ENVIRONMENTAL ESTIMATION AND SMOOTHING ALGORITHMS FOR DYNAMIC AND MODULAR ROBOTS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Daniel J. Lee

August 2016

ENVIRONMENTAL ESTIMATION AND SMOOTHING ALGORITHMS FOR DYNAMIC AND MODULAR ROBOTS

Daniel J. Lee, Ph.D.

Cornell University 2016

A set of estimation algorithm is presented for various robotics platforms. Among many different types of platforms, this work focuses on two types; large, dynamically complex systems with rich set of sensors and high computation power, and self-reconfigurable modular robots with limited computational resources and sensing capability. Three distinct types of estimation algorithm are presented; 1) adaptive Gaussian mixture smoother for highly complex nonlinear dynamics and measurement models with multi-modal noises, 2) a scalable and efficient surface normal estimation algorithms for computationally limited platforms, and 3) a set of estimation algorithms to calibrate a set of modular robots in order to estimate position and orientation of sensor module. Each set of algorithms are validated with appropriate numerical studies and experiments.

BIOGRAPHICAL SKETCH

Daniel Lee received his B.S. degree in Mechanical engineering from Cornell University, Ithaca, NY, USA in 2009. He received his M.S. degree from Cornell University in Mechanical engineering in 2014. His primary research focuses on estimation methods for state estimation, scene perception, sensor fusion, and modular robots.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Prof. Mark Campbell, for continuous support and great leadership for past eight years of my time at Cornell. It has been pleasure to learn and work in his lab and finally finishing my degree at Cornell. These experiences have been priceless and will lead my future career. I would like to thank my committee members, Prof. Hadas Kress-Gazit, and Prof. Noah Snavely, for the guidance on my thesis, and all the advice and critical questions to raise the quality of my work.

I would like to thank Prof. Nisar Ahmed, who was a Ph.D. student then as I joined the group, for all the brilliant and creative ideas, technical advice and discussions regarding research topics.

I would like to thank my fellow graduate students Kevin Wyffels and Rina Tse for sharing the graduate student time in Cornell and encouraging and helping each other to finally finish the degrees. I thank all fellow Ph.D. students in Autonomous Systems Lab for lengthy discussion on various research topics and supporting one another.

I would like to thank Korean Graduate Tennis Club at Cornell whose members have helped and supported me for past three years for all those fun tennis times and hangouts – of course, with lots of beer and wines; especially Jongrim Ha, Lawrence Jin, Jaesun Lee, and Younghwa Seok.

Lastly, I would never have been able to write this page without unconditional support and love from my family.

Funding agencies: NSF Graduate Research Fellowship Program

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Throughout the history of robotics, a large number of robotics platform have been developed. With rapid development and improvement of hardware, many platforms are equipped with high computation power, rich set of sensors, high accuracy manipulation devices, and more. These different variants of robotics platforms, in these days, consist of small robotics toys for children, flying and underwater vehicles, personal robots, modular robots, research mobile platforms, and more.

Different types of robotics platform exhibit different challenges to solve. In this work, two types of robotics platforms are considered; one is typically large in size, has rich set of highly accurate sensors, and carries out complex interaction with environment or humans. Such examples consist of autonomous driving vehicles, humanoid and home assisting robots, and large scale research platforms. While this type of platforms excels in sensing and interaction capabilities, it often suffers from having complex dynamics and measurement models. Given the complexity of the hardware, the dynamics and measurement models are often highly nonlinear and difficult to be modeled using single Gaussian distribution.

The other type of the platform to focus is modular robots. They are typically small in size, and have limited computational power and sensing capability. However, they are often self-reconfigurable which attracts much attention recently. Their ability to self-reconfigure, however, requires making connections with other modules. This imposes several challenges to consider: connection offsets between modules, encoder biases and nonlinear mapping of the encoder

values, bending and torsion due to material compliance, and external forces.

In this work, a few estimation methods are presented. Chapter 2 presents an estimation method for platforms that have highly nonlinear dynamics and measurement models, while the noises are multi-modal. Utilizing adaptive Gaussian mixture model, a smoothing estimation method is presented which significantly improves the estimates of highly nonlinear and complex model. Chapter 3 presents a novel surface normal estimation method for small platforms with limited computational resources. This surface normal estimation method serves as a base for high-level object recognition, classification, and segmentation work, and is suited for modular robots for its computational efficiency. Chapter 4 presents estimation methods for estimating sensor positions for various configurations using modular robots. This chapter presents solutions for each challenge that were mentioned before.

CHAPTER 2

A SMOOTHING ALGORITHM FOR NONLINEAR SYSTEMS USING GAUSSIAN MIXTURE MODELS

Nomenclature

x	State vector
z	Measurement vector
f	Propagation function
g	Measurement function
Q	Process noise covariance matrix
R	Measurement noise covariance matrix
N_t	The number of mixands of a Gaussian mixture at time t
$\mathcal{N}(x; m, P)$	Normal distribution of x with mean vector m and covariance matrix P
ω	Weight of a mixand of a Gaussian mixture
B	Backward corrector (discussed in Section 2.3)
D	Backward corrector's noise covariance matrix
C	Backward corrector's stacked Jacobian matrix parameter required to evaluate D
$\zeta(x)$	Predicted measurement vector given the state variable x
F	Jacobian matrix of nonlinear propagation function f
H	Jacobian matrix of nonlinear measurement function h
K	Gain matrix to compute the smoothed mean vector and covariance matrix
q	Multiplication factor to compute the smoothed weight

$\hat{\zeta}$ Least squares estimate of the measurement vector

$\hat{D}$ Least squares estimate of the backward corrector's noise covariance matrix

$\hat{C}$ Least squares estimate of the backward corrector's Jacobian matrix required to evaluate $\hat{D}$

θ Association variable of the mixands of a Gaussian mixture

Subscript

t	Time of the estimation
T	The last time index of the smoothing window
$t_1 : t_2$	Time interval from t_1 to t_2 inclusive

Superscript

i	Indicates i-th mixand of a Gaussian mixture
*	Indicates smoothed variable
T	Transpose of the vector/matrix variable

Symbols

$\vert_{(\cdot)}$	Evaluation/linearization point $(\cdot)$

2.1 Introduction

Estimation theory has been rigorously developed to optimally/sub-optimally produce estimates of systems and processes in real time, even in the presence of disturbances and noises from various sources. Many optimal filtering algorithms exist to solve certain classes of problems. For example, the Kalman filter [3], and its many variants are used extensively in the community for linear systems [72]. Extensions to the linear Kalman filter have been developed to handle nonlinear systems through linearization and the sigma-point transfor-

mation [6, 9, 77, 31]. Such filters typically operate on the first two moments, and have successfully solved many problems where accurate and precise estimates are required. However, classes of more complex problems which cannot be accurately solved with these algorithms exist, including applications with highly nonlinear dynamics, and non-Gaussian or multi-modal disturbances. Such examples include camera vision measurements, GPS signal drop-out and multi-path problems, wheel slips, and more. Many of these problems become more pronounced in sparse measurement conditions where there is a higher reliance on models.

To address highly nonlinear systems and/or non-Gaussian noise models, filters using sampling methods [16] or Gaussian mixture models have been developed [1]. Sampling methods, such as particle filters, consider both highly nonlinear dynamics and non-Gaussian noise models. While good performance has been shown in a variety of applications [47, 48], sampling methods are computationally demanding and rely on accurate system and likelihood models [4]. Gaussian mixture on the other hand needs much lower computation; for example, Gaussian mixture using sigma-points use $(2N+1)*M$ points (for M mixands and N-dimensional state) which is typically much lower than the number required by a particle filter. In the baseline Gaussian mixture filter [1], the number of mixands of the underlying Gaussian mixture model grows only with multi-modal/non-Gaussian noises, not nonlinearities. Researchers have tackled this limitation using splitting and condensation techniques [18, 17, 19, 25, 26]. This adaptive splitting enables the filter to maintain a reasonable number of mixands while capturing the non-Gaussian distribution effectively.

In addition to filtering (estimating current states based on the measurement

data and models), smoothing techniques have been developed which enable multiple passes of the measurement data. Smoothing algorithms utilize the measurements available in future time (relative to the time of interest) to improve the past estimates by reasoning about the system and measurement models. Thus, for challenging estimation problems (such as with sparse measurements) where the real-time estimate is inconsistent and/or inaccurate, smoothing algorithms provide an opportunity to greatly improve the estimates of interest using information over the past history as well as the future. Smoothing techniques have been used in the field for a wide range of applications including navigation of a land vehicle [81], underwater vehicle [79]; target tracking [64, 51]; and image restorations [5].

The literature includes smoothing techniques based on the Kalman filter family for linear and nonlinear systems [3, 60, 57, 58], and numerical solutions implementing the forward-backward algorithm are available including sampling techniques [16]. Kitagawa presented a two-filter implementation of a smoother based on Gaussian mixture models [34], but with a limitation on the number of measurements required to invert the measurement model. Recently, Vo *et al*. presented a closed-form solution for forward-backward algorithm implementation of Gaussian mixture models [76]; this approach was limited to linear systems, which enables the simplification of the smoothing steps.

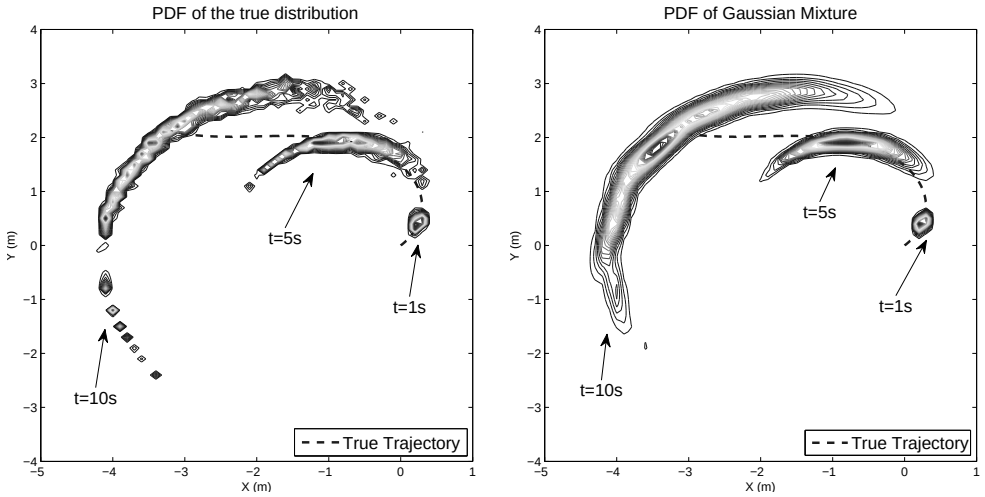
This paper presents a novel smoothing method using a Gaussian mixture representation for nonlinear systems. This method is particularly well suited for applications with highly nonlinear dynamics and/or measurement models, as well as for applications with sparse measurements where the effect of model nonlinearities becomes more pronounced. When measurements are not avail-

able, filtering algorithms only temporarily predict the evolution of the state distribution using models; this prediction can be greatly affected by the inherent nonlinearity of the system. When a measurement becomes available at a later time, however, utilizing the proposed smoothing algorithm enables these hard-to-observe errors in filtering process to be improved. A closed-form estimate of the backward corrector is derived which enables a computationally scalable implementation.

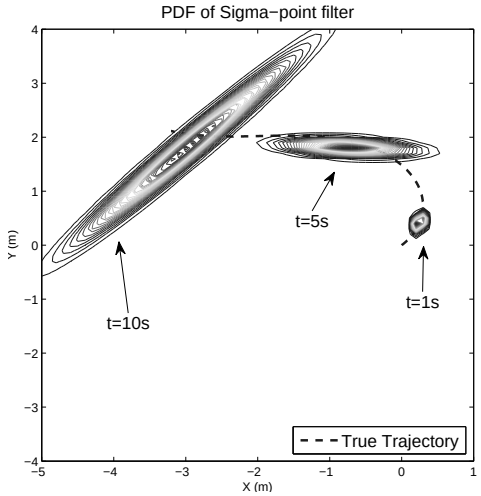
This paper is outlined as follows: Section 2.2 presents an overview of the problem, and challenges of Gaussian mixture smoothing. Section 2.3 presents a closed-form solution for a forward-backward algorithm with a Gaussian mixture prior, where the propagation and measurement models are nonlinear and the noises are modeled using single Gaussian distributions. Section 2.4 presents an adaptive Gaussian mixture model smoother for nonlinear systems with Gaussian mixture model priors. Section 2.5 presents simulation results for a highly nonlinear system to gain insight into its implementation, and experimental results for an indoor robot navigation experiment in presence of sparse measurements to demonstrate scalability and robustness. Results are compared to baseline algorithms to demonstrate the accuracy and the consistency of the proposed Gaussian mixture smoothers.

2.2 Overview of the Gaussian Mixture Smoothing Problem

As a motivating example, consider Fig. 2.1, which shows the simulation result of robot motion using a unicycle model with a small bias error in the wheel radius. The initial state is estimated as a Gaussian distribution with a mean



(a) True distribution (b) Gaussian mixture model



(c) Sigma-points

Figure 2.1: Contour plot of robot's location estimate pdfs, $p(x, y)$, of three prediction time steps shown for three estimators: (a) 30,000 particles, (b) Gaussian mixture with 10 mixands, (c) sigma-points. Dotted line shows the true trajectory robot has taken.

vector, $[x_0, y_0, \theta_0, R_0]^\mathsf{T}$, and associated covariance matrix; x, y, θ, and R represent the two-dimensional location, heading of the robot, and the radius of the wheel, respectively. There is no measurement in the simulation, so that Fig. 2.1 only shows temporal prediction of an initial Gaussian state distribution through non-linear dynamics. The marginal probability density functions (pdf) of robot's

two dimensional location estimates, $p(x, y)$, are shown using contour plots for three time steps: $t \in \{1, 5, 10\}$ seconds, along with the true trajectory taken. Figure 2.1(a) shows the true pdf defined using 30,000 particles; to calculate the marginal distribution, $p(x, y)$, each particle is treated as a Gaussian distribution with an infinitesimal covariance. Figures 2.1(b) and 2.1(c) are generated using a Gaussian mixture with maximum of 10 mixands [1], and sigma-points [77], respectively. Each simulation is initialized with the true state as the estimate, but with a small wheel radius bias.

The particles are initially sampled ($t = 0$) from a Gaussian distribution, and then evolve into larger crescent shape distributions over subsequent time steps. This is a result of the propagation of the small constant wheel radius error through the nonlinear system dynamics. In the case of the sigma-point predictor (Fig. 2.1(c)), the mean estimate is accurately captured (compared to the average of the particles), but the shape of the true distribution is not, because the uncertainty/error prediction is not well modeled with a single Gaussian distribution. On the other hand, however, a Gaussian mixture (Fig. 2.1(b)) captures the non-Gaussian shape of the true distribution well with ten mixands.

Figure 2.1 only considers the case of 'prediction'; consider now the problem of a Gaussian mixture smoother. A smoothing algorithm produces a *smoothed estimate*, $p(x_t|z_{1:T})$, utilizing measurements at past and future times; $T > t$, and x and z represent the state and the measurement, respectively. Because more measurements are used to produce the smoothed estimate, $p(x_t|z_{1:T})$, than to produce the posterior estimate, $p(x_t|z_{1:t})$, the smoothed estimates are expected to be more accurate; generally a smoother performs better than a filter, albeit with some delay of estimates and increased complexity and computation [3].

The forward-backward smoothing algorithm is described with following equation [8, 16],

$$p(x_t|z_{1:T}) = p(x_t|z_{1:t}) \int \frac{p(x_{t+1}|z_{1:T})}{p(x_{t+1}|z_{1:t})} \, p(x_{t+1}|x_t) \, dx_{t+1}, \tag{2.1}$$

and smoothing occurs backward in time from the end time T to the current time t. Consider an underlying Gaussian mixture model representation for the predicted state for time $t + 1$ given measurements up to time t as:

$$p(x_{t+1}|z_{1:t}) = \sum_{i=1}^{N_{t+1}} \omega_{t+1}^i \cdot \mathcal{N}(x_{t+1}; \bar{m}_{t+1}^i, \bar{P}_{t+1}^i) \tag{2.2}$$

where $\bar{m}_{t+1}^i$ and $\bar{P}_{t+1}^i$ represent the predicted mean vector and covariance matrix, respectively, for time $t + 1$. Equation (2.1) contains a division by a Gaussian mixture (Eq. (2.2)), which does not necessarily give a Gaussian mixture [34, 76]. Vo *et al*. developed a recursive closed-form solution to bypass this division for linear state propagation and measurement functions. The approach scales computationally with smoothing window $T - t$, as measurement vectors and covariance matrices are stacked over time [76].

Two smoothing methods for Gaussian mixture models with nonlinear system dynamics and measurement models are developed here. The first uses the forward-backward algorithm to find a closed-form solution (Section 2.3); this is a linearized version of that proposed by Vo *et al* [76]. In addition, a novel least squares fit to the integral of the Eq. (2.1) is proposed which reduces scaling from $O(T)$ to $O(1)$. A second approach is developed which adaptively splits Gaussian mixtures during the propagation step, and condenses the mixtures during the smoothing step (Section 2.4). This method, termed the adaptive Gaussian mixture model smoother, is designed to more accurately capture the propagation of the Gaussian mixture model through nonlinear dynamics and/or sparse measurements.

2.3 Nonlinear Forward-backward Smoothing with a Gaussian Mixture Prior

Consider the following system, with nonlinear propagation and measurement functions and additive Gaussian noise,

$$p(x_t|x_{t-1}) = \mathcal{N}(x_t; f(x_{t-1}), Q_t) \tag{2.3}$$

$$p(z_t|x_t) = \mathcal{N}(z_t; h(x_t), R_t) \tag{2.4}$$

where the propagation and measurement functions are described as $x_{t+1} = f(x_t) + w_t$ and $z_t = h(x_t) + v_t$. These functions have additive white noise, with noise covariance matrices $Q_t = E[w_t w_t^{\mathsf{T}}]$ and $R_t = E[v_t v_t^{\mathsf{T}}]$. The general smoothing algorithm given in Eq. (2.1) includes the 'forward' filtering step, composed of both propagation and measurement update,

$$p(x_t|z_{1:t}) = \frac{p(z_t|x_t)p(x_t|z_{1:t-1})}{\int p(z_t|x_t)p(x_t|z_{1:t-1})dx_t}. \tag{2.5}$$

The forward filtering step can be factored in the following manner:

$$p(x_t|z_{1:t}) = L_t(z_t; x_t)p(x_t|z_{1:t-1}) \tag{2.6}$$

where

$$L_t(z_t; x_t) = \frac{p(z_t|x_t)}{\int p(z_t|x_t)p(x_t|z_{1:t-1})dx_t}. \tag{2.7}$$

The 'backward' smoothing step is then defined using the backward corrector, B_t,

$$B_t(x_t|z_{1:T}) = \int \frac{p(x_{t+1}|z_{1:T})}{p(x_{t+1}|z_{1:t})} p(x_{t+1}|x_t)dx_{t+1}. \tag{2.8}$$

Then, the forward-backward smoothing method in Eq. (2.1) from time T to t can be factored as:

$$\begin{aligned} p(x_t|z_{1:T}) &= p(x_t|z_{1:t})B_t(x_t|z_{1:T}) \\ &= p(x_t|z_{1:t-1})L_t(z_t; x_t)B_t(x_t|z_{1:T}). \end{aligned} \tag{2.9}$$

Note that for end time $t = T$, $p(x_T|z_{1:T}) = p(x_T|z_{1:T}) \cdot B_T(x_T|z_{1:T})$; thus $B_T(x_T|z_{1:T}) = 1$. Importantly, for nonlinear propagation and measurement models with additive Gaussian noise as defined in Eqs. (2.3) and (2.4), both L_t and B_t (in Eqs. (2.7) and (2.8), respectively) can be shown to be an (unnormalized) Gaussian also [76].

