseline and SDSMM against the standalone baseline method using a long-term indoor and outdoor vehicle localization dataset. The results showed that our proposed method (combining the LPD and an SDSMM) produced a more accurate measurement model compared to a standalone LPD. The results also implied that the fixed isotropic noise might be an overestimate of the actual measurement noise.

In Chapter 3, we proposed a novel extension to the state-dependent sensor measurement model (SDSMM) framework. This extension is a two-stage method that leverages transfer learning to train SDSMMs with limited real data. This method can alleviate collecting an abundance of sensor and ground truth data to learn accurate sensor models and reduce poor generalization.

We evaluated the proposed method using two real-world experiments. The first real-world experiment demonstrated how to use our proposed method to pre-train an SDSMM with artificial sensor data. The artificial sensor data came from the physical model of a distance-and-bearing sensor. We then used multiple, limited numbers of real sensor data from a real distance-and-bearing sensor to fine-tune the pre-trained SDSMM. To simulate limited numbers of real sensor data for each robot, we used the bootstrap method to sample datasets of size 10,000 down to 100. Finally, we compared the proposed method (FLRD) with three other types of SDSMMs. The other SDSMMS were trained with artificially-generated data only (ADO), all available real data only (RDO), or limited real data without pre-training (LRDO). The results showed that FLRD learned measurement models that were similar to RDO, implying that we can learn accurate measurement models with as little as $\sim$19% of the real data. Furthermore, compared with ADO, FLRD produced more accurate predictions since FLRD learned using the real sensor data while ADO did not. Finally, FLRD learned slightly better measurement models than LRDO for large sample sizes. However, FLRD outperformed LRDO at small sample sizes.

Our proposed method broadly implies that we can learn accurate SDSMM with limited real data, reducing the burden of collecting large real-world datasets. With less difficulty, researchers or organizations can develop SDSMMs relatively quickly or with less hassle.

Finally, Chapter 4 demonstrated how to integrate state-dependent sensor measurement models with three state estimation algorithms. These algorithms were the Extended Kalman Filter, the Particle Filter, and the Extended Kalman Particle Filter. We discussed the assumptions of our state estimators, environment, and robot. Afterward, we provided a list of equations that described how to use an SDSMM with each state estimator.

To demonstrate that SDSMMs can reduce localization error, we used SDSMMs and baseline models in two real-world localization experiments. The first real-world experiment examined the performance of SDSMMs with an indoor, multi-robot localization study. We used the Extended Kalman Filter and the Extended Kalman Particle Filter as our state estimators. Both state estimators used *a priori* and state-dependent distance-and-bearing sensor models, which were learned in Section 3.6. The results showed that the SDSMMs that were trained with real data only (RDO) and fine-tuned with real data (FLRD) outperformed the baseline, SDSMMs trained with simulated data (ADO), and SDSMMs trained with limited real data only (LRDO). Furthermore, the results imply that employing an EKF with an SDSMM (RDO or FLRD) may be useful for robots with limited resources even though the EKF achieved the second-lowest errors.

Our second real-world experiment examined the performance of SDSMMs with an indoor and outdoor long-term localization study. For this experiment, we used the Particle Filter as our state estimator. The Particle Filter used a baseline and SDSMM-augmented LiDAR-based pole detection sensor models during localization. We described how to train the SDSMMs in Sections 2.8.2.2 and 2.8.2.3. Like in [47], the results showed that using the SDSMM predicted bias, noise, or both yields the lowest error. The results also showed that using both the bias and noise generally outperformed a model that used bias or noise only. In another experiment, we compared three SDSMMs that were trained under different conditions. The data collection conditions (and their resulting models) were named "clear days", "1Day", and "9Days." Clear days represented a model trained on sessions (days) with clear skies, clear weather, and ample lighting. Next, 1Day represented a model trained on the first session in the dataset. Finally, 9Days represented the 1Day model that was fine-tuned with the following nine sessions. We observed that the three models decreased the position RMSEs by median errors of 25% to 28% during the last eleven sessions compared to the baseline. We also observed that the three models had comparable position and angle RMSEs. Amongst the three models, 75% of the differences between the best and worst position RMSEs were at or below 2.1 centimeters. In one case, the 1Day had a position RMSE that was 2× greater than the other two models. However, we did not observe a comparable difference in performance for sessions with similar data collection conditions or trajectories. Therefore, we believe this case was an outlier.

Although the three models performed comparable, the "9Days" might be most valuable and practical. The training data for the 9Days model was the most diverse in terms of conditions, which improves generalization. Furthermore, the training data represented a practical scenario where one would collect training data over sequential sessions.