The backward corrector for next time step, B_{t-1}, can be computed recursively using B_t [76],

$$B_{t-1}(x_{t-1}|z_{1:T}) = \int B_t(x_t|z_{1:T})L_t(z_t; x_t)p(x_t|x_{t-1})dx_t. \tag{2.10}$$

The key requirement for a scalable and accurate Gaussian mixture model smoother is a closed-form expression for the backward corrector, B_t, in Eq. (2.10). This is developed in the next section.

2.3.1 Calculation of a Linearized Backward Corrector

The backward corrector always takes the form of a Gaussian distribution, given that $B_T = 1$ and the propagation and measurement models assume additive Gaussian noise (Eqs. (2.3) and (2.4)); note that $L_t(z_t; x_t)$ and $p(x_t|x_{t-1})$ are both Gaussian in Eq. (2.10). Therefore, the backward corrector at general time t (propagating back from end time T) can be written as:

$$B_t(x_t|z_{1:T}) = \frac{\mathcal{N}(z_{t+1:T}; \zeta_{t+1:T}(x_t), D_{t+1:T}(x_t))}{r_t} \tag{2.11}$$

where r_t is a normalization constant, $z_{t+1:T}$ is a stacked measurement vector growing with available measurements as smoothing time moves backward from time T, $\zeta_{t+1:T}(x_t)$ represents predicted measurements *from* the state x_t for the specified time span as described below:

$$z_{t+1:T} = \left[z_T^\mathsf{T}, z_{T-1}^\mathsf{T}, \cdots z_{t+1}^\mathsf{T}\right]^\mathsf{T} \tag{2.12}$$

$$\zeta_{t+1:T}(x_t) \triangleq \left[h(f^{T-t-1}(x_t))^\mathsf{T},\ h(f^{T-t-2}(x_t))^\mathsf{T},\ \cdots,\ h(f(x_t))^\mathsf{T}\right]^\mathsf{T} \tag{2.13}$$

where

$$f^N(x_t) \triangleq \underbrace{f(f(f(\cdots f(x_t))\cdots)}_{N}. \tag{2.14}$$

Equation (2.14) defines a superscript notation indicating propagation of a point x_t through a function f a total of N times.

The calculation of matrix $D_{t+1:T}$ in Eq. (2.11) involves stacking the history of the Jacobian matrices for time interval $[t+1,\ T]$, where each subset of the matrix represents the predicted measurement noise covariance for a specific time in the interval. Because constructing this matrix involves a complex series of matrix calculations, a few examples starting from time T are shown here to more fully understand the final closed-form recursive solution. The initial $D_{T:T}(x_{T-1})$ is calculated as:

$$\begin{aligned} z_{T:T} &= z_T \\ \zeta_{T:T}(x_{T-1}) &= h(f(x_{T-1})) \\ D_{T:T}(x_{T-1}) &= R_T + H_T|_{x_{T-1}} Q_T H_T^\mathsf{T}|_{x_{T-1}} \end{aligned}$$

where $H_T|_{x_{T-1}}$ is a Jacobian matrix of the measurement function $h(x_T)$ in Eq. (2.4), which can be computed analytically [3] and numerically [31]. Note that because $H_T|_{x_{T-1}}$ is the Jacobian matrix for $h(x_T)$ at time T, the linearization point is $x_T = f(x_{T-1})$.

The parameters of the backward corrector B_{T-2} for time step $T-2$ are:

$$
\begin{aligned}
z_{T-1:T} &= \begin{bmatrix} z_T^\mathsf{T}, & z_{T-1}^\mathsf{T} \end{bmatrix}^\mathsf{T} \\
\zeta_{T-1:T}(x_{T-2}) &= \begin{bmatrix} h(f(f(x_{T-2})))^\mathsf{T}, & h(f(x_{T-2}))^\mathsf{T} \end{bmatrix}^\mathsf{T} \\
D_{T-1:T}(x_{T-2}) &= \begin{bmatrix} R_T + H_T|_{x_{T-2}} Q_T H_T^\mathsf{T}|_{x_{T-2}} & 0 \\ 0 & R_{T-1} \end{bmatrix} + \begin{bmatrix} H_T|_{x_{T-2}} F_{T-1:T}|_{x_{T-2}} \\ H_{T-1}|_{x_{T-2}} \end{bmatrix} Q_{T-1} \begin{bmatrix} H_T|_{x_{T-2}} F_{T-1:T}|_{x_{T-2}} \\ H_{T-1}|_{x_{T-2}} \end{bmatrix}^\mathsf{T}
\end{aligned}
$$

where $F_{T-1:T}|_{x_{T-2}}$ represents a Jacobian matrix of the propagation function $f(x_{T-1})$ for time interval $[T-1, T]$ computed about the point x_{T-2}. Similarly, for time step $T-3$, the parameters of B_{T-3} are:

$$
\begin{aligned}
z_{T-2:T} &= \begin{bmatrix} z_T^\mathsf{T}, & z_{T-1}^\mathsf{T}, & z_{T-2}^\mathsf{T} \end{bmatrix}^\mathsf{T} \\
\zeta_{T-2:T}(x_{T-3}) &= \begin{bmatrix} h(f(f(f(x_{T-3})))^\mathsf{T}, & h(f(f(x_{T-3})))^\mathsf{T}, & h(f(x_{T-3}))^\mathsf{T} \end{bmatrix}^\mathsf{T}
\end{aligned}
$$

$$
\begin{aligned}
&D_{T-2:T}(x_{T-3}) = \\
&\begin{bmatrix} \begin{bmatrix} R_T + H_T|_{x_{T-3}} Q_T H_T^\mathsf{T}|_{x_{T-3}} & 0 \\ 0 & R_{T-1} \end{bmatrix} + \begin{bmatrix} H_T|_{x_{T-3}} F_{T-1:T}|_{x_{T-3}} \\ H_{T-1}|_{x_{T-3}} \end{bmatrix} Q_{T-1} \begin{bmatrix} H_T|_{x_{T-3}} F_{T-1:T}|_{x_{T-3}} \\ H_{T-1}|_{x_{T-3}} \end{bmatrix}^\mathsf{T} & 0 \\ 0 & R_{T-2} \end{bmatrix} \\
&\quad + \begin{bmatrix} H_T|_{x_{T-3}} F_{T-1:T}|_{x_{T-3}} F_{T-2:T-1}|_{x_{T-3}} \\ H_{T-1}|_{x_{T-3}} F_{T-2:T-1}|_{x_{T-3}} \\ H_{T-2}|_{x_{T-3}} \end{bmatrix} Q_{T-2} \begin{bmatrix} H_T|_{x_{T-3}} F_{T-1:T}|_{x_{T-3}} F_{T-2:T-1}|_{x_{T-3}} \\ H_{T-1}|_{x_{T-3}} F_{T-2:T-1}|_{x_{T-3}} \\ H_{T-2}|_{x_{T-3}} \end{bmatrix}^\mathsf{T}.
\end{aligned}
$$

As shown, the dimension of $D_{t+1:T}(x_t)$ matrix increases as the smoothing window $T-(t+1)$ increases. The formulation of this matrix can be described in a recursive form at a general time t, from the end time T, as:

$$
D_{t:T}(x_{t-1}) = \begin{bmatrix} D_{t+1:T}(x_{t-1}) & 0 \\ 0 & R_t \end{bmatrix} + \begin{bmatrix} C_{t+1:T}(x_{t-1}) \\ H_t|_{x_{t-1}} \end{bmatrix} Q_t \begin{bmatrix} C_{t+1:T}^\mathsf{T}(x_{t-1}) & H_t^\mathsf{T}|_{x_{t-1}} \end{bmatrix}^\mathsf{T} \tag{2.15}
$$

where the $C_{t+1:T}(x_{t-1})$ is a collection of Jacobian matrices for all predicted measurements from time $t+1$ to T, and defined by the recursion,

$$C_{t:T}(x_{t-1}) = \begin{bmatrix} C_{t+1:T}(x_{t-1}) \\ H_t|_{x_{t-1}} \end{bmatrix} F_{t-1:t}|_{x_{t-1}}. \tag{2.16}$$

These Jacobian matrices in $D_{t:T}(x_{t-1})$ and $C_{t:T}(x_{t-1})$ are evaluated about a point x_{t-1}, but over the entire smoothing time window $[t,\ T]$.

Given the backward corrector for time t, it can be shown that the backward corrector at time $t-1$ is defined as:

$$B_{t-1}(x_{t-1}|z_{1:T}) = \frac{\mathcal{N}([z_{t+1:T}^{\mathsf{T}}, z_t^{\mathsf{T}}]^{\mathsf{T}}; \zeta_{t:T}(x_{t-1}), D_{t:T}(x_{t-1}))}{r_t \nu_t(z_t)} \tag{2.17}$$

where the parameters are computed using Eqs. (2.15) and (2.16). The derivation of this closed-form recursive backward corrector calculation is shown in Appendix A.

If the models in Eqs. (2.3) and (2.4) are linear and Gaussian, the backward corrector in Eq. (2.17) reduces Eqs. (2.12)–(2.14) and Eqs. (2.15)–(2.16) into that of Vo *et al*. [76]. This form is complete in that all matrices are stacked constants, which makes the calculation of backward corrector recursive in closed-form without a need of re-computing Jacobians.

The backward corrector, B_t, represents the likelihood of the predicted measurements (ζ) against the true measurements (z) available at future times (up to time T). Substituting the backward corrector Eq. (2.11) and the posterior distribution, $p(x_t|z_{1:t}) = \sum_{i=1}^{N_t} \omega^i \cdot \mathcal{N}(x_t; m_t^i, P_t^i)$, which is a Gaussian mixture, into the smoothing equation Eq. (2.9), the smoothed estimate at time t is then written as

a Gaussian mixture

$$p(x_t|z_{1:T}) = p(x_t|z_t)B_t(x_t|z_{1:T}) \tag{2.18}$$

$$= \sum_{i=1}^{N_t} \frac{\omega^i \cdot \mathcal{N}(x_t; m_t^i, P_t^i) \cdot \mathcal{N}(z_{t+1:T}; \zeta_{t+1:T}(x_t), D_{t+1:T}(x_t))}{r_t} \tag{2.19}$$

$$= \sum_{i=1}^{N_t^*} \frac{q^i \cdot \omega^i}{r_t} \cdot \mathcal{N}(x_t; m_t^{*i}, P_t^{*i})$$

$$= \sum_{i=1}^{N_t^*} \omega^{*i} \mathcal{N}(x_t; m_t^{*i}, P_t^{*i}) \tag{2.20}$$

where

$$m_t^{*i} = m_t^i + K_t^i(z_{t+1:T} - \zeta_{t+1:T}^i(m_t^i)) \tag{2.21}$$

$$K_t^i = P_t^i(C_{t+1:T}^i(m_t^i))^\mathsf{T}(C_{t+1:T}^i(m_t^i)P_t^i(C_{t+1:T}^i(m_t^i))^\mathsf{T} + D_{t+1:T}^i(m_t^i))^{-1} \tag{2.22}$$

$$P_t^{*i} = (I - K^i C_{t+1:T}^i(m_t^i))P_t^i \tag{2.23}$$

$$q^i = \mathcal{N}(z_{t+1:T}; \zeta_{t+1:T}^i(m_t^i), D_t^i(m_t^i) + C_{t+1:T}^i(m_t^i)P_t^i(C_{t+1:T}^i(m_t^i))^\mathsf{T}) \tag{2.24}$$

$$= \frac{\exp(-\frac{1}{2}(z_{t+1:T} - \zeta_{t+1:T}^i(m_t^i))^\mathsf{T}(D_t^i(m_t^i) + C_{t+1:T}^i(m_t^i)P_t^i(C_{t+1:T}^i(m_t^i))^\mathsf{T})^{-1}(z_{t+1:T} - \zeta_{t+1:T}^i(m_t^i))}{\sqrt{(2\pi)^d|D_t^i(m_t^i) + C_{t+1:T}^i(m_t^i)P_t^i(C_{t+1:T}^i(m_t^i))^\mathsf{T}|}}. \tag{2.25}$$

The scalar q^i is a pdf value computed from the Gaussian distribution (as shown in Eq. (2.25)), since all parameters are known. Note that the superscript $()^*$ indicates the parameters for smoothed estimates, and d in Eq. (2.25) represents the dimension of the vector m_t^i. Due to the required use of different linearization points m_t^i, each mixand of $p(x_t|z_{1:t})$ in Eq. (2.18) has a different value for the backward corrector. This also changes the values of $\zeta_{t+1:T}(x_t)$ and $D_{t+1:T}(x_t)$ for each i-th mixand $\omega^i \cdot \mathcal{N}(x_t; m_t^i, P_t^i)$ in Eq. (2.19), resulting in $C_{t+1:T}^i(m_t^i)$, $D_{t+1:T}^i(m_t^i)$, and $\zeta_{t+1:T}^i(m_t^i)$, along with the backward corrector for each mixand, $B_t^i(x_t|z_{1:T})$,

$$B_t^i(x_t|z_{1:T}) = \frac{\mathcal{N}(z_{t+1:T}; \zeta_{t+1:T}^i(m_t^i), D_{t+1:T}^i(m_t^i))}{r_t} \tag{2.26}$$

where i denotes the index of the mixand. It is unnecessary to explicitly compute the normalization constant r_t in Eq. (2.19), as the smoothed weights, $\omega_t^{i^*}$, are normalized each time ($\omega^{*^i} \propto q^i \omega^i$). Note that the update equations Eqs. (2.21)–(2.24) resemble those of the Kalman filter update, and now give a closed-form solution for the desired nonlinear Gaussian mixture smoother.

2.3.2 Least Squares Estimate of the Backward Corrector

While Eqs. (2.21)–(2.24) give a closed-form solution of Gaussian mixture smoothing for nonlinear systems, there exist computational challenges for computing the backward corrector as the smoothing window $T - t$ increases. As the time of smoothing, t, moves backward, the linearization point, m_t^i, changes for each time (t) and each mixand (i). Because the Jacobian matrices for the predicted measurements from time $t + 1$ to time T are computed based on these linearization points, the stacked matrices $C_{t+1:T}^i(m_t^i)$ and $D_{t+1:T}^i(m_t^i)$ must be recomputed for each m_t^i. The computation time increases linearly with the increasing time difference $T - t$ for each measurement. Thus, the backward corrector at time t (Eq. (2.11)) requires $\sum_{k=1}^{M_m}(t_{\text{meas}}^k - t) \cdot \delta t / \delta t_{\text{sampling}}$ seconds to calculate, where M_m represents total number of measurements available in time interval, $[t, T]$, t_{meas}^k represents the time of k-th measurement occurred, and δt and $\delta t_{\text{sampling}}$ represent the Jacobian evaluation time and sampling time.

Consider now the form of the backward corrector; from Eq. (2.9), the backward corrector can be calculated by dividing the smoothed estimate by the pos-

terior distribution as shown in Eq. (2.27),

$$B_t(x_t|z_{1:T}) = \frac{p(x_t|z_{1:T})}{p(x_t|z_{1:t})} \tag{2.27}$$

$$= \frac{\mathcal{N}(z_{t+1:T}; \zeta_{t+1:T}(x_t), D_{t+1:T}(x_t))}{r_t}. \tag{2.28}$$

The posterior distribution, $p(x_t|z_{1:t})$, is a Gaussian mixture; therefore the division $\frac{p(x_t|z_{1:T})}{p(x_t|z_{1:t})}$ in Eq. (2.27) cannot be computed analytically.

In order to decrease the computation time and enable real-time recursion while maintaining accuracy, an estimate of the backward corrector is proposed using least squares fit. Instead of propagating the state for the linearized backward corrector all the way to time T (Eqs. (2.12)–(2.14) and (2.15)–(2.16)), an estimate of the measurement, $\hat{z}_{t+1:t+1}$ (called pseudo-measurement) and its associated measurement noise are used.

More formally, an estimate of the backward corrector, $\hat{B}_t(x_t|z_{1:T})$,

$$\hat{B}_t(x_t|z_{1:T}) = \frac{\mathcal{N}(\hat{z}_{t+1:t+1}; \hat{\zeta}_{t+1:t+1}(x_t), \hat{D}_{t+1:t+1}(x_t))}{r_t}, \tag{2.29}$$

requires three variables to be estimated: the measurement $\hat{z}_{t+1:t+1}$, the predicted measurement $\hat{\zeta}_{t+1:t+1}(x_t)$, and the predicted measurement noise $\hat{D}_{t+1:t+1}(x_t)$. Note that the estimated measurement $\hat{z}_{t+1:t+1}$ is not written as an explicit function of x_t because it is same for all mixands. Also, while the predicted measurement $\hat{\zeta}_{t+1:t+1}(x_t)$ and noise covariance $\hat{D}_{t+1:t+1}(x_t)$ are functions of the state, due to the linearization, the backward corrector must be estimated for *each mixand*, resulting in $\hat{B}^i_t(x_t|z_{1:T})$,

$$\hat{B}^i_t(x_t|z_{1:T}) = \frac{\mathcal{N}(\hat{z}_{t+1:t+1}; \hat{\zeta}^i_{t+1:t+1}(m^i_t), \hat{D}^i_{t+1:t+1}(m^i_t))}{r_t} \tag{2.30}$$

where the predicted measurement ($\hat{\zeta}^i_{t+1:t+1}(x_t)$) and associated Jacobian matrix

($\hat{C}^i_{t+1:t+1}(x_t)$)to evaluate $\hat{D}_{t+1:t+1}(x_t)$ for each mixand are set as:

$$\hat{\zeta}^i_{t+1:t+1}(m^i_t) = h(f(m^i_t)) \tag{2.31}$$

$$\hat{C}^i_{t+1:t+1}(m^i_t) = H_{t+1}|_{m^i_t} \cdot F_{t:t+1}|_{m^i_t}. \tag{2.32}$$

The dimension of $\hat{\zeta}^i_{t+1:t+1}$ is the same as a single measurement, z_{t+1}; similarly, $\hat{\zeta}^i_{t+1:t+1}$ and $\hat{D}^i_{t+1:t+1}$ have the same dimension (i.e. $z_{t+1} = \Re^{d\times 1}$, $\hat{\zeta}^i_{t+1:t+1} \in \Re^{d\times 1}$, $\hat{D}^i_{t+1:t+1} \in \Re^{d\times d}$). Given the predicted measurement, $\hat{\zeta}^i_{t+1:t+1}(m^i_t)$, estimated from the mean of the posterior distribution at time t, the gain matrix K^i_t in Eq. (2.22) for each mixand is estimated using Eq. (2.23) as follows:

$$\begin{aligned} \Delta P^i_t &= P^{*^i}_t - P^i_t \\ &= -K^i_t \hat{C}^i_{t+1:t+1}(m^i_t) P^i_t \end{aligned} \tag{2.33}$$

$$-\Delta P^i_t (P^i_t)^{-1} = K^i_t \hat{C}^i_{t+1:t+1}(m^i_t) \tag{2.34}$$

$$-\Delta P^i_t (P^i_t)^{-1} (\hat{C}^i_{t+1:t+1}(m^i_t))^\mathsf{T} (\hat{C}^i_{t+1:t+1}(m^i_t)(\hat{C}^i_{t+1:t+1}(m^i_t))^\mathsf{T})^{-1} = K^i_t. \tag{2.35}$$

From Eq. (2.22), the noise covariance is obtained for each mixand using pseudo-inverse as shown below:

$$\hat{D}^i_{t+1:t+1} = ((\hat{C}^i_{t+1:t+1}(m^i_t)(\hat{C}^i_{t+1:t+1}(m^i_t))^\mathsf{T})^{-1} \hat{C}^i_{t+1:t+1}(m^i_t)(P^i_t)^{-1} K^i_t)^{-1} - \hat{C}^i_{t+1:t+1}(m^i_t) P^i_t (\hat{C}^i_{t+1:t+1}(m^i_t))^\mathsf{T}. \tag{2.36}$$

Finally, an estimate of the measurement, $\hat{z}_{t+1:t+1}$, is then obtained using a least squares fit over all mixands. The mixand mean difference is written as:

$$\Delta m^i_t = m^{*^i}_t - m^i_t = K^i_t(\hat{z}_{t+1:t+1} - \hat{\zeta}^i_{t+1:t+1}(m^i_t)) = K^i_t(\hat{z}_{t+1:t+1} - h(f(m^i_t))). \tag{2.37}$$

Rearranging to separate out the estimated measurement $\hat{z}_{t+1:t+1}$, and stacking for all mixands results in

$$\begin{bmatrix} \Delta m^1_t + K^1_t h(f(m^1_t)) \\ \vdots \\ \Delta m^{N_t}_t + K^{N_t}_t h(f(m^{N_t}_t)) \end{bmatrix} = \begin{bmatrix} K^1_t \\ \vdots \\ K^{N_t}_t \end{bmatrix} \hat{z}_{t+1:t+1}, \qquad \forall i \in \{1, \cdots, N_t\}. \tag{2.38}$$

The estimated measurement $\hat{z}_{t+1:t+1}$ can be found via pseudo-inverse on Eq. (2.38).

The process of computing the fast, recursive smoothed estimates is as follows. First, the smoothed estimate for time t, $p(x_t|z_{1:T})$, is computed using the available backward corrector from previous step. Then, an estimate of the backward corrector, $\hat{B}_t^i(x_t|z_{1:T})$, is computed using Eqs. (2.31)–(2.38) for each mixand; this step essentially reduces the dimension of the backward corrector to the size of one measurement. Then, $\hat{B}_{t-1}^i(x_{t-1}|z_{1:T})$ at time $t-1$ is obtained from $\hat{B}_t^i(x_t|z_{1:T})$ by Eq. (2.10), using Eqs. (2.15)–(2.17) for all $i \in [1, \ldots, N_{t-1}]$. Finally, the smoothed estimates at time $t-1$, $p(x_{t-1}|z_{1:T})$, is obtained using $\hat{B}_{t-1}^i(x_{t-1}|z_{1:T})$ and Eq. (2.18). Note that the backward corrector $\hat{B}_t(x_t|z_{1:T})$ is only complete if given a full set of all N_{t-1} mixands.

By estimating the backward corrector, $\hat{B}_t^i(x_t|z_{1:T})$, directly from the smoothed and posterior estimates at each time step, the computation time to estimate the backward corrector is constant for a constant number of mixands. The time to compute $B_{t-1}^i(x_{t-1}|z_{1:T})$ therefore is also constant by Eq. (2.10). Note that in order to avoid numerical instability, an information form of the measurement update step can be utilized to avoid the subtraction in Eq. (2.36).

2.4 Adaptive Gaussian Mixture Model Smoother for Nonlinear Systems

Section 2.3 describes closed-form smoothing solutions for nonlinear systems with Gaussian mixture prior. The propagation and measurement models are

described as a single Gaussian, which implies that the number of mixands in the posterior distribution does not change over time. However, for highly nonlinear systems, a fixed number of mixands cannot sufficiently capture the varying degree of nonlinearity and non-Gaussianity, and thus can lead to inconsistent and inaccurate estimates over time. Previous researchers have demonstrated successful filtering performance using adaptive Gaussian splitting algorithms [26, 41, 75, 13, 70] which nicely capture the effects from these severe nonlinearities over long periods of time. An analogous Gaussian mixture smoother is proposed here, which adapts the number of Gaussian mixands over time, as the nonlinearity effects change; the newly proposed adaptive Gaussian smoother will address a wider variety of complex estimation and smoothing problems. The key challenge in the proposed adaptive Gaussian mixture smoother is to develop an approach to smooth back and forth over time, as the number of mixands varies.

2.4.1 Overview of Gaussian Mixture Model Smoother

In the general adaptive Gaussian mixture model smoother, unlike the static Gaussian mixture model smoother derived in Section 2.3, the number of mixands vary over time in order to capture the evolution of complex pdfs through the nonlinear dynamics. A combination of Gaussian splitting and condensation methods are used to address smoothing across time steps, while maintaining consistent associations across time steps.

Association variables θ_t are first introduced to capture the assignments of mixands across time steps. The smoothed distribution, $p(x_t|z_{1:T})$, in Eq. (2.1) is

then rewritten as:

$$p(x_t|z_{1:T}) = \sum_{\theta_t} p(x_t, \theta_t|z_{1:T}) \tag{2.39}$$

where

$$\theta_t = \begin{bmatrix} 0 & \cdots & 1 & \cdots & 0 \end{bmatrix}^{\mathsf{T}}. \tag{2.40}$$

The association variable θ_t is a binary column vector, such that the k-th element $\theta_t(k)$ satisfies: $\theta_t(k) \in \{0, 1\}$ and $\sum_k \theta_t(k) = 1$; $\theta_t(k) = 1$ indicates that the k-th mixand at time $t + 1$ was propagated from the k-th mixand at time t. This requires the number of mixands at t and $t + 1$ are the same during the smoothing process. The smoothed distribution can then be factored as follows:

$$p(x_t|z_{1:T}) = \sum_{\theta_t} \underbrace{p(x_t|\theta_t, z_{1:T})}_{\text{smoothing}} \cdot \underbrace{p(\theta_t|z_{1:T})}_{\text{association update}}. \tag{2.41}$$

Equation (2.41) shows a factorization of the smoothed estimate with association variables. The second factor, $p(\theta_t|z_{1:T})$, represents the associations of the smoothed mixands, whereas the first factor, $p(x_t|\theta_t, z_{1:T})$, represents the smoothed distribution of the state variables, given that the associations are known. Intuitively, given the measurements *and* associations, the first factor, $p(x_t|\theta_t, z_{1:T})$, is equivalent to a Kalman smoother, where the smoothed mean vector and covariance matrix are estimated.

Splitting is required when the propagation of a Gaussian through the nonlinear dynamics does not produce another Gaussian. Following Ref. [25], the splitting process for the proposed adaptive Gaussian mixture smoother can be accomplished as follows: 1) measure the non-Gaussianity of the propagated distribution, 2) if the non-Gaussianity is greater than a threshold, then split the initial Gaussian and propagate all mixands again. Many metrics exist to compute the non-Gaussianity vector as presented in Refs. [18, 17, 26, 25, 75, 32]; if

the nonlinearity measure exceeds a threshold or one of these metrics, the Gaussian distribution splits into a Gaussian mixture for the next time step. It is noted that a Gaussianity metric in most of these splitting routines could be used in the proposed adaptive Gaussian mixture smoother without loss of generality.

The Gaussian splitting is described more specifically by considering the propagation step,

$$p(x_{t+1}|z_{1:t}) = \int \mathcal{N}(x_{t+1}; f(x_t), Q_t) \cdot \underbrace{\mathcal{N}(x_t; m_t, P_t)}_{\text{split}} dx_t. \tag{2.42}$$

The propagated distribution $p(x_{t+1}|z_{1:t})$ in Eq. (2.42) is evaluated for Gaussianity. In this work, the residual vector of sigma-points after fitting a linear transformation to the nonlinear function $f(x_t)$ is used because of its simplicity in implementation and accuracy, as described in Ref. [25]. If the output distribution is considered to be too far from a Gaussian, the original distribution is split, creating a new, larger Gaussian mixture, shown here with M mixands,

$$p(x_{t+1}|z_{1:t}) \cong \int \mathcal{N}(x_{t+1}; f(x_t), Q_t) \cdot \sum_{i=1}^{M} \omega^i \cdot \mathcal{N}(x_t; m_t^i, P_t^i) dx_t \tag{2.43}$$

$$= \sum_{i=1}^{M} \int \mathcal{N}(x_{t+1}; f(x_t), Q_t) \cdot \omega^i \cdot \mathcal{N}(x_t; m_t^i, P_t^i) dx_t \tag{2.44}$$

$$= \sum_{i=1}^{M} \omega^i \cdot \mathcal{N}(x_{t+1}; f(m_t^i), P_{t+1}^i). \tag{2.45}$$

The number of Gaussians in the mixture representing $p(x_{t+1}|z_{1:t})$ has increased from one to M in Eq. (2.45). The above equations can be easily adapted for the general case of N_t mixands to N_{t+1} mixands.

In the closed-form Gaussian mixture smoother presented in Section 2.3, $N_t = N_{t+1}$ and the association across the time steps is clear. However, when splitting occurs, as in the adaptive Gaussian mixture smoother, the association variable θ_t must map the N_t mixands at time t to the N_{t+1} mixands at time $t+1$. The growth

and shrinkage of mixands over time is most easily described by introducing a tree structure for the mixtures over time.

2.4.2 Tree Construction

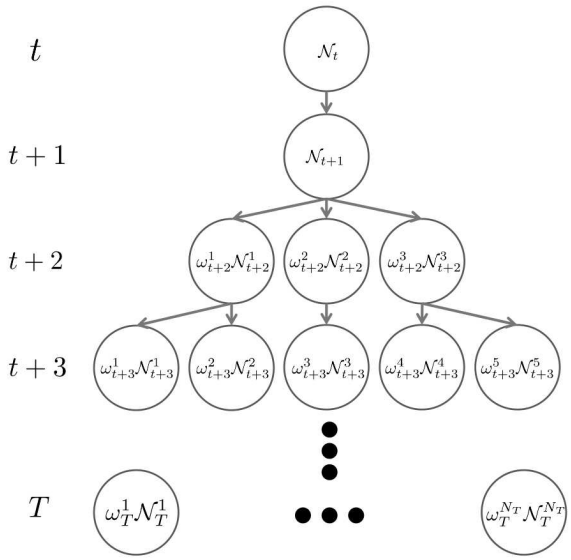


Figure 2.2: An example of forward tree undergoing splitting for initial four time steps starting at t.

A graph denoted as the forward tree is constructed as the Gaussian mixture propagates forward in time. Each node of the tree represents a mixand of a Gaussian mixture, and each depth represents a time step. For terminology, a *parent* node splits into multiple nodes (mixands) which are called *children*. Figure 2.2 shows an example of forward tree constructed. The vertical axis represents the time flow from time t to T, and circles denote mixands of the Gaussian mixture for each time step. Note that the specific parameters of each node/Gaussian distribution are omitted for notational simplicity, and only the weights, ω_t^j, are shown in order to demonstrate how the Gaussian mixture grows in a tree structure during propagation; the subscript t indicates the time

step, and the superscript i indicates the index of mixands. In Fig. 2.2, no split occurs from time t to time $t+1$. At time $t+1$, the non-Gaussianity threshold is reached and splitting is required, spawning three children nodes at time $t+2$. At time $t+2$, the first and third nodes split into two children nodes each, resulting in total of five mixands at $t+3$. Because no condensation or reduction of mixtures occurs during the forward propagation phase, the number of child nodes is the same (no splitting) or greater than the parents (splitting).

2.4.3 Methods to Obtain Smoothed Estimate, $p(x_t|\theta_t, z_{1:T})$

Given the forward tree representing Gaussian mixtures over the time window $t \to T$, backward smoothing methods are then developed. These methods use a nonlinear Kalman smoother as a baseline, which can be implemented via linearization or sigma-point/unscented transformation [58, 57, 56]. In the current literature, all variants of Kalman smoothers require *one-to-one* relationship between the parents and children; this is consistent with the proposed requirement of the association variable, θ_t. When splitting occurs, the number of children nodes is greater than the number of parent nodes, leading to a *one-to-many* relationship. In order to compensate for this association problem, two smoothing methods are developed: *children collapsing* and *parent splitting*.

The *children collapsing* (CC) method condenses children nodes with the same parent into a single node, such that a one-to-one relationship with the parent is created. This relationship, in turn, allows the nonlinear Kalman smoother to be applied. Given the tree structure described in Section 2.4.2, the assignments between the condensed child nodes and the parents are guaranteed; a

parent node in a tree only has children nodes if they originate from the parent node. Figure 2.3 shows an example of the children collapsing method. Three stages are shown for clarity. In 2.3(a), a forward tree with four time steps are shown, which is identical to that in Fig. 2.2. At time $t + 3$, the five nodes are assumed to be already smoothed (shown in gray). In order to obtain smoothed estimates for time $t + 2$, the two sets of children (denoted with boxes in 2.3(a)) are condensed in order to create the required one-to-one relationship, as shown in 2.3(b). Given the one-to-one relationships between parents $(t+2)$ and children $(t + 3)$, a nonlinear Kalman smoother is used to produce the smoothed estimates for time $t + 2$, as shown in 2.3(c). Note that the collapsing/condensation step is accomplished by matching the moments. More specifically, for condensing M Gaussians, following formulas are used [6].

$$m_t^{\text{cond}} = \sum_{i=1}^{M} \frac{\omega_t^i}{\omega_t^{\text{cond}}} \cdot m_t^i \tag{2.46}$$

$$P_t^{\text{cond}} = \sum_{i=1}^{M} \left(\frac{\omega_t^i}{\omega_t^{\text{cond}}} \cdot (P_t^i + m_t^i (m_t^i)^{\mathsf{T}}) \right) - m_t^{\text{cond}} \cdot (m_t^{\text{cond}})^{\mathsf{T}} \tag{2.47}$$

$$w_t^{\text{cond}} = \sum_{i=1}^{M} \omega_t^i. \tag{2.48}$$

Unlike the children collapsing method, the *parent splitting* (PS) method splits the parent into the same number of mixands as the children, using the same splitting parameters that were used during the forward tree construction. More specifically, during the forward filtering, the axis to split each Gaussian is evaluated based on the non-Gaussianity calculation and stored. The same axis is used again to split the parent so that one-to-one relationships between the split parents and children are made. Utilizing the tree structure and the splitting order, the assignments between split parents and the children are guaranteed. Figure 2.4 shows an example of the parent splitting method. Figure 2.4(a) shows the

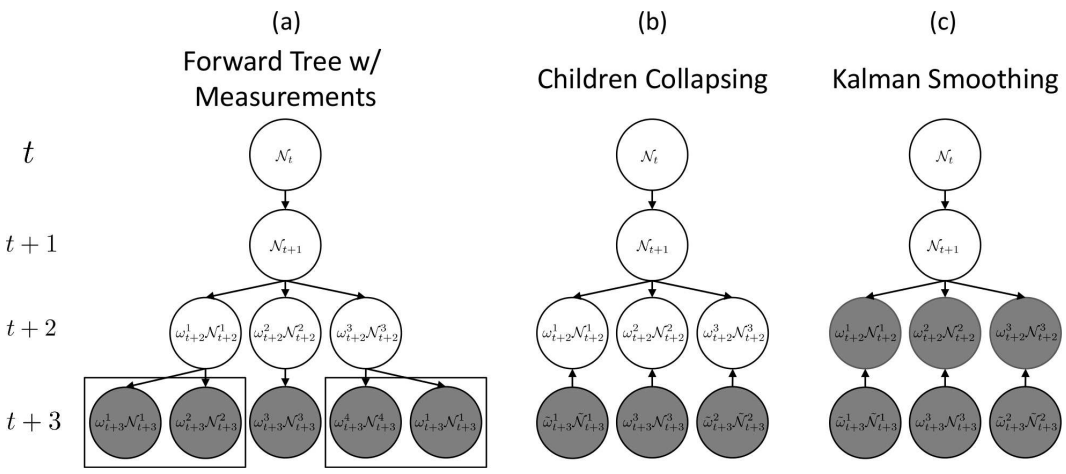


Figure 2.3: An example schematic of children collapsing method. $(\tilde{\cdot})$ indicates the parameters of combined/collapsed nodes.

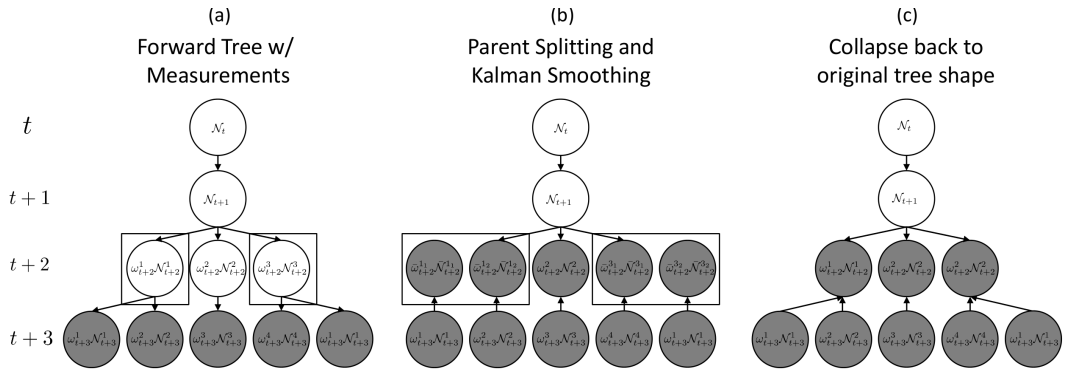


Figure 2.4: An example schematic of parent splitting method. $(\bar{\cdot})$ indicates the parameters of split nodes.

forward tree with five smoothed nodes (in gray) at time $t + 3$, where two boxes indicate the parent nodes with multiple children. In 2.4(b), these parents are split into the same number of the nodes as the children using the stored non-Gaussianity vector. Then a nonlinear Kalman smoother is used to obtain the smoothed estimates for time $t + 2$. In 2.4(c), the smoothed nodes are condensed back to keep the original tree structure.