5.2 Future Work

Presently, our work assumes that our datasets are diverse and, in some cases, abundant. However, this assumption does not hold in practice. The datasets that we (the robotics community) aggregate do not reflect all the scenarios (or states) a sensor will experience (for example, weather conditions, times of day, geography, and other environmental characteristics will vary). Therefore, a natural progression would be to collect data that the model has not experienced. As mentioned before, the task of data collection, including ground-truth information, is tedious, complicated, time-consuming, and resource-intensive. To alleviate the burden of collecting *all* novel data, we can randomly choose new experiences to collect, focusing on gathering ground-truth data from those experiences as well. However, random selection may not improve the performance of an SDSMM over time, potentially wasting time and resources.

Active learning is one principled technique for selecting which data (including ground-truth) to obtain to improve the performance of an SDSMM. Active learning is a technique in machine learning in which a learning algorithm can ask an oracle to provide a label to new observations interactively. This technique is helpful in cases where we want to improve the performance of our model, but we only have access to unlabelled data. We believe that active learning would help improve state-dependent sensor measurement models over time.

References

[1] Pieter Abbeel, Adam Coates, Michael Montemerlo, Andrew Y. Ng, and Sebastian Thrun. Discriminative Training of Kalman Filters. In *Robotics: Science and Systems*, 2005.

[2] Ethem Alpaydin. *Introduction to Machine Learning.* The MIT Press, 2014.

[3] Chris M. Bishop. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 7(1):108–116, 1 1995.

[4] Christopher M Bishop. Mixture Density Networks. Technical report, Aston University, 1994.

[5] Martin Brossard, Axel Barrau, and Silvère Bonnabel. AI-IMU Dead-Reckoning. 2019.

[6] Nicholas Carlevaris-Bianco, Arash K. Ushani, and Ryan M. Eustice. University of Michigan North Campus long-term vision and lidar dataset. *International Journal of Robotics Research*, 35(9), 2016.

[7] Amitava Chatterjee. Differential evolution tuned fuzzy supervisor adapted extended Kalman filtering for SLAM problems in mobile robots. *Robotica*, 27(3), 2009.

[8] Sungjoon Choi, Kyungjae Lee, Sungbin Lim, and Songhwai Oh. Uncertainty-Aware Learning from Demonstration Using Mixture Density Networks with Sampling-Free Variance Modeling. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 6915–6922, 2018.

[9] B. Csáji. Approximation with artificial neural networks. *MSc. thesis*, 2001.

[10] Bradley Efron. Second Thoughts on the Bootstrap. *Statist. Sci.*, 18(2):135–140, 9 2003.

[11] Bradley Efron, David Rogosa, and Robert Tibshirani. Resampling Methods of Estimation. In *International Encyclopedia of the Social & Behavioral Sciences*, pages 492–495. 9 2015.

[12] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[13] Igor Gilitschenski, Roshni Sahoo, Wilko Schwarting, Alexander Amini, Sertac Karaman, and Daniela Rus. Deep Orientation Uncertainty Learning based on a Bingham Loss. In *International Conference on Learning Representations*, 2020.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[15] Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop KF: Learning discriminative deterministic state estimators. *Advances in Neural Information Processing Systems*, (Nips):4383–4391, 2016.

[16] Humphrey Hu and George Kantor. Parametric covariance prediction for heteroscedastic noise. In *IEEE International Conference on Intelligent Robots and Systems*, 2015.

[17] Rico Jonschkowski, Divyam Rastogi, and Oliver Brock. Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors. 2018.

[18] Guoliang Kang, Jun Li, and Dacheng Tao. Shakeout: A New Approach to Regularized Deep Neural Network Training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1245–1258, 5 2018.

[19] Alex Kendall and Yarin Gal. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In *Advances in Neural Information Processing Systems*, volume 2017-December, 2017.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 12 2014.

[21] Jonathan Ko and Dieter Fox. GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots*, 27(1), 2009.

[22] K. Kobayashi, K.C. Cheok, K. Watanabe, and F. Munekata. Accurate differential global positioning system via fuzzy logic Kalman filter sensor fusion technique. *IEEE Transactions on Industrial Electronics*, 45(3), 6 1998.

[23] Solomon Kullback. *Information theory and statistics.* Courier Corporation, 1997.

[24] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[25] Keith Y K Leung, Yoni Halpern, Timothy D Barfoot, and Hugh H T Liu. The UTIAS multi-robot cooperative localization and mapping dataset. 30(8):969–974, 2011.

[26] Yinan Li and Fang Liu. Whiteout: Gaussian Adaptive Noise Regularization in Deep Neural Networks. *arXiv:1612.01490*, 2016.