2.4.4 Update of the Association Factor, $p(\theta_t|z_{1:T})$

While obtaining the smoothed estimates *given* the association variable, $p(x_t|\theta_t, z_{1:T})$, is solved using a nonlinear Kalman smoother in these proposed algorithms, the association factor, $p(\theta_t|z_{1:T})$, must be explicitly computed. The smoothed/updated associations of each Gaussian mixand are computed using the update step for the linearized forward-backward algorithm, as shown in Eq. (2.24); the smoothed weight update requires a backward corrector. For short time windows, the linearized closed-form solution in Section 2.3.1 (Eqs. (2.21)–(2.24)) works well. However, for longer smoothing windows, the backward corrector estimation method, as proposed in Section 2.3.2 (Eqs. (2.31)–(2.38)), may allow better computation versus performance trade-offs.

2.5 Simulation and Experiment Result

2.5.1 Simulation

A simulation of a nonlinear system is used to demonstrate and understand the performance of the proposed smoothing methods. The *univariate non-stationary growth model* (UNGM) is used because of its wide acceptance for nonlinear filter evaluations [15, 23]. The UNGM is described with following equations:

$$x_k = \alpha x_{k-1} + \beta \frac{x_{k-1}}{1 + x_{k-1}^2} + \gamma \cos(1.2(k-1)) + q_k \tag{2.49}$$

$$z_k = \frac{x_k^2}{20} + r_k \tag{2.50}$$

where k represents the time index. For this simulation, $\alpha = 1, \beta = 1, \gamma = 1$ are used, and $q_k \sim \mathcal{N}(q_k; 0, 0.1^2)$, $r_k \sim \mathcal{N}(r_k; 0, 20^2)$. Note that $\alpha = 1$ to prevent the

model from converging over time.

Five different implementations of the adaptive Gaussian mixture model smother are compared, along with a sigma-point smoother (SPS, also known as unscented Kalman smoother) as a benchmark. The five Gaussian mixture model smoothers are: 1) CC method with full weight update, 2) PS method with full weight update, 3) CC method with least squares (LS) backward corrector (BC) estimator, 4) PS method with LS BC estimator, and 5) linearized forward-backward (FB) algorithm as described in Section 2.3. The full weight update uses Eq. (2.24) for the weight update; the full backward corrector is computed as in Section 2.3.1, but is only used for the weight update in 1) and 2).

The simulation is run 1000 times, each with 50 time steps. Only one measurement is available, at the last time step ($k = 50$) and smoothing is done once for each of the five smoother implementations; thus, all smoothed estimates in previous time steps are affected by this only measurement. This long time window with no measurement is purposely used to demonstrate the effectiveness of the smoothing methods in a challenging case of sparse measurements. For each smoother, the Kullback-Leibler divergence (KLD) is computed to evaluate performance [39],

$$D_{\mathrm{KL}}(P\|Q) = \int \ln\left(\frac{p(x)}{q(x)}\right) p(x)dx. \tag{2.51}$$

The KLD computes the expectation of log difference between two distributions, and is not a symmetric measure (i.e. generally $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$). Typically, $p(x)$ is often a 'true' distribution that is precisely calculated, whereas $q(x)$ is a distribution that approximates $p(x)$. Among other divergences, KLD is chosen because it measures how well the smoothed estimates represent the true distribution. The *true* forward predicted and backward smoothed distributions are

numerically computed with a 0.05 resolution grid using Eq. (2.5) and Eq. (2.1), respectively; integrals are replaced with summations to use in discretized grid space.

KLD values are first computed between the true predicted distribution (numerically computed), and the predicted distributions from a sigma-point predictor and an adaptive Gaussian mixture model predictor with splitting. The average KLD values over 1000 trials for the sigma-point predictor over 50 time steps is 0.7068, while that of adaptive Gaussian mixture model predictor is 0.4120. The KLD values for sigma-point predictor are, in fact, always higher than those of adaptive Gaussian mixture model predictor at all time steps. These results indicate that adaptively splitting the Gaussian mixture enables the true distribution shape to be better predicted than using a single Gaussian.

Figure 2.5 compares the mean KLD values of the five smoothers, and the sigma-point smoother (SPS) over time. The SPS has higher KLD values compared to all other smoothers for most of the time (46 out of 50). Between the time $k = 10$ and $k = 50$, the KLD values of CC and PS with LS BC estimator are higher than those of using full weight update most of the time. This indicates the LS BC estimator slowly loses information over time, which is expected because only two consecutive time steps are used for the LS BC estimate (Section 2.3.2). With recurring BC estimates over time, the KLD values become higher than that of SPS for $k > 4$. The CC and PS with full weight update, however, have KLD values that are very close to the linearized FB algorithm for all time, which best represents the true distribution. The KLD analysis clearly indicates that a full backward corrector is the best choice if the computation is affordable.

Figure 2.6 shows the evolution of the smoothed distribution for one trial and

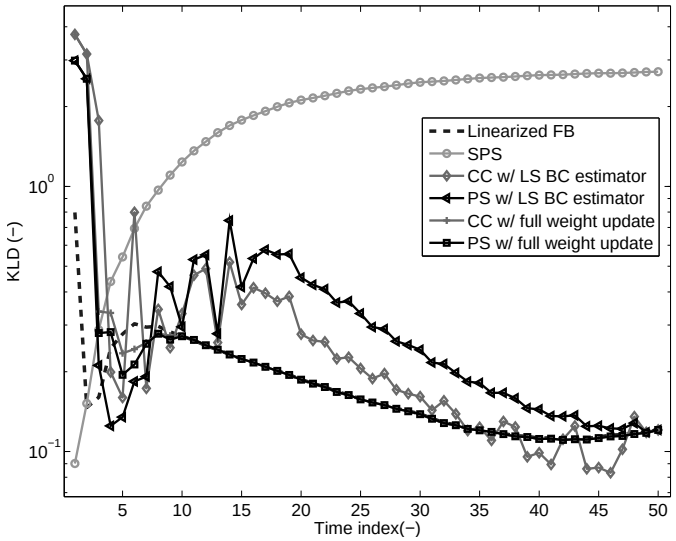
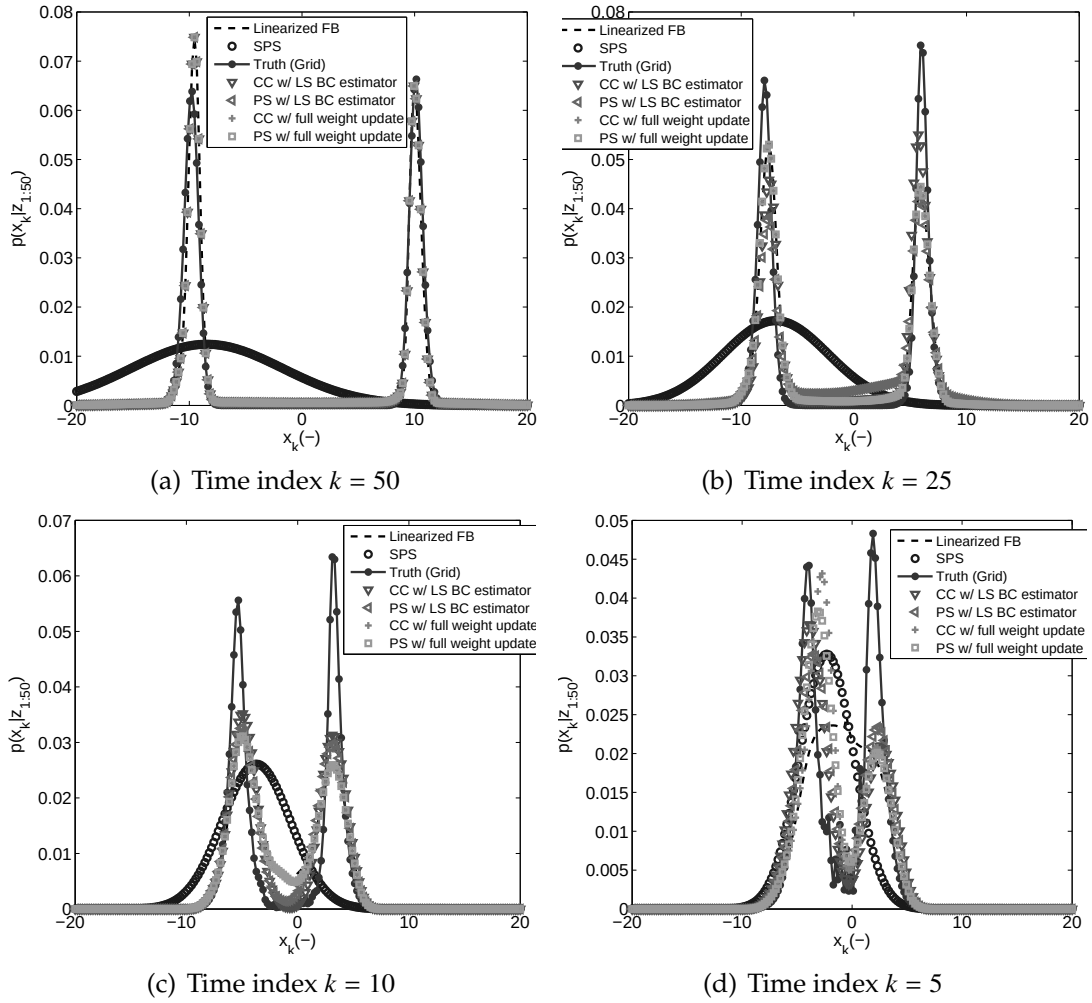


Figure 2.5: KLD computed for distributions of five adaptive Gaussian mixture smoothers, and sigma-point smoother.

four different time steps, $k \in \{50, 25, 10, 5\}$. At the beginning of the smoothing process, as shown in Fig. 2.6(a), all adaptive Gaussian mixture smoothers show the same distribution, including capturing the peaks of the true distribution. The SPS, however, does not capture the multi-modal distribution. This trend continues as the smoother runs back to $k = 25$ (Fig. 2.6(b)). Figure 2.6(c) shows the smoothed distributions at $k = 10$. All adaptive Gaussian mixture smoothers show slightly wider peaks compared to the truth. While the SPS still could not capture the multi-modal distribution, the Gaussian mixture smoothers and linearized FB algorithm qualitatively captures the true distribution well. The CC and PS with full weight update (shown with + and □ markers) nearly match the linearized FB algorithm; this is also shown in the previous time steps (e.g. $k = 25$). Figure 2.6(d) shows the distributions at $k = 5$. While the same trends are clear, all five Gaussian mixture smoothers show distributions that are slightly different from one another. With recurring backward corrector estimates, the CC and PS curves capture the primary characteristics of the true shape includ-



(a) Time index $k = 50$

(b) Time index $k = 25$

(c) Time index $k = 10$

(d) Time index $k = 5$

Figure 2.6: Evolution of smoothed distributions over time. Linearized FB algorithm and CC and PS methods with full weight update overlap for (a), (b), and (c).

ing two peaks. As the smoothing time is close to the start of the window, and the true smoothed distribution (e.g. $p(x_5|z_{1:50})$) is smaller and tighter, the single Gaussian distribution of the SPS covers the true multi-modal distribution better compared to that of later time steps, as the decreasing KLD values indicates (Fig. 2.5).

The average time to compute the BC for the linearized FB smoother along with the PS and CC methods with LS BC estimator is shown in Table 2.1. The

	Absolute Time (s)	Relative Scale Factor (-)
Linearized FB	0.02957	4.6937
CC w/ LS BC Estimator	0.00633	1.0048
PS w/ LS BC Estimator	0.00630	1.0

Table 2.1: Average absolute and relative time to compute the backward corrector (BC) for three versions of the smother implementations in the simulation.

absolute execution time is measured in MATLAB on a Core i7-950 desktop computer; scale factors of relative execution time are shown as well to remove hardware specific dependencies. The average computation time for linearized FB smoother is more than four times larger compared to the LS BC estimator. Note that the computation time for the linearized FB smoother grows as $T - t$ window increases because the number of propagation steps increases; the CC and PS with LS BC estimator approaches maintain constant computation time as $T - t$ increases. Thus, for simulations and experiments with large smoothing windows, the computation time savings of the backward corrector calculation become more significant.

2.5.2 Experiment

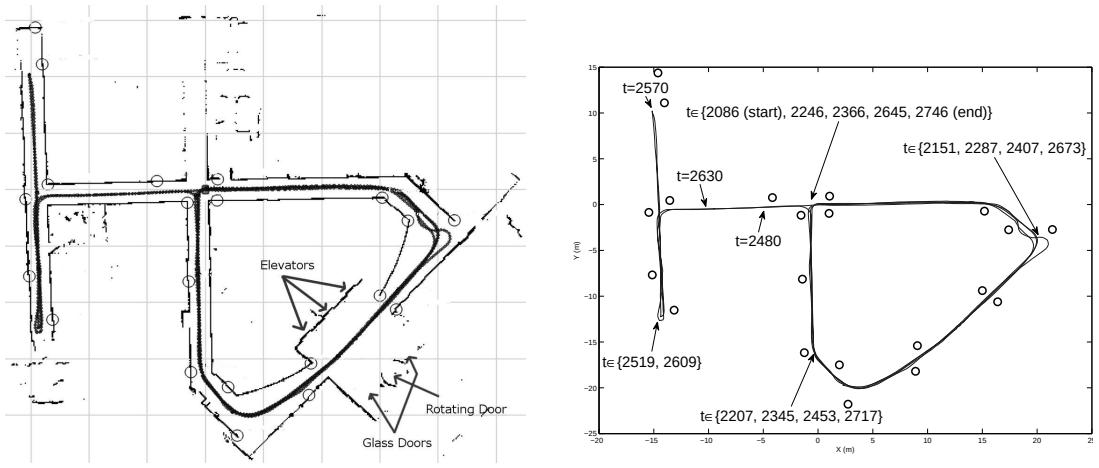
Experimental Setup

An indoor navigation problem using vision sensing in sparse-featured environment is presented to demonstrate and analyze the performance of the proposed smoothing methods. This application is particularly good at elucidating the key elements of the problem space – highly nonlinear dynamics and sparse measurements.

The mobile robot used in this experiment is based on Segway RMP50XL platform with a custom frame and sensor mountings. The robot has two primary wheels and a third caster wheel that has no actuation. For this work, the sensors used include a MEMS inertial measurement unit (IMU), two wheel encoders, a set of three cameras for panoramic view, and a SICK laser range finder. The three cameras together cover approximately 150 degrees of field of view, and provide measurement of features of the environment for the smoothing algorithms.

The robot motion is modeled using 15 states: inertial x, y, z positions, local x, y, z velocities, three inertial Euler angles, three accelerometer and three gyroscope biases. The forward filter is initialized with a single Gaussian with predefined noise covariances. The six IMU biases are initialized after collecting 500 data packets while the robot remains motionless. The robot model is driven using accelerometer and gyroscope readings of the IMU. Wheel odometry and vision inputs are used as measurements in the localization algorithm.

An environment is set up in Rhodes Hall at Cornell University, with Augmented Reality Tags (ARTags) used as environmental measurements [20]. Vision based measurements of the ARTag give unique identification numbers, and position $(x_{\mathrm{AR}}, y_{\mathrm{AR}}, z_{\mathrm{AR}})$ with respect to the camera. Each ARTag is a known feature in *a priori* map. Figure 2.7(a) shows the 2D map of the environment along with the ARTag locations (circles). Figure 2.7(a) also shows the output of grid-based FastSLAM [24] using laser range finder to create the occupancy grid map with 10 centimeter resolution; the grid lines shown in the figure are five meters apart for ease of reading. The FastSLAM output trajectory is treated as truth for these experiments.



(a) ARTag (O) map arrangement in the indoor experiment environment is shown along with the grid-based FastSLAM trajectory output.

(b) Robot trajectory (from FastSLAM) with time steps in seconds. Circles represent the ARTag locations used.

Figure 2.7: (a) Occupancy grid map and ARTag location for the experiment environment. (b) Robot trajectory is shown along with time of the path at key points.

Figure 2.7(b) shows the trajectory along with time steps in order to help understand how the robot navigated the environment. The robot started at the middle of the map (near the four-way intersection) facing right. The robot completed a total of four loops around the right half side and one and a half loops in the left half side of the map. The repeated loops enabled the FastSLAM algorithm to create an accurate and consistent map of the environment, which was then used as truth. Examining Fig. 2.7(a), the right half side of the map is very consistent and represents the environment well because of the four repeated loops. However, the left side of the map, especially the vertical hallway, is slightly skewed because the path does not include as many loops.

Results

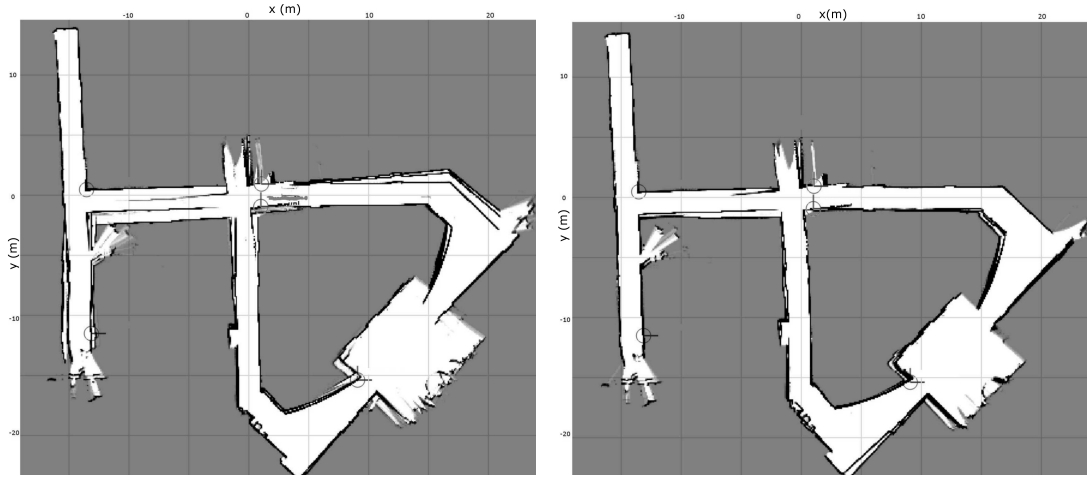
In order to analyze the performance of the smoothing algorithms as a number of key parameters are varied, a single data log was collected with 21 ARTags in the environment. Then, the smoothing algorithms are run over the data log while varying 1) the number of observed ARTags and 2) the maximum number of mixands used after splitting in each of the smoothers. Note that changing the number of ARTags changes the sparseness of measurements in the environment. Changing the maximum number of mixands changes the complexity of the model representation and the computation. In general, using more mixands leads to better estimation/smoothing performance, but at the expense of more computation.