[27] Junfa Liu, Yiqiang Chen, and Yadong Zhang. Transfer Regression Model for Indoor 3D Location Estimation. In *International Conference on Multimedia Modeling*, volume 5916 LNCS, pages 603–613. Springer, 2010.

[28] Katherine Liu, Kyel Ok, William Vega-Brown, and Nicholas Roy. Deep Inference for Covariance Estimation: Learning Gaussian Noise Models for State Estimation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1436–1443. IEEE, 5 2018.

[29] Guoyu Lu, Yan Yan, Li Ren, Philip Saponaro, Nicu Sebe, and Chandra Kambhamettu. Where am I in the dark: Exploring active transfer learning on the use of indoor localization based on thermal imaging. *Neurocomputing*, 173, 1 2016.

[30] Songrit Maneewongvatana and David M. Mount. Analysis of approximate nearest neighbor searching with clustered point sets. 1 1999.

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

[32] Geoffrey McLachlan and David Peel. *Finite Mixture Models*. John Wiley & Sons, 2004.

[33] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve Restricted Boltzmann machines. In *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*, 2010.

[34] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H Wallach, H Larochelle, A Beygelzimer, F d Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[36] Valentin Peretroukhin, Lee Clement, Matthew Giamou, and Jonathan Kelly. PROBE: Predictive robust estimation for visual-inertial navigation. In *IEEE International Conference on Intelligent Robots and Systems*, volume 2015-Decem, 2015.

[37] Valentin Peretroukhin, Lee Clement, and Jonathan Kelly. Reducing drift in visual odometry by inferring sun direction using a Bayesian Convolutional Neural Network. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2017.

[38] Valentin Peretroukhin, Lee Clement, and Jonathan Kelly. Inferring sun direction to improve visual odometry: A deep learning approach. *International Journal of Robotics Research*, 37(9), 2018.

[39] Valentin Peretroukhin, William Vega-Brown, Nicholas Roy, and Jonathan Kelly. PROBE-GK: Predictive robust estimation using generalized kernels. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2016-June, 2016.

[40] Valentin Peretroukhin, Brandon Wagstaff, and Jonathan Kelly. Deep Probabilistic Regression of Elements of SO(3) using Quaternion Averaging and Uncertainty Injection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 6 2019.

[41] KB Petersen and MS Pedersen. The Matrix Cookbook, vol. 7. *Technical University of Denmark*, 15, 2008.

[42] Kristijan Petrovski, Stole Jovanovski, Miroslav Mirchev, and Lasko Basnarkov. On the Kalman filter approach for localization of mobile robots. *Advances in Intelligent Systems and Computing*, 665:123–133, 2018.

[43] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of Adam and beyond. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[44] Jonas Rothfuss, Fabio Ferreira, Simon Boehm, Simon Walther, Maxim Ulrich, Tamim Asfour, and Andreas Krause. Noise Regularization for Conditional Density Estimation. *arXiv:1907.08982*, 2019.

[45] Suwanchai Sangsuk-Iam and Thomas E Bullock. The discrete-time Kalman filter under uncertainty in noise covariances. In *Control and Dynamic Systems*, volume 76, pages 363–415. Elsevier, 1996.

[46] Alexander Schaefer, Daniel Buscher, Johan Vertens, Lukas Luft, and Wolfram Burgard. Long-term urban vehicle localization using pole landmarks extracted from 3-D lidar scans. In *2019 European Conference on Mobile Robots, ECMR 2019 - Proceedings*, 2019.

[47] Alexander Schaefer, Daniel Büscher, Johan Vertens, Lukas Luft, and Wolfram Burgard. Long-term vehicle localization in urban environments based on pole landmarks extracted from 3-D lidar scans. *Robotics and Autonomous Systems*, 136, 2021.

[48] Roland Siegwart, Illah R Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots, Second Edition*, volume 23. 2011.

[49] Abhijeet Tallavajhula, Barnabás Póczos, and Alonzo Kelly. Nonparametric distribution regression applied to sensor modeling. In *IEEE International Conference on Intelligent Robots and Systems*, volume 2016-Novem, pages 619–625, 2016.

[50] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279, 2018.

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

[52] Chris Turner, Hamed Sari-Sarraf, and Eric Hequet. Training a new instrument to measure cotton fiber maturity using transfer learning. *IEEE Transactions on Instrumentation and Measurement*, 66(7), 2017.

[53] Rudolph van der Merwe, Arnaud Doucet, Nando de Freitas, and Eric Wan. The Unscented Particle Filter. Technical report, Cambridge University Engineering Department, 8 2000.

[54] William Vega-Brown, Abraham Bachrach, Adam Bry, Jonathan Kelly, and Nicholas Roy. CELLO: A fast algorithm for Covariance Estimation. In *2013 IEEE International Conference on Robotics and Automation*, pages 3160–3167. IEEE, 5 2013.