The number of tags are chosen to be one of $\{3, 5, 10, 15, 21\}$ tags, and the maximum number of mixands to use is chosen as one of $\{1, 5, 7\}$ mixands. The splitting threshold is tuned to control the maximum number of mixands. A total of 20 trials are run for each set of ARTags and maximum number of mixands. The CC and PS with LS BC estimator are both used equally. A key goal of the experiment is to show that the LS estimator works well in practical and larger problems while trading accuracy and computation.

Figure 2.8 shows the occupancy grid maps created from both forward (a) and smoothed (b) estimates using five ARTags and a maximum of seven mixands; the smoothed estimates are obtained using CC method. Each figure shows the ARTag locations (circles); the grey, black, and white colors indicate unknown, occupied, and free space, respectively.

The map created from the smoothed estimates is qualitatively similar to

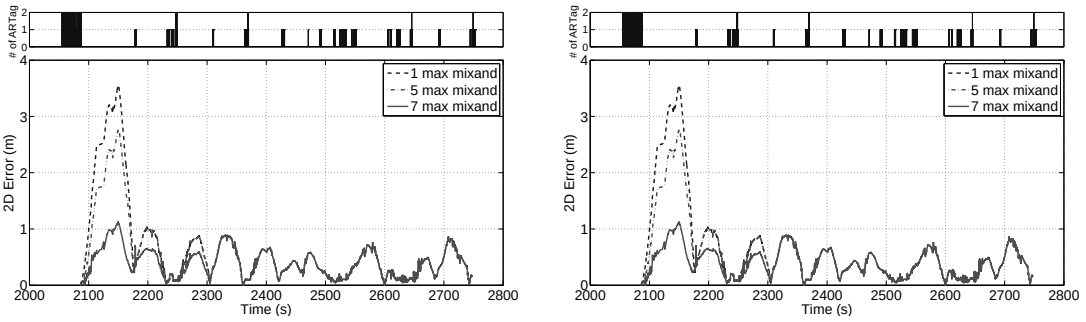
those from the forward estimates, but more consistent. To see this, consider the right hallway (from $x \in [0\text{m},\ 15\text{m}]$) in Figure 2.8(a), where the forward estimates create multiple hallways. Similarly, the vertical hallways located on the left ($x \approx -14\text{m}, y \in [-15\text{m},\ 15\text{m}]$) show multiple hallways as well. The smoothed estimates, however, do not show multiple hallways. The right bottom region ($x \in [13\text{m},\ 24\text{m}], y \in [-8\text{m},\ -20\text{m}]$) is the building entrance with three elevators, two glass doors, and a rotating door (as shown in Fig. 2.7(a)). The forward estimates in Fig. 2.8(a) show a spurious hallway extended in this region ($x \approx 16\text{m}, y \approx -9\text{m}$); the forward estimates are shifted due to the measurement from the ARTag at ($x \approx 9\text{m}, y \approx -15\text{m}$). This spurious hallway does not appear in the maps using the smoothed estimates in Fig. 2.8(b).



(a) Forward estimate (b) Smoothed estimate

Figure 2.8: Occupancy grid map created from both forward and smoothed estimates using the adaptive Gaussian mixture smoothing method with a maximum of seven mixands.

Figure 2.9 shows the 2D position error graphs of using CC and PS Gaussian mixture smoothing methods compared to the grid-based FastSLAM output, along with the number of ARTags in view (top of the figure). The output of FastSLAM is interpolated to match that of smoothed estimates to create the



(a) 2D smoothed error for 5 ARTags with CC method.

(b) 2D smoothed error for 5 ARTags with PS method.

Figure 2.9: 2D error (bottom) of the CC and PS smoothers with LS BC update, as compared to the grid-based FastSLAM output. The figure shows the result of using a total of five ARTags.

plots. Each figure shows the error graphs of using three different maximum mixands: $\{1, 5, 7\}$. All curves show error oscillations due to ARTags moving in and out of sensor view, as shown in the top of the Fig. 2.9. All three curves are nearly identical after ≈2300 seconds; this time corresponds to the robot having completed one and half loops on the right side, and having observed the same ARTags twice (two at the start position and one at the bottom hallway). By observing the ARTags multiple times, the pose uncertainty becomes small enough to be a single Gaussian and there is no benefit to the extra number of mixands. The largest error occurs near the start of the run $t \in [2086\text{s}, 2180\text{s}]$; this interval is between the start (two ARTags in view) and the next available measurement (at $x \approx 9\text{m}, y \approx -15\text{m}$). Because no measurement is available during this period, the error in the forward estimates rapidly increases because of propagation of uncertainty through the nonlinear dynamics; this growth mostly originates from errors in the IMU biases. These errors are improved once the measurements in later times are used with the proposed smoothing algorithm. In general, the maximum number of mixands increases, the smoother is better able to capture the evolving non-Gaussian distribution over time, and the error is smaller. Be-

cause a more accurate filtered posterior distribution, $p(x_t|z_{1:t})$, can be obtained using more mixands, the accuracy and consistency of the smoothed estimates are improved.

Comparing the performance of the CC and PS methods with LS BC update (Figs. 2.9(a) and 2.9(b)), the error plots are qualitatively similar. The difference between CC and PS methods are: $-7.525e^{-6} \pm 0.0009$ m using five mixands, and $-7.227e^{-7} \pm 0.0015$ m using seven mixands, indicating both methods perform similarly.

The smoothing performance (i.e. accuracy of the smoothed trajectory compared to the truth) is greatly affected by the location and sparseness of the measurements. For example, if no ARTag and associated measurement are available on the right side of the environment, the robot navigates the environment for about 4 minutes before problems arise. With no measurements, the initial prediction of the forward filter quickly becomes inconsistent and inaccurate; this is primarily a result of IMU biases, which are not updated without inertial measurements. Therefore, the performance of the smoothing with different ARTag locations are studied. The number of ARTags is chosen to be one of $\{3, 5, 10, 15, 21\}$ tags. For each number of ARTags and each number of maximum mixands, a total of 20 trials are run using the same robot path, with random selections of ARTags for each trial. For the case of 21 tags, only one trial is completed because all 21 tags are used. Figure 2.10 shows the statistics of the 2D location errors over the full time of each run, for both forward and smoothed estimates as a function of the number of ARTags in the environment. For reference, the line in the box represents the median, and the edges (of the boxes) represent 25% and 75% percentiles. The whiskers extend to the most extreme

data which are not considered to be outliers. Outliers are individually plotted. For each number of ARTags used, three boxes indicate no splitting, maximum of five mixands, and maximum of seven mixands cases, respectively from the left.

The backward smoother estimates (Fig. 2.10 right) have lower errors on average and tighter distributions (small distance between edges), compared to the forward estimate errors (Fig. 2.10 left). Also, the forward estimates include a large number of outliers even when the splitting technique is used. On the other hand, the distances between the edges of the boxes of the smoothed estimates are much smaller than those of the forward estimates, indicating significant reduction of the variance of the errors. The number and range of outliers are significantly smaller as well, especially as the number of ARTags used increases. As the number of mixands increases, the variance of the error also decreases, indicating that smoothing using Gaussian mixtures performs better than the single Gaussian smoothing (a Kalman smoother) by capturing non-Gaussian densities at key times of the run. Note that the number of outliers and the variance are the highest for each case when no splitting occurs. This is because the single Gaussian representation cannot sufficiently capture the propagation of the distribution through the nonlinearity.

As the number of ARTags used increases, the performance becomes better; this intuitively makes sense given the reliance on additional sensor data. In general, using a larger number of ARTags and mixands yields improved performance, typically at the expense of increased computation. In this experiment, the performance of smoothing using five or seven mixands becomes indifferent when at least 10 ARTags are used; increasing the number of ARTags yields only

slightly better performance as shown in Fig. 2.10.

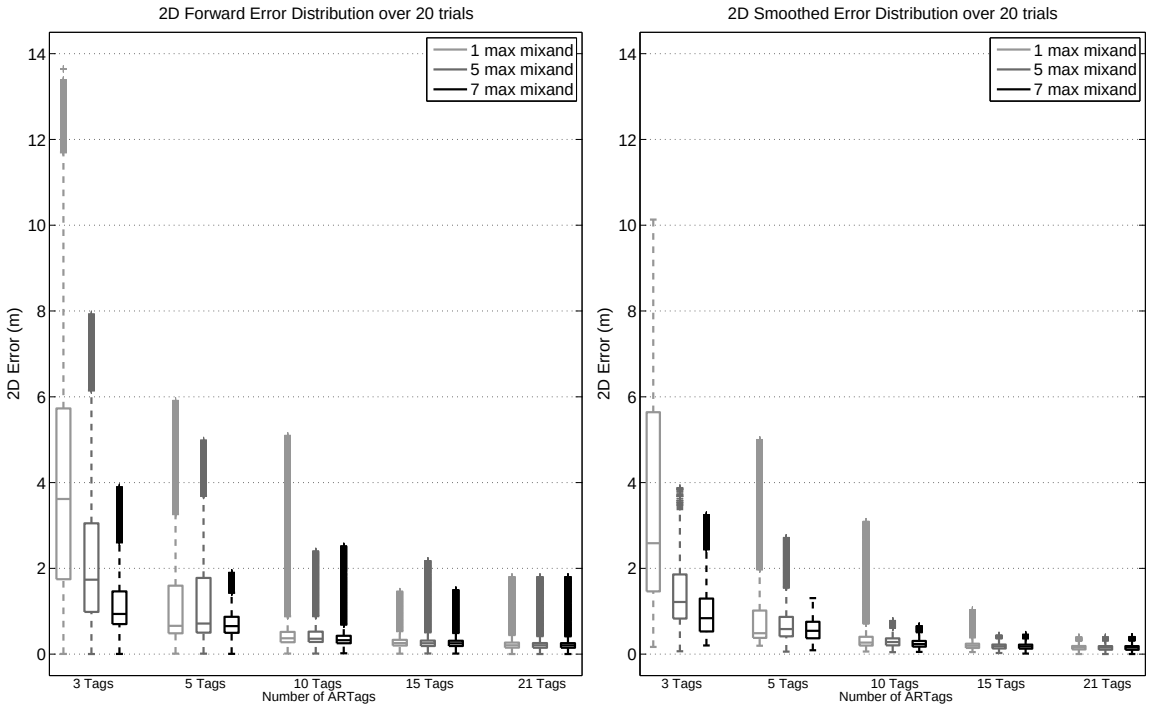


Figure 2.10: Statistics of 2D location errors for 20 trial runs (10 each for the CC and PS methods with LS BC update) represented by boxplots.

2.6 Conclusion

In this paper, novel smoothing methods using adaptive Gaussian mixture models for nonlinear systems are presented. A fully linearized version of the forward-backward smoothing algorithm is developed which is the theoretical optimal, but is computationally challenging due to the recursive steps that require continuous re-linearization. A batch least squares form of the backward corrector estimation is proposed to overcome these computational challenges, yet achieve similar performance. In addition, using Gaussian splitting and

condensation methods, along with an association tree, an adaptive version of the Gaussian mixture smoother is developed which addresses a wider range of problems where the non-Gaussianity of the distribution growth can be addressed. Simulation results on a highly nonlinear system and one measurement show that the least squares backward corrector update works well for shorter time smoothing windows, with a very large savings in computation. Experimental results for an indoor navigation problem and varying numbers of localization features show that performance is very high when the maximum numbers of mixands is greater, and the evolution of the distribution through the nonlinear dynamics is well captured during the smoothing process.

CHAPTER 3

SCALABLE PROBABILISTIC MULTI-RESOLUTION SURFACE NORMAL ESTIMATION

3.1 Introduction

In modern field robotics, perception tasks range from low-level processing of data into consistent and accurate estimates of information (robot's states, terrain height) to high-level tasks such as understanding the environments (e.g. object recognition). The recent availability of inexpensive high-resolution 3D scanning devices (e.g. Microsoft Kinect, Intel RealSense) enable not only the easy collection of a large amount of high-quality 3D data about the environment, but also the active manipulation and interaction with the environment. Given the large quantity of data, typically, one of the key low-level algorithms in robotics is processing the raw data into a form for which it can be used for higher level algorithms such as shape estimation, object recognition, or classification.

Consider the task of object recognition using a shape; many recent algorithms have been developed using descriptors such as 3D points. Johnson and Hebert present the Spin Image descriptor which computes the histogram of points representing radial distance between the keypoint and the neighborhood [29]. Variants of Spin Image have evolved, such as the face-based spin image [10], multi-resolution spin image [14], scale invariant spin image [11], and color spin image [54]. Similarly, a large number of descriptors have been developed that encode the relationship between neighboring points and the tangential plane's normal vector [67, 68, 42, 44, 65, 66]. Many of these descriptors used in the detection/recognition tasks, require the surface normal information

to characterize a feature.

The most popular surface normal estimation algorithms use 3D point clouds. Klasing *et al*. [35] remark and study several optimization and averaging methods to compute normal vectors, including *PlanePCA*, *PlaneSVD*, *QuadSVD*. Jordan and Mordohai extend these methods by updating the anchoring reference points, normalizing the normal vectors, and weighing neighboring points differently [30]. Liu *et al*. present a surface normal estimation method using sparse tensor voting [43], which allows tuning of voting kernel size for flexible structure extraction. Holzer *et al*. present an estimation method using integral images [74] and corresponding covariance matrices; an adaptive neighborhood size using integral images for more efficient and robust covariance estimation is presented in [27].

While these algorithms show robustness and good performance in surface normal estimation, most use point clouds as their base representation of 3D space. Point clouds, however, scale with the number of measurements, not the environment or object size. The number of points can increase over time due to the ease of collecting sensor data leading to computational challenges. The memory required to store the points increases, as well as the computation/processing for subsequent tasks, such as object recognition. Thus, scaling becomes a limiting factor, especially for small robotics platform with limited computational resources.

In a related area, multi-resolution mapping algorithms have been developed, as well as companion planning algorithms that utilize the map. Unlike uniformly distributed discretized space, multi-resolution representation combine nearby space with similar information. Importantly, the maps and associated

planning space scale well with the environment, not the sensor data, leading to computationally efficient algorithms for real time implementation on robots. Montemerlo and Thrun solve a robot navigability problem with limited resolution of terrain grids and sensor measurements [49] by using multiple maps in different resolutions with updates based on measurement ranges. Kraetzschmar *et al.* present a probabilistic quad-tree for efficient storage and mapping of large environment [37]. Furuyama and Niisuma utilize a quad-tree structure that adaptively changes the resolution based on human motion frequency for robot path planning [22]. Ryde and Hu present multi-resolution occupied voxel lists that are used for map alignment, and successfully create 3D maps in large scale [63]. Khan *et al.* relax the cubic grid cell assumptions and present an adaptive rectangular cuboid which utilizes an incremental fusion process that adapts the grid size based on the occupancy probabilities [33].

In this paper, an efficient scalable probabilistic multi-resolution surface normal estimation framework is presented, capable of running for long periods of time using a variety of sensors, on small robotics platforms. The proposed method discretizes 3D space into a 3D occupancy grid, and utilizes occupancy information of surrounding cells to probabilistically infer the surface normal vectors. As the space is discretized, the computational need for storage is significantly reduced compared to those of using point clouds, as shown in other related works; surface normals are computed much faster given the same amount of sensor data. Both recursive and batch formulations are derived to solve the surface normal estimation problem. In addition, the capability to represent the space in multi-resolution enables users to choose the necessary amount of information to use for a given task, or adapt areas of the scene that do not require the complex representation (e.g. a flat wall or floor). Another advantage of the

proposed algorithm is that many other estimation and planning algorithms typically use occupancy grids, and thus the approach is both complementary and potentially computationally free.

This paper builds on the authors' previous work which provides the initial development of the surface normal estimation framework [40]. Extensions include: 1) capability to estimate surface normal vectors in multiple different resolutions, 2) improvement on the computational efficiency while maintaining the accuracy and consistency of the estimates for multi-resolution, 3) extensive numerical comparison study between the theoretical model and efficient implementations, 4) extended experiments and analysis to study the multi-resolution estimation performance.

This paper is organized as follows: Section 3.2 presents the problem setup and formulation, followed by the proposed the surface normal estimation methods in Section 3.3. The multi-resolution surface normal estimation is presented in Section 3.4. Section 3.5 presents a likelihood model to capture both the occupancy and surface in the grid representation. Section 3.6 studies real-time the performance of the surface normal estimation algorithm. Simulation and experimental results are shown in Section 3.7.

3.2 Problem Statement

The surface normal estimation problem is captured in the probabilistic graphical model shown in Fig. 3.1. The x variable represents the surface normal state of a single cell, while o^i and z^i_j represent i-th neighboring cell's occupancy and a measurement for i-th cell at time j, respectively. The proposed framework is

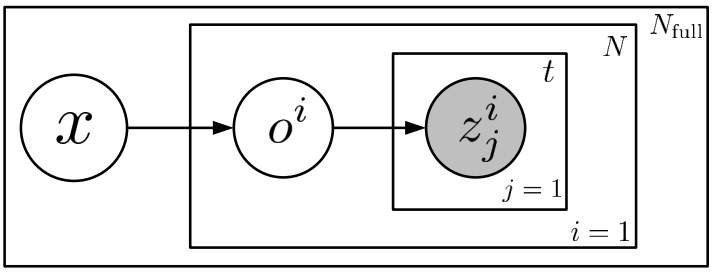


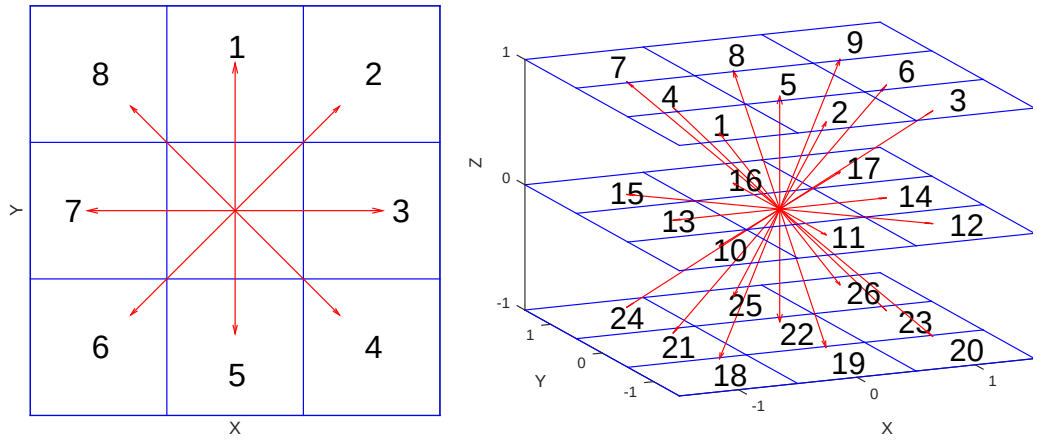
Figure 3.1: Probabilistic graphical model for the surface normal estimation problem. Variable x represents the surface normal state of the cell of interest, o^i represents the occupancy state of i-th neighboring cell, and z_j^i represents a measurement for i-th neighboring cell at time j. Variables N represents the number of neighboring grid cells, and N_{full} represents the number of grid cells in the full space.

general to both 2D and 3D space.

Given K possible surface normals, the surface normal state x uses 1-of-K representation,

$$x = [0, \cdots, 1, \cdots, 0]^{\mathsf{T}} \tag{3.1}$$

where each element is a binary variable, $x(i) \in \{0, 1\}$, and $\sum_i^K x(i) = 1$. The occupancy of each cell $o^{()}$ is a binary variable, $o^{()} \in \{0, 1\}$, where 0 and 1 represent the cell being empty or occupied, respectively. The K discrete options for surface normals can be defined in many ways, without loss of generality. Here, surface normals are selected based on the index of grid cells, drawn through the center of each. Fig. 3.2 visually shows all possible states for both 2D and 3D cases. The measurements $\mathcal{Z}_t = \{z_t^1, \cdots, z_t^N\}$ are binary variables, $z_t^{()} \in \{0, 1\}$, where 1 indicates that the measurement falls within the cell, and 0 indicates that the measurement provides information that the cell is empty at time t. The variable N represents the number of neighboring cells. The environment is assumed to



(a) 2D state visualization (b) 3D state visualization

Figure 3.2: Surface normal vector directions corresponding to x value. Each side of cell/voxel is in equal length.

be static, and thus the surface normal state x and occupancy, $O = \{o^1, \cdots o^N\}$, do not have the time subscript whereas the measurements $\mathcal{Z}_0$ do. Note that the model in Fig. 3.1 is defined for one cell, and thus is repeated for each cell in the full space, where N_{full} cells reside. The surface normal vector in 2D and 3D Cartesian space, $\overrightarrow{n}$, is obtained using the unit vectors for each state element, $\overrightarrow{n_i}$, as shown in Fig. 3.2.

$$\overrightarrow{n} = \sum_{i=1}^{K} p(x(i) = 1) \cdot \overrightarrow{n_i} \tag{3.2}$$

The goal of the surface estimation problem is to obtain the posterior distribution of the surface normal state x given all available measurements. Ideally, the full joint distribution of all normal vectors for all N_{full} grid cells in the full space are estimated simultaneously. Intuitively, this makes sense because neighboring cells' surface normal information is coupled. A Markov Random Field model would allow this type of problem to be cast [73]. However, this problem does not scale well and is computationally intractable. The traditional occupancy grid formulation assumes each cell to be independent [71]; this enables a fast,

real-time implementation to be realized, at the cost of a slightly conservative estimate. The approach to the surface normal estimation developed here takes a similar approach. Thus, the key problem to solve is the posterior distribution, $p(x|\mathcal{Z}_{1:t}) = p(x|\mathcal{Z}_1, \cdots, \mathcal{Z}_t)$.

3.3 Surface Normal Estimation

The posterior distribution of surface normal state, $p(x|\mathcal{Z}_{1:t})$, can be computed in two different ways: 1) recursively, and 2) in batch. Both methods are derived here, and shown to yield identical results theoretically. Subsequent sections then describe advantages/disadvantages and uses for each formulation.

3.3.1 Recursive Formulation

Since both x and O are latent variables (see Fig. 3.1), obtaining the posterior distribution of the state, $p(x|\mathcal{Z}_{1:t})$, requires marginalization over the occupancy O variables, written as:

$$p(x|\mathcal{Z}_{1:t}) = \sum_{O} p(x, O|\mathcal{Z}_{1:t}) \tag{3.3}$$

$$= \sum_{O} \frac{p(x) \ \prod_{i=1}^{N} p(o^i|x) \ \prod_{j=1}^{t} p(z_j^i|o^i)}{p(\mathcal{Z}_{1:t})} \tag{3.4}$$

where

$$p(\mathcal{Z}_{1:t}) = \sum_{x} \sum_{O} p(x) \ \prod_{i=1}^{N} p(o^i|x) \ \prod_{j=1}^{t} p(z_j^i|o^i)$$

Solving (3.4) is challenging, as the computation scales with time t. Therefore, a recursive solution is developed.

The posterior distribution at the next time step $t+1$, $p(x|\mathcal{Z}_{1:t+1})$, is computed by marginalizing over the occupancy cells, O:

$$p(x|\mathcal{Z}_{1:t+1}) = \sum_{O} p(x, O|\mathcal{Z}_{1:t+1}) \tag{3.5}$$

The joint distribution for time $t+1$, $p(x, O|\mathcal{Z}_{1:t+1})$, can be simplified using the fact that all measurements $\mathcal{Z}_{()}$ and the state x are conditionally independent given O (see Fig. 3.1).

$$\begin{aligned} p(x, O|\mathcal{Z}_{1:t+1}) &= p(x, O|\mathcal{Z}_{1:t}, \mathcal{Z}_{t+1}) && (3.6)\\ &= \frac{p(x, O, \mathcal{Z}_{t+1}|\mathcal{Z}_{1:t})}{p(\mathcal{Z}_{t+1}|\mathcal{Z}_{1:t})} && (3.7)\\ &= \frac{p(\mathcal{Z}_{t+1}|x, O, \mathcal{Z}_{1:t}) \cdot p(x, O|\mathcal{Z}_{1:t})}{p(\mathcal{Z}_{t+1}|\mathcal{Z}_{1:t})} && (3.8)\\ &\propto p(\mathcal{Z}_{t+1}|O) \cdot p(x, O|\mathcal{Z}_{1:t}) && (3.9) \end{aligned}$$

Equation (3.9) shows the recursive solution for the joint distribution $p(x, O|\mathcal{Z}_{1:t+1})$ at time $t+1$ based on the previous estimate at time t, $p(x, O|\mathcal{Z}_{1:t})$. Substituting (3.9) into (3.5) then yields the posterior distribution,

$$p(x|\mathcal{Z}_{1:t+1}) \propto \sum_{O} p(\mathcal{Z}_{t+1}|O) \cdot p(x, O|\mathcal{Z}_{1:t}) \tag{3.10}$$

3.3.2 Batch Formulation using Occupancy Grid

Consider now the specific case where O is a set of N grid cells. Noting that x and $\mathcal{Z}_{()}$ are conditionally independent given O, the posterior distribution can be

simplified as following:

$$p(x|\mathcal{Z}_{1:t}) = \sum_{O} p(x, O|\mathcal{Z}_{1:t}) \tag{3.11}$$

$$= \sum_{O} \frac{p(O|x) \cdot p(x)}{p(O)} \cdot p(O|\mathcal{Z}_{1:t}) \tag{3.12}$$

$$= p(x) \cdot \sum_{o^1} \cdots \sum_{o^N} \frac{p(o^1, \cdots, o^N|x)}{p(o^1, \cdots, o^N)} \cdot p(o^1, \cdots, o^N|z^1_{1:t}, \cdots, z^N_{1:t}) \tag{3.13}$$

where $z^i_{1:t} = \{z^i_1, \cdots, z^i_t\}$ represents all measurements up to time t for i-th neighboring cell. Equation (3.12) and (3.13) show that the posterior distribution $p(x|\mathcal{Z}_{1:t})$ can be obtained at any time needed, t, so long as the posterior distribution of neighboring cells' occupancy, $p(O|\mathcal{Z}_{1:t})$, are maintained. And, as previously discussed, maintaining an occupancy grid online is very common for robots.

While (3.13) gives a full solution for estimating the surface normal vectors using an occupancy grid, the posterior distribution of occupancy, $p(o^1, \cdots, o^N|z^1_{1:t}, \cdots, z^N_{1:t})$ is unfortunately computationally expensive to compute. This posterior can be expanded as:

$$p(o^1, \cdots, o^N|z^1_{1:t}, \cdots, z^N_{1:t}) = \frac{p(o^1, \cdots, o^N) \prod_{j=1}^{t} \prod_{i=1}^{N} p(z^i_j|o^i)}{p(z^1_{1:t}, \cdots, z^N_{1:t})} \tag{3.14}$$

where

$$p(o^1, \cdots, o^N) = \sum_{x} p(x) \prod_{i=1}^{N} p(o^i|x) \tag{3.15}$$

Substituting (3.14) into (3.13) yields:

$$p(x|\mathcal{Z}_{1:t}) = p(x) \cdot \sum_{o^1} \cdots \sum_{o^N} \frac{p(o^1, \cdots, o^N|x) \prod_{j=1}^{t} \prod_{i=1}^{N} p(z^i_j|o^i)}{p(z^1_{1:t}, \cdots, z^N_{1:t})} \tag{3.16}$$

Since $p(z^i_{1:t}, \cdots, z^N_{1:t})$ is a normalization constant, and the occupancy grid cells are also conditionally independent given the surface normal x, or $p(o^1, \cdots, o^N|x) = \prod_{i=1}^{N} p(o^i|x)$ (see Fig. 3.1), the posterior distribution of the surface normal state can be simplified to be:

$$p(x|\mathcal{Z}_{1:t}) \propto p(x) \sum_{o^1} \cdots \sum_{o^N} \prod_{i=1}^{N} p(o^i|x) \cdot \prod_{j=1}^{t} p(z^i_j|o^i) \tag{3.17}$$

Note that (3.17) is theoretically equivalent to (3.10).

3.3.3 Sufficient Measurements

Consider now the posterior distribution of occupancy of i-th neighboring cell:

$$p(o^i|z^i_{1:t}) = \frac{p(o^i) \prod_{j=1}^{t} p(z^i_j|o^i)}{p(z^i_{1:t})} \propto p(o^i) \prod_{j=1}^{t} p(z^i_j|o^i) \tag{3.18}$$

Dividing by $p(o^i)$ then yields $\prod_{j=1}^{t} p(z^i_j|o^i) \propto \frac{p(o^i|z^i_{1:t})}{p(o^i)}$, which can be substituted into (3.17) to yield the posterior distribution of the surface normal state x as:

$$p(x|\mathcal{Z}_{1:t}) \propto p(x) \sum_{o^1} \cdots \sum_{o^N} \prod_{i=1}^{N} p(o^i|x) \frac{p(o^i|z^i_{1:t})}{p(o^i)} \tag{3.19}$$

The posterior in (3.19) now uses the posterior distribution of occupancy directly, $p(o^i|z^i_{1:t})$, along with the prior $p(o^i)$. Using the model presented in Fig. 3.1, the prior $p(o^i)$ is expressed as $p(o^i) = \sum_x p(x) \cdot p(o^i|x)$. In the traditional occupancy grid formulation, $p(o^i)$ is uniform for all cells [71]. If a traditional occupancy grid algorithm is used to estimate the posterior occupancy, $p(o^i|z^i_{1:t})$, the prior distribution, $p(o^i)$, could differ from the model presented in Fig. 3.1. Equation (3.18) suggests, however, that the posterior distribution of an occupancy grid algorithm converges if sufficient measurements are available such

that $\prod_{j=1}^{t} p(z_j^i|o^i)/p(z_{1:t}^i) \gg p(o^i)/p(z_{1:t}^i)$; or $p(o^i|z_{1:t}^i) \approx \prod_{j=1}^{t} p(z_j^i|o^i)$ for a large time t. Therefore, if sufficient number of measurements are given, the posterior distribution of the surface normal state x can be simplified to be:

$$p(x|\mathcal{Z}_{1:t}) \propto p(x) \sum_{o^1} \cdots \sum_{o^N} \prod_{i=1}^{N} p(o^i|x) \cdot p(o^i|z_{1:t}^i) \tag{3.20}$$

$$= p(x) \cdot \prod_{i=1}^{N} \sum_{o^i \in \{0,1\}} p(o^i|x) \cdot p(o^i|z_{1:t}^i) \tag{3.21}$$

A numerical study to validate this condition is presented in Section 3.6.

Equation (3.21) presents a scalable solution to estimate the surface normal by significantly reducing the amount of memory required. The memory required to estimate the joint distribution of all neighboring cells' occupancy, $p(o^1, \cdots, o^N)$, in (3.15) increases exponentially with N. For example, $2^9 = 512$ floating-point numbers are required for 9 cells in 2D, and $2^{27} = 134,217,728$ numbers with 27 cells in 3D. In addition, the marginalization over all cells $O = \{o^1, \cdots, o^N\}$ is also computationally challenging (see (3.13)) as the number of iteration exponentially scales with N as well. However, using (3.21) with a sufficient number of measurements produces a posterior distribution very efficiently and quickly, with the required memory reduced from 2^N to $2N$.

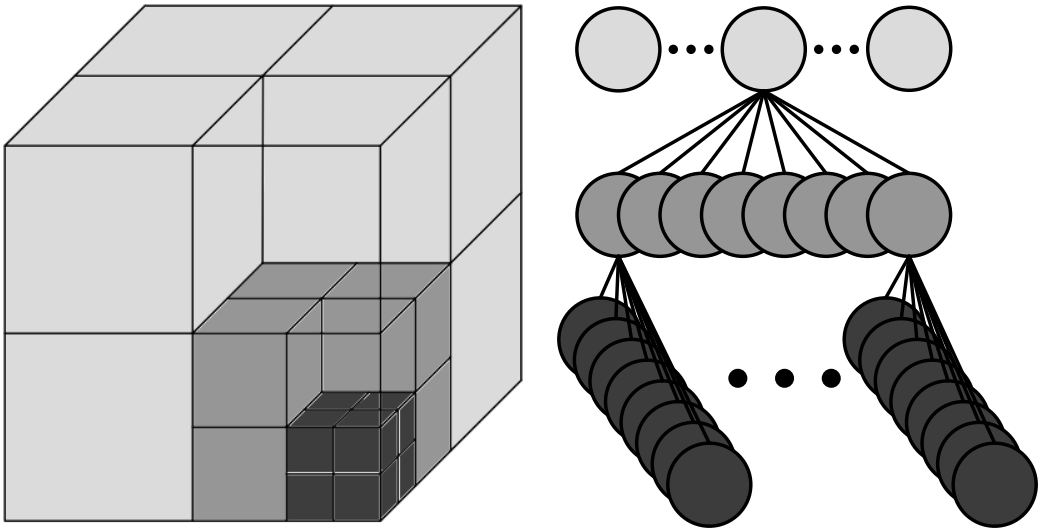
The problem of surface normal estimation can now be defined with two components: 1) maintaining the posterior distribution of neighboring cells' occupancy over time, which can be solved using one of any number of open source libraries (see Ref. [71], [28], [59], for example) and 2) estimating surface normal vectors when needed using (3.21). The latter requires a measurement model between the occupancy grid and the surface normals which is shown in Section 3.5.

3.4 Multi-resolution Surface Normal Estimation

Tasks performed by robots typically vary in the level of environmental information required. For example, for a robot to navigate a relatively uncluttered and non-complex environment, coarse spatial information is sufficient. More complex tasks, however, such as picking up a bottle of water or an egg out of a refrigerator requires higher resolution spatial information in order to plan an accurate motion to avoid objects and grasp the object. Therefore, a multi-resolution estimation of the spatial information enables a robot to adaptively adjust the estimate and subsequent computational resources to meet the task requirements successfully.

Octomap [28] is used in this framework to provide the requisite multi-resolution occupancy grid, although any tree-structure-based occupancy grid mapping algorithm can be employed without loss of generality. Octomap utilizes octree [46] which divides a 3D cubic space into 8 equally sized sub-spaces and registers them as children nodes. Each node in Octomap represents the posterior estimate of occupancy. Therefore, each depth of the tree represents the 3D occupancy grid in a different resolution. As the depth increases, the resolution increases and the number of grid cells increases as well. Figure 3.3 shows an example of 3D space represented using Octomap. Each cube in the 3D space (Fig. 3.3(a)) can be divided into eight cubes (shown in a darker color); each child cube can be further divided (shown in the darkest color) to increase resolution. In this paper, 'higher depth' level or deeper into the hierarchical octree indicates higher resolution, and higher resolution indicates smaller grid cell.