[55] Zhuo Wei. *Fuzzy Logic and Neural Network-aided Extended Kalman Filter for Mobile Robot Localization*. PhD thesis, 2011.

[56] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[57] Jin Wenrui and Zhan Xingqun. A Modified Kalman Filtering via Fuzzy Logic System for ARVs Location. In *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation, ICMA 2007*, 2007.

[58] Troi Williams and Yu Sun. Learning State-Dependent, Sensor Measurement Models for Localization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3090–3097. IEEE, 11 2019.

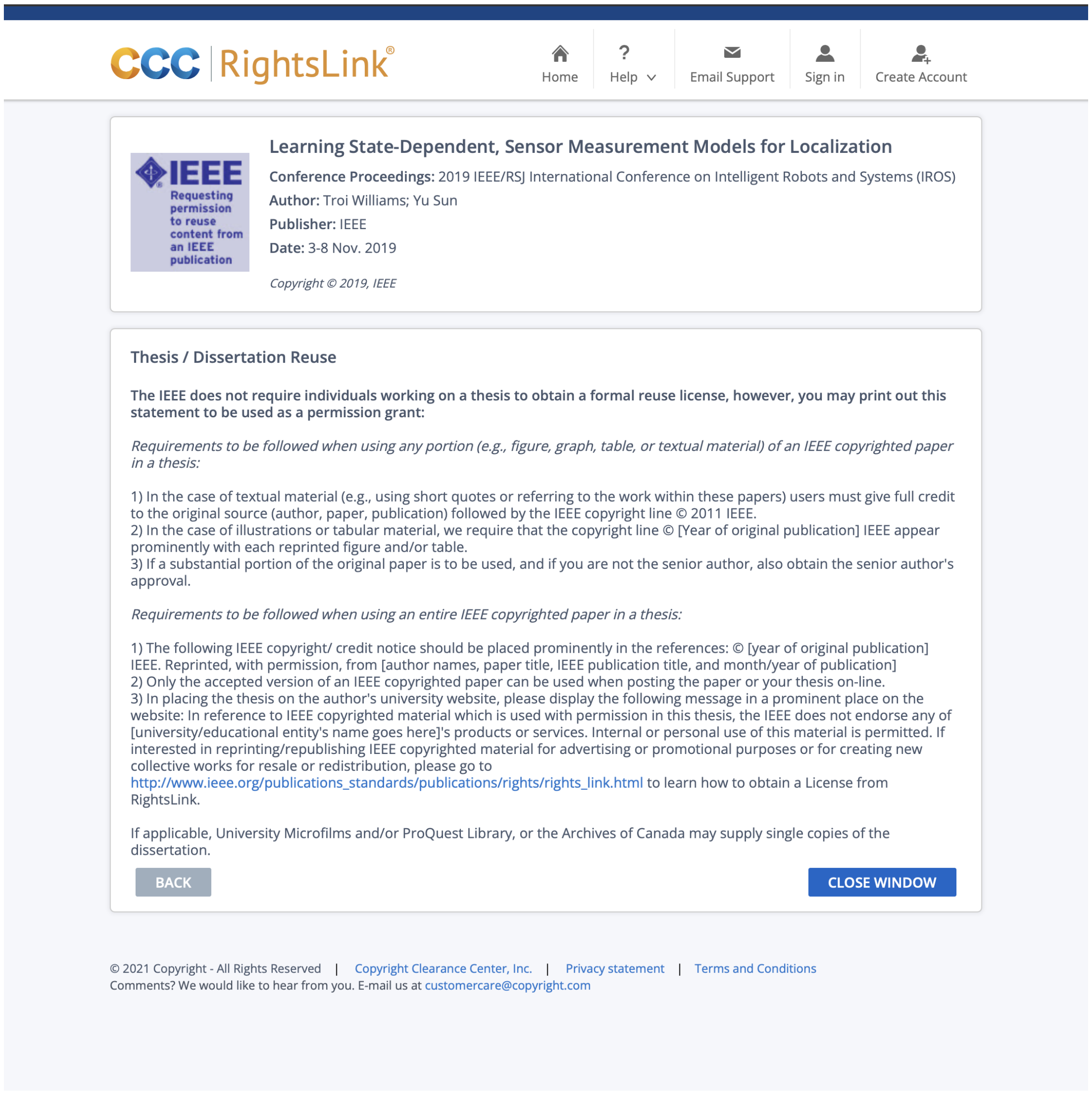
[59] Troi Williams and Yu Sun. Learning State-Dependent Measurement Likelihood Models with Limited Sensor Data. In *Robotics: Science and Systems (RSS) Pioneers workshop*, 2021.