Two methods to estimate the surface normal vectors for different resolutions



(a) Octomap spatial representation (b) Octree structure

Figure 3.3: Example of a 3D space represented using Octomap, with shading between (a) and (b) implying correlation. Lower right cube in (a) is shown for multiple depths of the tree. (b) shows the octree structure for corresponding space where each node/circle denotes an occupancy state. Each node can be decomposed into eight children nodes to increase resolution.

are proposed. Figure 3.4 shows the probabilistic graphical models for MR1 and MR2 algorithms. The first method (MR1) estimates the surface normal state for each parent node in the model, $p(x_0^{k-1,0}|\mathcal{Z}_{1:t})$, by marginalizing over the surface normal state of M children nodes, $p(x_0^{k,0}|\mathcal{Z}_{1:t})$. The first superscript of $x_0^{0,0}$ and $o_0^{0,0}$ indicates the tree depth level, and the second indicates the index of the parent node (at one lower depth level). The subscript indicates the index of the current node. The second method (MR2) uses a multi-resolution occupancy grid, $p(o_0^{0,0}|\mathcal{Z}_{1:t})$, such as Octomap, and estimates the surface normal state, $p(x_0^{0,0}|\mathcal{Z}_{1:t})$, for each individual resolution. For MR2, the posterior estimate of the occupancy at depth $k-1$ can be recursively computed using the

posterior estimate of the children nodes at depth k.

$$p(o_\alpha^{k-1,()}|\mathcal{Z}_{1:t}) = \sum_{o_1^{k,\alpha}} \cdots \sum_{o_M^{k,\alpha}} p(o_\alpha^{k-1,()}, o_1^{k,\alpha}, \cdots, o_M^{k,\alpha}|\mathcal{Z}_{1:t}) \tag{3.22}$$

$$= \sum_{o_1^{k,\alpha}} \cdots \sum_{o_M^{k,\alpha}} p(o_\alpha^{k-1,()}|o_1^{k,\alpha}, \cdots, o_M^{k,\alpha}) \cdot p(o_1^{k,\alpha}, \cdots, o_M^{k,\alpha}|\mathcal{Z}_{1:t}) \tag{3.23}$$

$$= \sum_{o_1^{k,\alpha}} \cdots \sum_{o_M^{k,\alpha}} p(o_\alpha^{k-1,()}|o_1^{k,\alpha}, \cdots, o_M^{k,\alpha}) \cdot \prod_{i=1}^{M} p(o_i^{k,\alpha}|z_{1:t}^i) \tag{3.24}$$

where

$$p(o_\alpha^{k-1,()}|o_1^{k,\alpha}, \cdots, o_M^{k,\alpha}) = \frac{\sum_{i=1}^{M} \delta(o_i^{k,\alpha})}{M} \tag{3.25}$$

$$\delta(o_i^{k,\alpha}) \triangleq \begin{cases} 0 & o_i^{k,\alpha} = 0 \\ 1 & o_i^{k,\alpha} = 1 \end{cases} \tag{3.26}$$

where $\delta()$ is Kronecker delta function. Note that given the occupancy states at depth k, $\{o_1^{k,\alpha}, \cdots, o_M^{k,\alpha}\}$, the occupancy state at depth $k-1$ and the measurements $\mathcal{Z}_{1:t}$ are conditionally independent. Equation (3.25) and (3.26) enable posterior occupancy state estimation via computing average. If more conservative method is desired, (3.27) can be used.

$$p(o_\alpha^{k-1,()}|o_1^{k,\alpha}, \cdots, o_M^{k,\alpha}) = \begin{cases} 0 & o_i^{k,\alpha} \neq \max(o_1^{k,\alpha}, \cdots, o_M^{k,\alpha}) \\ 1 & o_i^{k,\alpha} = \max(o_1^{k,\alpha}, \cdots, o_M^{k,\alpha}) \end{cases} \tag{3.27}$$

The estimation of posterior surface normal state, $p(x_\alpha^{k-1,()}|\mathcal{Z}_{1:t})$, is similarly obtained using (3.21) as following:

$$p(x_\alpha^{k-1,()}|\mathcal{Z}_{1:t}) \propto p(x_\alpha^{k-1,()}) \prod_{i=1}^{N} \sum_{o_i^{k-1,\alpha} \in \{0,1\}} p(o_i^{k-1,\alpha}|x) \cdot p(o_i^{k-1,\alpha}|z_{1:t}^i) \tag{3.28}$$

Similarly, for MR1, the posterior surface normal state can be recursively es-

timated (see Fig. 3.4(a)).

$$p(x_\alpha^{k-1,()}|\mathcal{Z}_{1:t}) = \sum_{x_1^{k,\alpha}} \cdots \sum_{x_M^{k,\alpha}} p(x_\alpha^{k-1,()}, x_1^{k,\alpha}, \cdots, x_M^{k,\alpha}|\mathcal{Z}_{1:t}) \tag{3.29}$$

$$= \sum_{x_1^{k,\alpha}} \cdots \sum_{x_M^{k,\alpha}} p(x_\alpha^{k-1,()}|x_1^{k,\alpha}, \cdots, x_M^{k,\alpha}) \cdot p(x_1^{k,\alpha}, \cdots, x_M^{k,\alpha}|\mathcal{Z}_{1:t}) \tag{3.30}$$

$$= \sum_{x_1^{k,\alpha}} \cdots \sum_{x_M^{k,\alpha}} p(x_\alpha^{k-1,()}|x_1^{k,\alpha}, \cdots, x_M^{k,\alpha}) \prod_{i=1}^{M} p(x_i^{k,\alpha}|z_{1:t}^i) \tag{3.31}$$

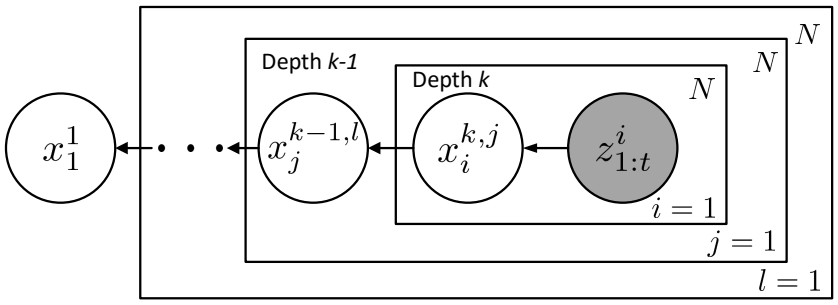
where,

$$p(x_\alpha^{k-1,()}(j) = 1|x_1^{k,\alpha}, \cdots, x_M^{k,\alpha}) = \frac{\sum_{i=1}^{M} \delta(x_i^{k,\alpha}(j))}{M} \tag{3.32}$$

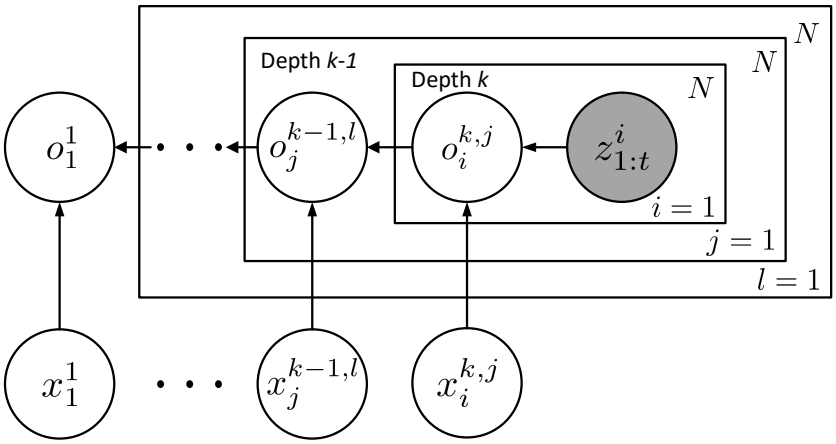
$$\delta(x_i^{k,\alpha}(j)) \triangleq \begin{cases} 0 & x_i^{k,\alpha}(j) = 0 \\ 1 & x_i^{k,\alpha}(j) = 1 \end{cases} \tag{3.33}$$

Note that given the surface normal states at depth k, $\{x_1^{k,\alpha}, \cdots, x_M^{k,\alpha}\}$, the surface normal state at depth $k-1$ and the measurements $\mathcal{Z}_{1:t}$ are conditionally independent.

The two methods, MR1 and MR2, operate on different information of course: MR1 on surface normals, and MR2 on occupancy grid. Both methods also perform on averaging operation – MR1 averages in the surface normals and MR2 averages in the octree. These averaging operations tend to lose information, which intuitively makes sense; lower resolution models have less information than higher resolution models. The results section compares and discusses both multi-resolution surface normal estimation approaches.



(a) MR1



(b) MR2

Figure 3.4: Probabilistic graphical models for hierarchical multi-resolution surface normal estimation; few depth levels are shown. Variable k and $k-1$ denote the depth levels, $\mathcal{Z}^i$ represents all available measurements for $o_i^{k,j}$ cell, and N represents the number of neighboring cells. $x_0^{0,0}$ variable represents the surface normal estimate. Subscript denotes the index of the cell at current depth. Superscripts denote the related (parent) cell in the lower depth; first superscript denotes the depth level, while the second one denotes the index of the parent node.

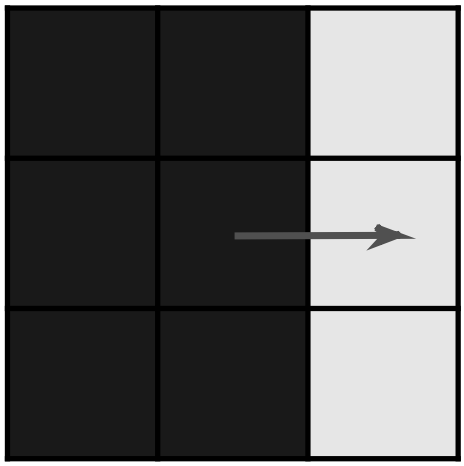
3.5 Occupancy Grid Likelihood Model

The likelihood model, $p(O|x) = p(o^1, \cdots, o^N|x)$, plays an important role in the proposed estimation framework as it provides a model for connecting the neighboring cells' occupancy to the direction of the normal vector. Intuitively, the likelihood model represents the likelihood of each neighboring cells being occupied given a normal vector direction. Figure 3.5 shows four examples of the likelihood model for 2D and 3D cases. Color (in grayscale) in each cell represents the likelihood of the cell being occupied; darker color indicates higher likelihood. The index of x in each example follows the indexing convention shown in Fig. 3.2.

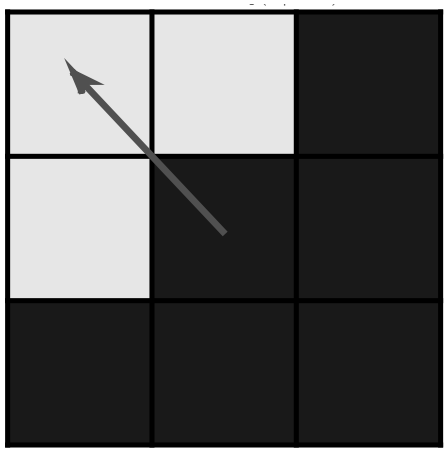
The likelihood models are defined by considering both positive (being occupied) and negative (being empty) cells, which enables both a surface normal vector and the correct direction to be considered simultaneously. This is an advantage over point clouds where viewpoint ambiguity is a challenge to solve without heuristics. As shown in the examples, the surface normal vector of the likelihood models originates in the center, and points out of cells with low occupancy.

3.6 Numerical study of the Effect of Sufficient Measurements

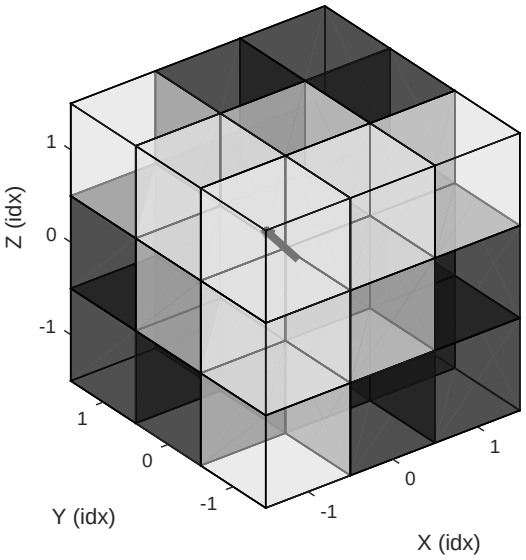
This section presents numerical results to study the effects of assuming a sufficient number of measurements (eq. (3.21)) on the surface normal estimation performance, as compared to the full solution. A variety of environments are compared to understand the performance and computation trade between the



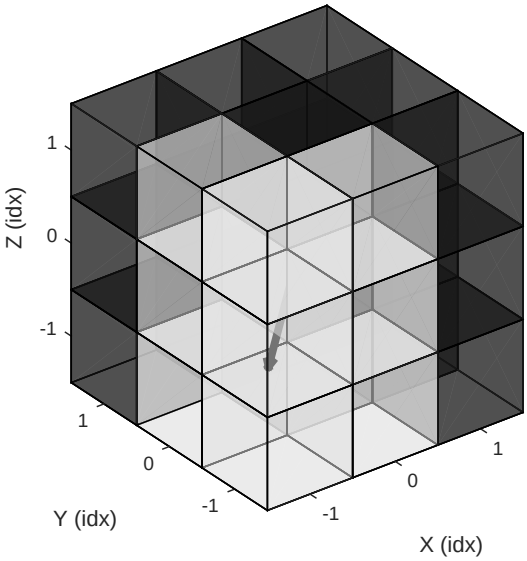
(a) Likelihood $p(O|x(3) = 1)$. Surface normal towards right.



(b) Likelihood $p(O|x(8) = 1)$. Surface normal towards left top.



(c) Likelihood $p(O|x(1) = 1)$. Surface normal towards (-1, -1, 1)



(d) Likelihood $p(O|x(10) = 1)$. Surface normal towards (-1, -1, 0)

Figure 3.5: Examples of surface normal likelihood model for 2D and 3D case. Arrows are drawn to show the surface normal direction

full and scalable solutions.

3.6.1 State Estimation Comparison

Fig. 3.6 shows seven different surfaces varying over 3×3 2D grid with 0.1m resolution. Total of 24 sensor locations were surveyed each in a circle with radius of 0.3m about the center of the middle cell. Each measurement consists of seven laser rays in a 60 degree of field of view with 10 degree of angular resolution. The measurement likelihood, $p(z^{()} = 1|o^{()} = 1) = 0.7$.

In this study, two variants of the surface estimation methods are compared: 1) the full solution (F) using (3.13) where $p(o^1, \cdots, o^N)$ is computed using (3.15); and 2) the scalable solution (S) using (3.21), where the prior distribution $p(o^1, \cdots, o^N)$ is uniform. The posterior estimates of occupancy are computed using a Bayes filter [71] as other occupancy grid mapping algorithms use.

Both performance and computation are compared for each of the variants of the surface normal estimation algorithm. Given that the full solution is available, performance is defined as the Kullback-Leibler Divergence (KLD) between the full distribution (F) and the scalable solutions (S). The KLD is the relative entropy between two probability distributions defined as:

$$D_{\mathrm{KL}} = \sum_{i=1}^{K} p_F(x(i) = 1) \log \frac{p_F(x(i) = 1)}{p_S(x(i) = 1)}$$

where p_F and p_S represent the posterior probability distribution of the full solution (F) and the scalable solution (S), respectively. Therefore, a smaller value of KLD indicates p_S is more consistent with p_F. A total of 1000 trials are run for each surface model, each with Gaussian white noise added on both range and

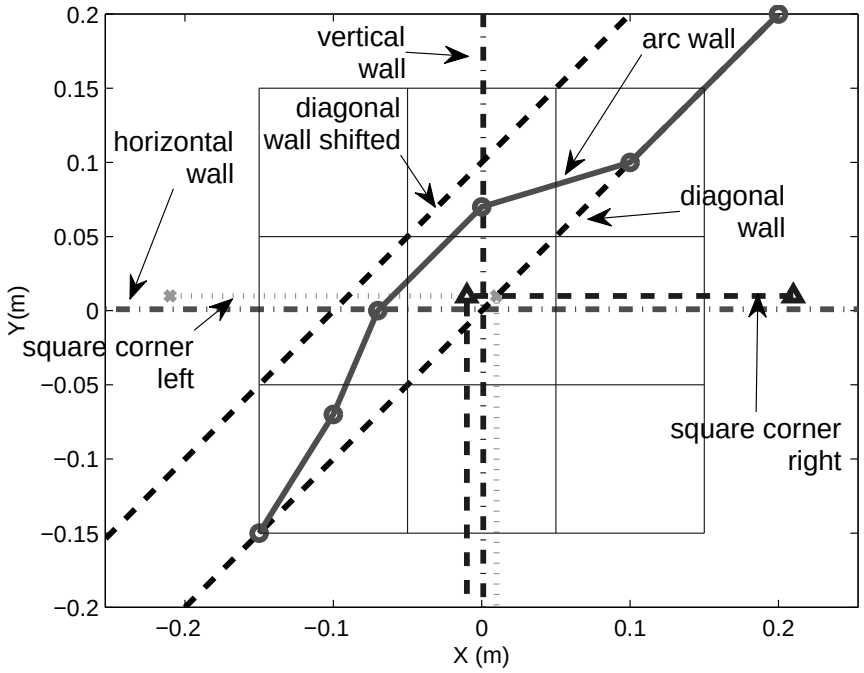


Figure 3.6: Seven different surfaces for numerical study

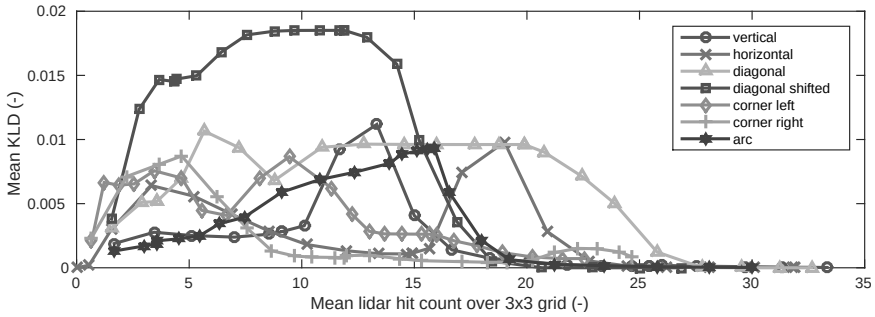


Figure 3.7: KLD mean as a function the average number of LIDAR hits over the 3×3 grid. The sensor noise is $\sigma_r = 0.05$m, $\sigma_\theta = 5$ deg.

bearing sensor readings: $\sigma_r = 0.05$m, $\sigma_\theta = 5$ deg.

Figure 3.7 shows the mean KLD over 1000 runs for scalable solution S with the sensor noise; each line corresponds to different surface shape. The KLD values are presented as a function of the average number of LIDAR hits over the 3×3 grid. This plot provides intuition of how much information (number of

measurements in this case) is required in order to obtain consistent estimates. The KLD values drop and converge when 25 LIDAR hits/misses are collected. Therefore, for surface normal estimation problem in 2D space, approximately 25 LIDAR measurements are required to have consistent estimates of the occupancy grid, and to obtain consistent and accurate normal estimates, compared to the full solution (F).

These results indicate that the posterior distributions of the occupancy grid of the full solution F and the scalable solution S become nearly identical after gathering sufficient measurements. This makes intuitive sense because, theoretically, as more measurements are taken, the occupancy grid converges to the true state. Therefore, the posterior distributions of the surface normal, $p(x|\mathcal{Z}_{1:t})$, can be accurately and consistently estimated with sufficient measurements.

3.6.2 Occupancy Estimate of Single Cell

As shown in (3.21), the posterior surface normal state depends on the posterior occupancy estimates; and only depends on them if $p(o^i|x)$ and $p(x)$ are constant. Therefore, in order to show that the sufficient measurement assumption is valid in both 2D and 3D, the posterior occupancy estimates of occupancy grid cells, $p(o^i|z^i_{1:t})$, with different priors are compared. Figure 3.8 shows the KLD computed between the two posterior occupancy estimates of two separate grid cells as a function of consistent measurement counts (i.e. no false positives) and varying values of measurement likelihood, $p(z^i|o^i)$. One grid cell is initialized with the 'true' prior, $p(o^i) = \sum_x p(x) \cdot p(o^i|x) = 0.5923$ (Fig. 3.8 left) and $p(o^i) = 0.9$ (Fig. 3.8 right). The other cell is initialized with 'uniform' prior

$p(o^i) = 0.5$ as often used in occupancy grid mapping algorithms. The 'true' values of 0.5923 and 0.9 are the minimum and maximum possible prior based on the likelihood model $p(o^i|x)$ implemented, respectively. As the measurement likelihood value, $p(z^i = 1|o^i = 1)$, increases, fewer consistent measurements are required to reduce the KLD values. For $p(z^i = 1|o^i = 1) \geq 0.6$, it generally takes 13–15 measurements to converge (KLD within 1% variation). The likelihood value of $p(z^i = 1|o^i = 1) = 0.55$ requires more measurements to converge, as the measurement confidence is very low (close to 50/50). These results generally show that 15 measurements per cell are sufficient in order to confidently attain accurate and consistent posterior occupancy estimate; five measurements may be sufficient for most sensors that are reasonably accurate.