[60] Troi Williams and Yu Sun. Learning State-Dependent Sensor Measurement Models with Limited Sensor Measurements. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, 2021. IEEE.

[61] Weiming Zhi, Ransalu Senanayake, Lionel Ott, and Fabio Ramos. Spatiotemporal Learning of Directional Uncertainty in Urban Environments with Kernel Recurrent Mixture Density Networks. *IEEE Robotics and Automation Letters*, 4(4), 2019.

Appendix A: Copyright Permissions

The permission below is for the reproduction of material in [58] within Chapters 2 and 4.


CCC RightsLink®
Home
Help
Email Support
Sign in
Create Account
IEEE Requesting permission to reuse content from an IEEE publication

Learning State-Dependent, Sensor Measurement Models for Localization

Conference Proceedings: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Author: Troi Williams; Yu Sun
Publisher: IEEE
Date: 3-8 Nov. 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

© 2021 Copyright - All Rights Reserved | Copyright Clearance Center, Inc. | Privacy statement | Terms and Conditions
Comments? We would like to hear from you. E-mail us at customercare@copyright.com

Appendix B: Back-propagation Equations for Mixture Density Networks

We derive the back-propagation equations for a Mixture Density Network (MDN) that predicts parameters for a Gaussian Mixture Model (GMM). We assume the GMM is a multi-modal, multivariate distribution with full covariance matrices. In the following subsections, we derive back-propagation equations for the mixing coefficient, the multivariate mean, and the full covariance matrix.

Before we begin the derivation, we briefly restate related equations for the reader's convenience. The GMM equation for an MDN as

$$p(\mathbf{z} \mid \Theta) = \sum_{k=1}^{m} p_k(\mathbf{z} \mid \Theta_k) = \sum_{k=1}^{m} \alpha_k(\boldsymbol{\lambda}) \cdot \phi_k(\mathbf{z} \mid \boldsymbol{\lambda}). \tag{B.1}$$

Here, $p(\mathbf{z} \mid \Theta)$ is the total probability of the distribution, and $p_k(\mathbf{z} \mid \Theta_k)$ is the weighted probability of the k-th component. m is the number of Gaussian components used to represent the total probability distribution. $\Theta = \{\Theta_1, \dots, \Theta_k, \dots, \Theta_m\}$ and Θ_k denote the distribution parameters for the entire model and the k-th component, respectively. α_k is the k-th mixture coefficient and $\sum_{k=1}^{m} \alpha_k = 1$. $\phi_k(\mathbf{z} \mid \Theta_k)$ is the k-th Gaussian probability density function (PDF) and is defined as

$$\phi_k(\mathbf{z} \mid \Theta_k) = \frac{1}{(2\pi)^{s/2} \cdot \left|\boldsymbol{\Sigma}_k\right|^{1/2}} \exp\left\{\left(\mathbf{z} - \boldsymbol{\mu}_k\right)^{\top} \cdot \left[\ \boldsymbol{\Sigma}_k\ \right]^{-1} \cdot \left(\mathbf{z} - \boldsymbol{\mu}_k\right)\right\}. \tag{B.2}$$

$\mathbf{z} \in \mathbb{R}^s$ is the target variable (such as a sensor measurement) and $\boldsymbol{\mu}_k \in \mathbb{R}^s$ is the conditional mean of the k-th Gaussian PDF. The variable s is the dimension of the target and Gaussian mean vectors. $\boldsymbol{\Sigma}_k \in \mathbb{R}^{s \times s}$ is the conditional, full covariance matrix for the k-th Gaussian PDF. $\Theta_k = \{\alpha_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{m}$ are the output parameters of a GMM layer of an MDN.

The notations below are used in the following sections. We first use the shorthand notation in the following equations. We use p, p_i, and $\mathcal{N}$ instead of $p(\mathbf{z} \mid \Theta)$, $p_k(\mathbf{z} \mid \Theta)$, and $\mathcal{N}_k(\mathbf{z} \mid \boldsymbol{\lambda})$, respectively. Second, like Bishop [4], we use the notation $\boldsymbol{\pi}$ to help simplify some equations:

$$\boldsymbol{\pi}(\boldsymbol{\lambda}, \mathbf{z}) = \frac{\alpha_k \cdot \phi_i}{\sum_{j=1}^{m} \alpha_j \cdot \phi_j}, \tag{B.3}$$

where $\sum_{i=1}^{m} \boldsymbol{\pi} = 1$. $\boldsymbol{\pi}$ represents a posterior probability, and Bishop [4] obtained this equation using Bayes theorem.

B.1 Back-propagation Equations for a Mixture Model Component

We derive the back-propagation equations for the k-th weight probability p_k:

$$\frac{\partial \mathcal{E}^i}{\partial p_k} = \frac{\partial \mathcal{E}^i}{\partial p} \cdot \frac{\partial p}{\partial p_k}, \tag{B.4}$$

where $\mathcal{E}^i$ is the error for the i-th example, and p and p_k are defined in (B.1). For this derivative, we used the chain rule to transform the left-hand side of the equation into the right-hand side. The first partial derivative in (B.4) becomes

$$\frac{\partial \mathcal{E}^i}{\partial p} = \frac{\partial}{\partial p}\Big\{ - \ln(p)\Big\} = -1 \cdot \frac{\partial}{\partial p}\Big\{ \ln(p)\Big\} = -\frac{1}{p} = -\frac{1}{\sum_{j=1}^{K} \alpha_j \cdot \phi_j}. \tag{B.5}$$

The second partial derivative in (B.4) is

$$\begin{aligned}\frac{\partial p}{\partial p_k} &= \frac{\partial}{\partial p_k}\Big\{p_1 + ... + p_k + ... + p_m\Big\} \\ &= \frac{\partial}{\partial p_k}\Big\{p_k + \sum_{j\neq k}^{m} p_j\Big\} \\ &= \frac{\partial}{\partial p_k}\Big\{p_k\Big\} + \frac{\partial}{\partial p_k}\Big\{\sum_{j\neq k}^{m} p_k\Big\} \\ &= 1.\end{aligned} \tag{B.6}$$

Combining (B.5) and (B.6), we compute (B.4) as

$$\frac{\partial \mathcal{E}^i}{\partial p_k} = \frac{\partial \mathcal{E}^i}{\partial p} \cdot \frac{\partial p}{\partial p_k} = -\frac{1}{\sum_{j=1}^{m} \alpha_j \cdot \phi_j}. \tag{B.7}$$

B.2 Back-propagation Equations for the Mixture Coefficients

We describe the partial derivative of the loss $\mathcal{E}^i$ with respect to γ_k^α:

$$\frac{\partial \mathcal{E}^i}{\partial \gamma_k^\alpha} = \sum_{c=1}^{m} \frac{\partial \mathcal{E}^i}{\partial p_c} \cdot \frac{\partial p_c}{\partial \alpha_c} \cdot \frac{\partial \alpha_c}{\partial \gamma_k^\alpha}. \tag{B.8}$$

Here, $\gamma_k^\alpha \in \mathbb{R}$ is one of m linear outputs from the network that is used to compute the mixture coefficient α_k. Since each α_k is a function of the outputs $\{\gamma_k^\alpha\}_{k=1}^m$ (via the softmax function), we compute this partial derivative as a summation (see [4] for details). $\partial\mathcal{E}^i/\partial p_c$ is (B.7) and $\partial p_c/\partial\alpha_c$ is

$$\frac{\partial p_c}{\partial \alpha_c} = \frac{\partial}{\partial \alpha_c}\Big\{\alpha_c \cdot \phi_c\Big\} = \phi_c \cdot \frac{\partial}{\partial \alpha_c}\Big\{\alpha_c\Big\} = \phi_c. \tag{B.9}$$

Combining (B.7), (B.9), and (B.3) we derive

$$\frac{\partial \mathcal{E}^i}{\partial \alpha_c} = \frac{\partial \mathcal{E}^i}{\partial p_c} \cdot \frac{\partial p_c}{\partial \alpha_c} = \frac{-1}{\sum_{j=1}^{m} \alpha_j \cdot \phi_j} \cdot \frac{\phi_c}{1} = \frac{-1}{\sum_{j=1}^{m} \alpha_j \cdot \phi_j} \cdot \frac{\alpha_c}{\alpha_c} \cdot \frac{\phi_c}{1} = -\frac{\boldsymbol{\pi}}{\alpha_c}. \tag{B.10}$$

Finally, we focus on the last partial derivative in (B.8). This partial derivative becomes

$$\frac{\partial \alpha_c}{\partial \gamma_k^{\alpha}} = \alpha - \alpha^2 \tag{B.11}$$

when $c = k$. However, when $c \neq k$, the partial becomes

$$\frac{\partial \alpha_c}{\partial \gamma_k^{\alpha}} = -\alpha_c \alpha_k. \tag{B.12}$$

Combining (B.8), (B.10), (B.11), and (B.12), we get

$$\frac{\partial \mathcal{E}^i}{\partial \gamma_k^{\alpha}} = \alpha_k - \boldsymbol{\pi}. \tag{B.13}$$

B.3 Back-propagation Equations for the Mean

We begin the derivative of the loss function $\mathcal{E}^i$ with respect to $\gamma_{k,a}^{\mu}$, the linear outputs for the location $\mu_{k,a}$. Here, the index a is the a-th dimension in the multivariate location $\boldsymbol{\mu}_k$. We remind the reader that $\mu_{k,a} = \gamma_{k,a}^{\mu}$. Therefore, $\partial \mathcal{E}^i / \partial \gamma_{k,a}^{\mu}$ is

$$\frac{\partial \mathcal{E}^i}{\partial \gamma_{k,a}^{\mu}} = \frac{\partial \mathcal{E}^i}{\partial p_k} \cdot \frac{\partial p_k}{\partial \phi_k} \cdot \frac{\partial \phi_k}{\partial u_k} \cdot \frac{\partial u_k}{\partial \gamma_{k,a}^{\mu}}, \tag{B.14}$$

where $\mathcal{N}_k$ is (B.2) and

$$u_k = -\frac{1}{2} \cdot (\mathbf{z} - \boldsymbol{\mu}_k)^{\top} \boldsymbol{\Sigma}_k^{-1} (\mathbf{z} - \boldsymbol{\mu}_k). \tag{B.15}$$

The first partial derivative in the chain rule is (B.7). The second partial derivative becomes

$$\frac{\partial p_k}{\partial \phi_k} = \frac{\partial}{\partial \phi_k} \Big\{ \alpha_k \cdot \phi_k \Big\} = \alpha_k \cdot \frac{\partial}{\partial \phi_k} \Big\{ \phi_k \Big\} = \alpha_k. \tag{B.16}$$

We compute the third derivative as

$$\frac{\partial \phi_k}{\partial u_k} = \frac{\partial}{\partial u_k} \Big\{ S \cdot \exp\{u_k\} \Big\} = S \cdot \tfrac{\partial}{\partial u_k} \exp\{u_k\} = S \cdot \exp\{u_k\} = \phi_k.$$

Finally, we use (86) from [41] to compute the last partial derivative in (B.14):

$$\begin{aligned}\frac{\partial u_k}{\partial \gamma^{\mu}_{k,a}} &= -\frac{1}{2} \cdot \frac{\partial}{\partial \mu_{k,a}} \Big\{ (\mathbf{z} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{z} - \boldsymbol{\mu}_k) \Big\} \\ &= \frac{1}{-2} \cdot \frac{-2}{1} \cdot \boldsymbol{\Sigma}_{k,a}^{-1} (\mathbf{z} - \boldsymbol{\mu}_k) \\ &= \boldsymbol{\Sigma}_{k,a}^{-1} (\mathbf{z} - \boldsymbol{\mu}_k),\end{aligned} \tag{B.17}$$

where $\boldsymbol{\Sigma}_{k,a}$ is a row vector that corresponds to the a-th row of the covariance matrix $\boldsymbol{\Sigma}_k$. Putting together (B.7), (B.16), (B.17), and (B.17), we get

$$\begin{aligned}\frac{\partial \mathcal{E}^i}{\partial \gamma^{\mu}_{k,a}} &= \frac{\partial \mathcal{E}^i}{\partial p_k} \cdot \frac{\partial p_k}{\partial \phi_k} \cdot \frac{\partial \phi_k}{\partial u_k} \cdot \frac{\partial u_k}{\partial \gamma^{\mu}_{k,a}} \\ &= \frac{-1}{\sum_{j=1}^{m} \alpha_j \cdot \phi_j} \cdot \frac{\alpha_k}{1} \cdot \frac{\phi_k}{1} \cdot \frac{\boldsymbol{\Sigma}_{k,a}^{-1} (\mathbf{z} - \boldsymbol{\mu}_k)}{1} \\ &= \boldsymbol{\pi} \cdot \boldsymbol{\Sigma}_{k,a}^{-1} (\boldsymbol{\mu}_k - \mathbf{z}).\end{aligned} \tag{B.18}$$

B.4 Back-propagation Equations for the Full Covariance Matrix

Our final derivation focuses on the free parameters $\gamma^{d}_{k,a}$ and $\gamma^{l}_{k,a,b}$ for the matrices $\mathbf{D}_k$ and $\mathbf{L}_k$ (respectively), where $\boldsymbol{\Sigma}_k = \mathbf{L}_k \mathbf{D}_k \mathbf{L}_k^\top$. Particularly, we derive

$$\frac{\partial \mathcal{E}^i}{\partial \gamma^{d}_{k,a}} = \frac{\partial \mathcal{E}^i}{\partial \boldsymbol{\Sigma}_k} \cdot \frac{\partial \boldsymbol{\Sigma}_k}{\partial D_{k,a,a}} \cdot \frac{\partial D_{k,a,a}}{\partial \gamma^{d}_{k,a}}, \tag{B.19}$$

$$\frac{\partial \mathcal{E}^i}{\partial \gamma^{l}_{k,a,b}} = \frac{\partial \mathcal{E}^i}{\partial \boldsymbol{\Sigma}_k} \cdot \frac{\partial \boldsymbol{\Sigma}_k}{\partial \gamma^{l}_{k,a,b}}, \tag{B.20}$$

where each $\gamma^{d}_{k,a}$ maps to $D_{k,a,a}$, a diagonal position in $\mathbf{D}_k$, and each $\gamma^{l}_{k,a,b}$ maps to $L_{k,a,b}$, a lower triangle index in $\mathbf{L}_k$. Leveraging (396) in [41], the first partial derivative in (B.19) and (B.20) equates to

$$\frac{\partial \mathcal{E}^i}{\partial \boldsymbol{\Sigma}_k} = \frac{\boldsymbol{\pi}}{2} \Big(\boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1} \mathbf{v}_k \mathbf{v}_k^\top \boldsymbol{\Sigma}_k^{-1} \Big), \tag{B.21}$$

where $\mathbf{v}_k = \mathbf{z} - \boldsymbol{\mu}_k$. Given $\boldsymbol{\Sigma}_k = \mathbf{L}_k\mathbf{D}_k\mathbf{L}_k^\top$, the second partial derivative in (B.19) becomes

$$\frac{\partial \boldsymbol{\Sigma}_k}{\partial D_{k,a,a}} = \frac{\partial}{\partial D_{k,a,a}}\mathbf{L}_k\mathbf{D}_k\mathbf{L}_k^\top = \mathbf{L}_k\mathbf{1}_{k,a,a}\mathbf{L}_k^\top, \tag{B.22}$$

where $\mathbf{1}_{k,a,a}$ denotes an $s \times s$ matrix with a value of 1 at index (a, a) and zeros everywhere else (see [16] for example). The last derivative in (B.19) becomes

$$\frac{\partial D_{k,a,a}}{\partial \gamma^d_{k,a}} = \frac{\partial}{\partial \gamma^d_{k,a}} \exp\left(\gamma^d_{k,a}\right) = D_{k,a,a}. \tag{B.23}$$

Combining (B.21), (B.22), and (B.23) for (B.19), we get

$$\frac{\partial \mathcal{E}^i}{\partial \gamma^d_{k,a}} = \frac{\boldsymbol{\pi}}{2}\left(\boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1}\mathbf{v}_k\mathbf{v}_k^\top\boldsymbol{\Sigma}_k^{-1}\right)\mathbf{L}_k\mathbf{1}_{k,a,a}\mathbf{L}_k^\top D_{k,a,a}. \tag{B.24}$$

We now focus on the last partial derivative in (B.20). The product rule is applied to the partial derivative to produce the result

$$\frac{\partial \boldsymbol{\Sigma}_k}{\partial \gamma^l_{k,a,b}} = \mathbf{1}_{k,a,b}\mathbf{D}_k\mathbf{L}_k^\top + \mathbf{L}_k\mathbf{D}_k\mathbf{1}_{k,b,a}. \tag{B.25}$$

We draw the reader's attention to the matrices $\mathbf{1}_{k,a,b}$ in the first term and $\mathbf{1}_{k,b,a}$ in the second term, where indices a and b are swapped due to transposing $\mathbf{L}_k$. Combining (B.20), (B.21) and (B.25), we get

$$\frac{\partial \mathcal{E}^i}{\partial \gamma^l_{k,a,b}} = \frac{\boldsymbol{\pi}}{2}\left(\boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1}\mathbf{v}_k\mathbf{v}_k^\top\boldsymbol{\Sigma}_k^{-1}\right)\left(\mathbf{1}_{k,a,b}\mathbf{D}_k\mathbf{L}_k^\top + \mathbf{L}_k\mathbf{D}_k\mathbf{1}_{k,b,a}\right). \tag{B.26}$$

About the Author

Troi Williams is a native of the United States (U.S.) Virgin Islands. He earned a Bachelor of Science degree in Computer Science from the University of the Virgin Islands, U.S. Virgin Islands, USA in 2011. Then Troi earned a Master of Science degree in Computer Science from Norfolk State University, Virginia, USA in 2014. Finally, Troi earned a Ph.D. at the University of South Florida within the Department of Computer Science and Engineering. During his Ph.D., he was advised by Dr. Yu Sun, a professor of Computer Science and director of the Robot Perception and Action Lab.

His research interests include sensor modeling, robot perception, SLAM, and applications in autonomous vehicles.