3.7 Experimental Results

The multi-level surface normal estimation algorithm is validated against theoretical truth (a few objects) and a benchmark point cloud approach using simulation and experimental data. The results of using only the highest resolution (leaf nodes of octree) are presented first, followed by the multi-level surface normal estimation results. Performance is evaluated in terms of 1) accuracy of the estimated surface normal vectors, 2) computation time and storage required, and 3) segmentation performance using estimated normal vectors. Comparisons between the outputs of the developed algorithm, the truth (a few objects), and the Point Cloud Library (PCL) [62] are presented.

Octomap is implemented to generate 3D occupancy grids, $p(o^i|z^i_{1:t})$. Octomap is an open-source library which enables memory efficient and fast occupancy

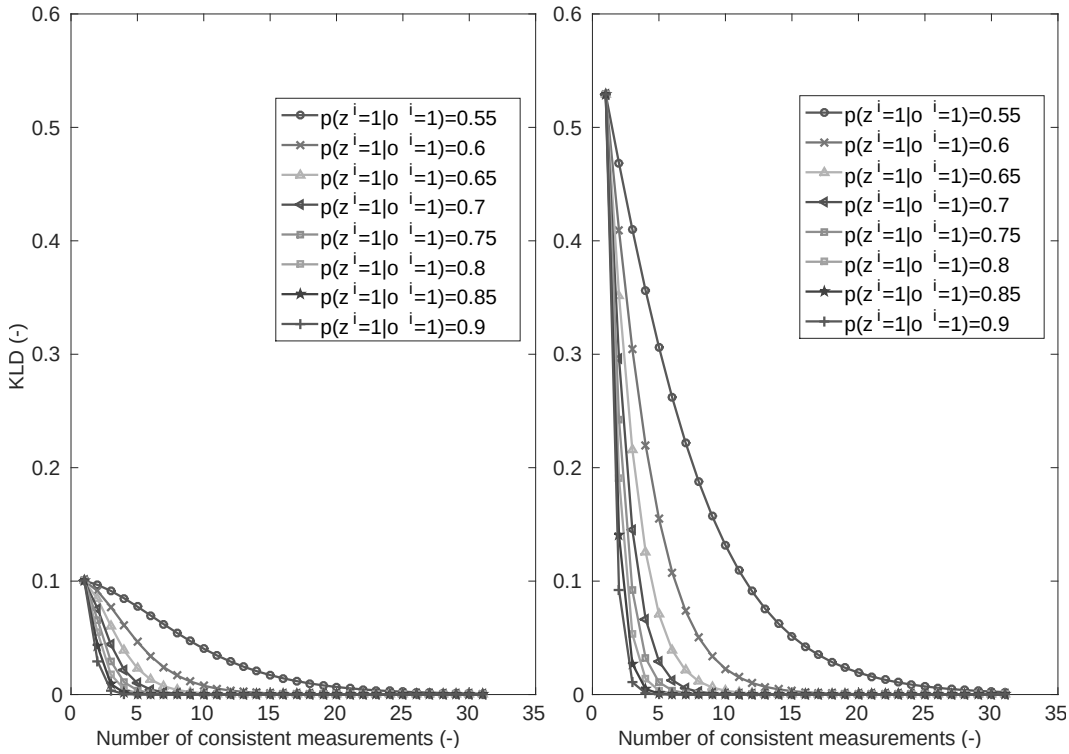


Figure 3.8: KLD computed between two occupancy grid cells with different priors (one initialized with the prior used in this work, one with uniform prior) as a function of number of consistent (i.e. no false positive) measurements. KLD changes are shown with the minimum possible prior ($p(o^i) = 0.5923$, left), and maximum prior ($p(o^i) = 0.9$, right).

grid mapping (including 3D ray tracing), while also providing occupancy grids in different resolutions. The smallest grid resolution is set to be 0.01m. Based on the occupancy grid, the scalable (S) surface normal estimation algorithm given in (3.21) is implemented.

The surface normal algorithm is used for all occupancy cells on the *surface* of the environment. The algorithm assumes that the cells being evaluated are on a surface of an object or area, with at least one empty adjacent cell; adjacent cells do not consist of ones in diagonal directions (both in 2D and 3D). Those cells not on the surface are not considered for accuracy evaluation.

PCL estimates of the surface normal vectors are generated using the eigenvectors of covariance matrix computed over neighboring points. The neighbor points are chosen to be within a circle with radius of 1.5 times the grid resolution in order to match the extent of neighboring cells used in S. The neighboring points are found using `kdtree` for fast search.

3.7.1 Simulation Results

Surface normal estimates are studied for seven objects in simulation: three synthetic objects: {cube, cylinder, sphere} and four man-made objects: {bookshelf, dumpster, barrel, cone}. Given the simplicity of the synthetic objects, the true (T) surface normals can be generated via geometry. The Gazebo 3D simulator [36] is used to generate the objects and sensor measurements. The top row of Fig. 3.10 shows all seven objects, with photos taken from same position to show scale. The cube has side length of 1.125m, while cylinder and sphere are 1.125m in diameter; the cylinder's height is 1m. For each object, a Kinect sensor is placed at 16 different locations around a circle with 2m radius. The Kinect sensor measurements are simulated to provide range data over a 640×480 resolution, with a 60 degree of field of view. Gazebo currently does not add sensor noise to the Kinect measurements. Thus, each 3D point is perturbed with 3D Gaussian white noise with standard deviation: $\sigma_x = \sigma_y = \sigma_z = \{0.000, 0.001, 0.0025, 0.005\}$m.

Accuracy Metric

The estimated surface normal vectors using the occupancy grid are compared to those of using the truth (synthetic objects) and PCL (all objects). Intuitively,

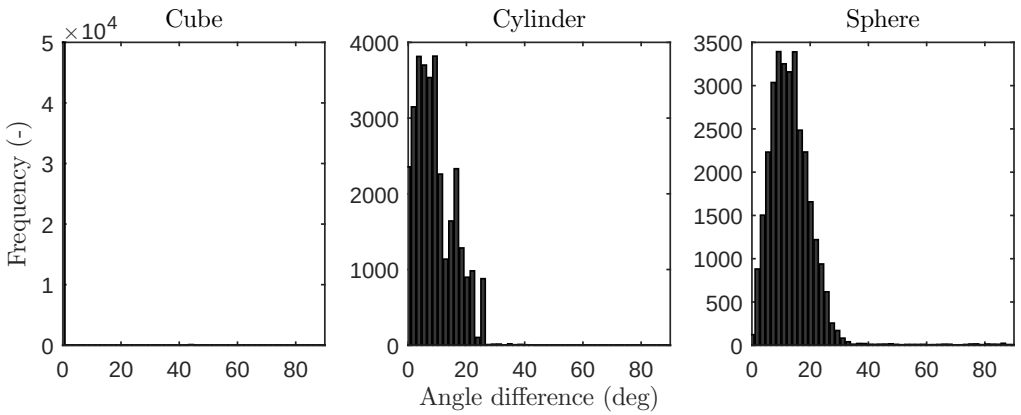


Figure 3.9: Angle difference histogram for synthetic objects (cube, cylinder, sphere) between the proposed S algorithm and the true normal direction.

PCL's normal vectors converge to the true state as the number of measurement points increases. The angle difference between two normal vectors using the scalable algorithm S and either the truth (T) or PCL is computed and used to evaluate accuracy. The closest point to the center of each occupancy grid cell is chosen for comparison. The angle between two normals are computed using dot product as:

$$\theta = \arccos \frac{\overrightarrow{n}_{\text{PCL/T}} \cdot \overrightarrow{n}_{\text{S}}}{\|\overrightarrow{n}_{\text{PCL/T}}\| \, \|\overrightarrow{n}_{\text{S}}\|}$$

where $\overrightarrow{n}_{\text{PCL/T}}$ and $\overrightarrow{n}_{\text{S}}$ represent normal vectors computed using PCL or T and S, respectively. The vector $\overrightarrow{n}_{\text{S}}$ is computed using (3.2).

Figure 3.9 shows binned histograms of the frequency of angle difference, θ, between S and T, for all grid cells for three synthetic objects. The true normal directions are analytically computed. For cube object, the estimated normal vectors match the truth very well with 99% of difference less than 5 degree; those few points with higher angle difference correspond to the edges of cube. The cylinder object has wider range of difference with 63.7%, and 93.5% of the difference less than 10 and 20 degree, respectively. The sphere object has a wider

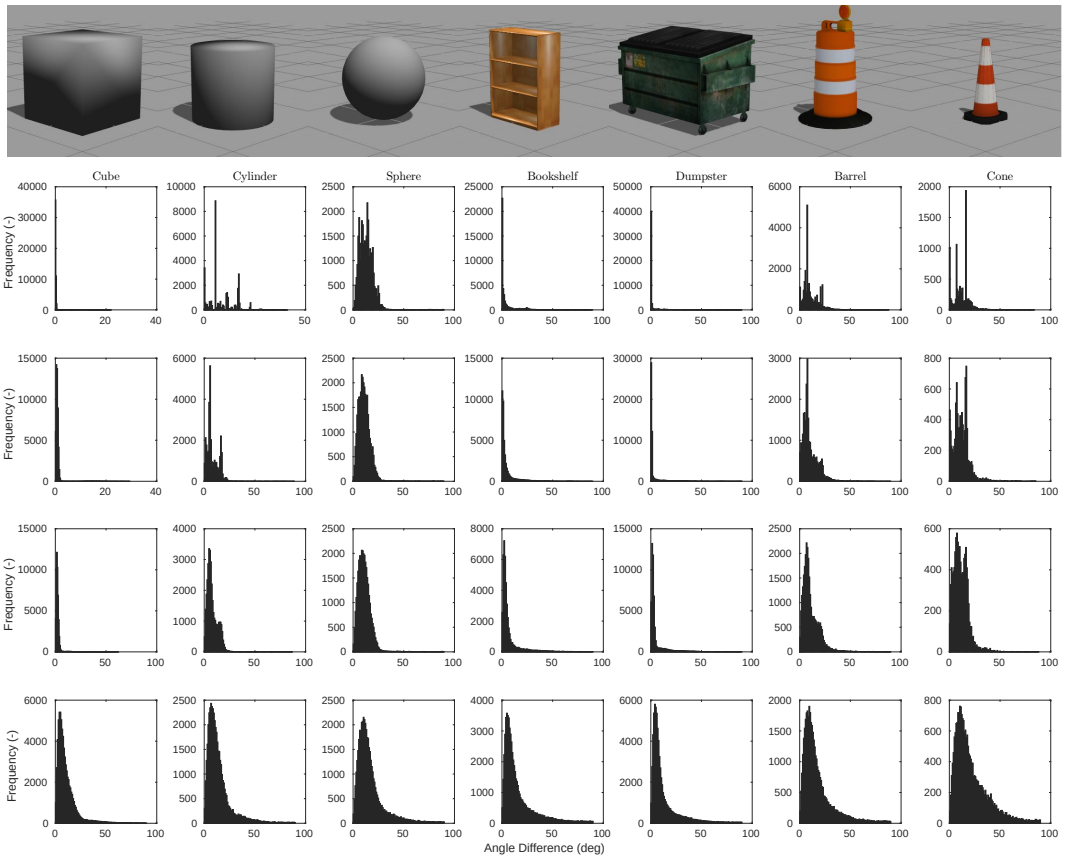


Figure 3.10: Simulation results comparing the scalable surface normal estimation algorithm with PCL for seven objects in simulation. (top) Objects used in simulation: {cube, cylinder, sphere, bookshelf, dumpster, barrel, cone} (bottom) Histograms of angle difference compared to those of PCL. The horizontal axis represents the angle difference in degree, while the vertical axis shows frequency. Each column corresponds to an object. Each row shows the results of using different magnitude of perturbation $\{\sigma_x = \sigma_y = \sigma_z\} \in \{0.000, 0.001, 0.0025, 0.005\}$m from top to bottom.

distribution of angle differences with 61.7% and 91.6% less than 10 and 20 degree, respectively. These results indicate that the continuously varying surfaces of the cylinder and sphere are more difficult to estimate than the regular cube. This phenomenon is primarily due to the discretization/resolution associated with curved surfaces. The discretized grid space has limited capability to de-

scribe curved surfaces, especially when considering only three neighboring cells in each dimension. As the amount of curved surface increases (sphere), performance declines. In addition, the number of possible surface normal directions is also limited using a constant measurement likelihood value, $p(z_t^{()} = 1|o^{()} = 1)$. This implies that the performance of the surface normal estimation algorithm is a function of the environment and the user chosen model parameters. This is due to the discretization of the occupancy grid as well as the discretization of the surface normal space. However, given that more than 60% and 90% of all angle differences are less than 10 and 20 degree, respectively, the proposed algorithm is sufficiently accurate for even challenging objects.

Figure 3.10 presents results for all seven objects comparing the scalable (S) algorithm with PCL. The bottom four rows of Fig. 3.10 show binned histograms of the frequency of angle difference between the S algorithm and the PCL estimates; four rows with different noise levels; the top row is for noiseless (best) case. The histograms of cube, bookshelf, and dumpster show better accuracy compared to the other objects with curved surfaces – cylinder, sphere, barrel, and cone; this is consistent with the prior comparisons with the truth (Fig. 3.9).

As the magnitude of sensor noise increases (from top to bottom in Fig. 3.10), the angle difference distribution becomes smoother and wider for all object types. This is more prevalent for objects with curved surfaces; the frequency of larger angle difference (beyond 30 degree) increases more quickly compared to the flat surfaces. As the noise increases, the surfaces of the objects become less smooth since the sensor readings are less consistent; sensor noise of $\sigma_x = \sigma_y = \sigma_z = 0.005$m is half of the grid cell size. Therefore, occupancy estimates of the grid cells are less consistent and accurate, resulting in larger angle

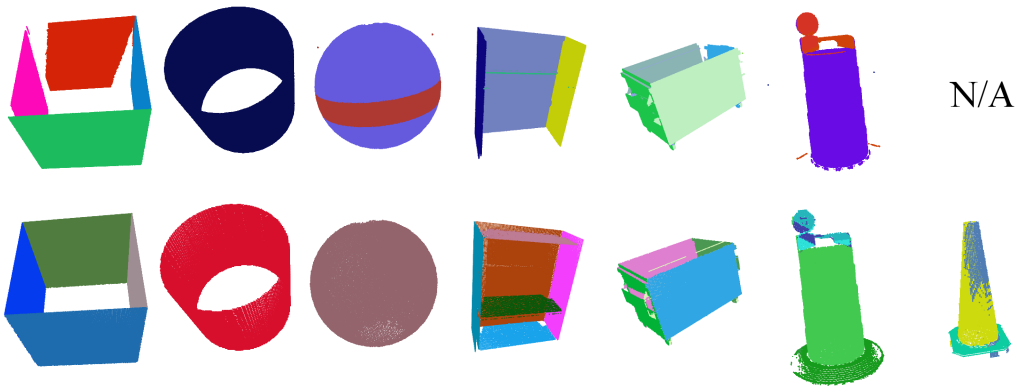


Figure 3.11: Segmentation results using surface normals of PCL (top) and S (bottom) for seven objects. The segmentation algorithm extracts plane, cylinder, or sphere shapes. Colors are selected randomly for each segment to simply show different segments. The baseline shape evaluated for each object from left to right: {plane, cylinder, sphere, plane, plane, cylinder, cylinder}

differences. Given the number of neighboring cells to consider, this inaccuracy is more prevalent for curved surfaces.

Computational Cost

The computational cost is also compared between the S and PCL methods in terms of memory (pts/cells) and time. Table 3.1 shows that the number of points in the point cloud (for PCL) is 15–41 times more than the number of cells, dependent on the object type. It is also noted that as more data is collected, the number of cells in S converges to a constant, whereas the number of points in the PCL model increases proportionally with the number of measurements. Computation time is measured using a timer of the `Boost` library, which measures CPU time based on Intel i7-3770 CPU (3.4 GHz) desktop computer. Table 3.1 shows the results using noisiest ($\sigma_x = \sigma_y = \sigma_z = 0.005$m) data, where t^n_{PCL} and t^n_{S} repre-

	Surface Normal Estimation					Segmentation	
	# points	# cells	t^n_{PCL}	t^n_{S}	t_{octo}	t^s_{PCL}	t^s_{S}
Cube	3176720	186008	25249.81	324.10	448.07	36.62	0.32
Cylinder	3076042	142115	37515.29	248.31	299.84	28.96	0.07
Sphere	2929328	143659	40509.98	258.27	228.69	2.80	0.02
Bookshelf	2968248	142719	38716.03	260.47	297.64	11.32	0.35
Dumpster	3052729	196255	21138.72	348.75	428.10	26.45	0.38
Barrel	2961883	117171	48308.70	201.69	211.60	27.15	0.14
Cone	2859320	68858	62164.26	112.41	76.51	N/A	0.05

Table 3.1: Comparison of computational resources: memory consumption (points & cells) and time (in second) between PCL and S. Both surface normal estimation and segmentation tasks are compared.

sent the surface normal computation time for each method in seconds. The time to estimate the posterior occupancy using Octomap, t_{octo}, is presented as well. Computation times for surface normals based on PCL are 60–553 times larger than the scaled S algorithm using the grid cells. Combined with Octomap computation time, the scaled S algorithm performs 27–329 times faster than PCL. The computation time for PCL increases with the number of points. Thus, using an occupancy grid reduces the number of points to keep significantly, which results into much lower computation time and memory required.

Segmentation Performance

Surface normal vectors are often used in segmentation algorithms. In order to demonstrate the subsequent usefulness of the estimated normal vectors, a segmentation algorithm is tested on the outputs of S and PCL. Simple segmentation algorithms finding plane, cylinder, and sphere baseline shapes are implemented using RANSAC [21]; the surface normal vectors estimated using S and PCL's

PCA analysis are used. Figure 3.11 shows segmentation results for seven objects using the output of both algorithms; top row for using PCL, and bottom for using S. In order to show the robustness of the developed algorithm, the largest sensor noise, $\sigma_x = \sigma_y = \sigma_z = 0.005$m, are used. Each segmented shape is rendered in random color; thus colors do not represent correspondences between results of S and PCL. For synthetic objects, both S and PCL perform well except PCL created a ring-shaped near the center for sphere model. For the bookshelf model, S yields better plane segments than PCL. PCL does not find the roof and the base, nor the shelves (top shelf has only few points), while S successfully finds them all. The bottom ring of the barrel is not found using PCL. And, while the cylinder shape is not found using PCL, several segments of the cone are found using S. These results generally show that the S method providing the same or better results for segmentation than PCL.

Table 3.1 also shows the computation time for segmentation; t^s_{PCL} and t^s_{S} represent the computation time in seconds using the normal estimated from PCL and S, respectively. Using an identical segmentation algorithm, S takes a fraction of second to yield good segmentation performance while PCL takes much longer. S performed the segmentation at least 95 times faster (dumpster), and 866 times faster at most (cone). This is due to the fact that PCL's performance scales with the number of measurements, while S does not.

3.7.2 Multi-resolution Surface Normal Estimation Results

Multi-resolution surface normal estimation results are presented for several different resolutions using MR1 and MR2 algorithms as presented in Section 3.4.

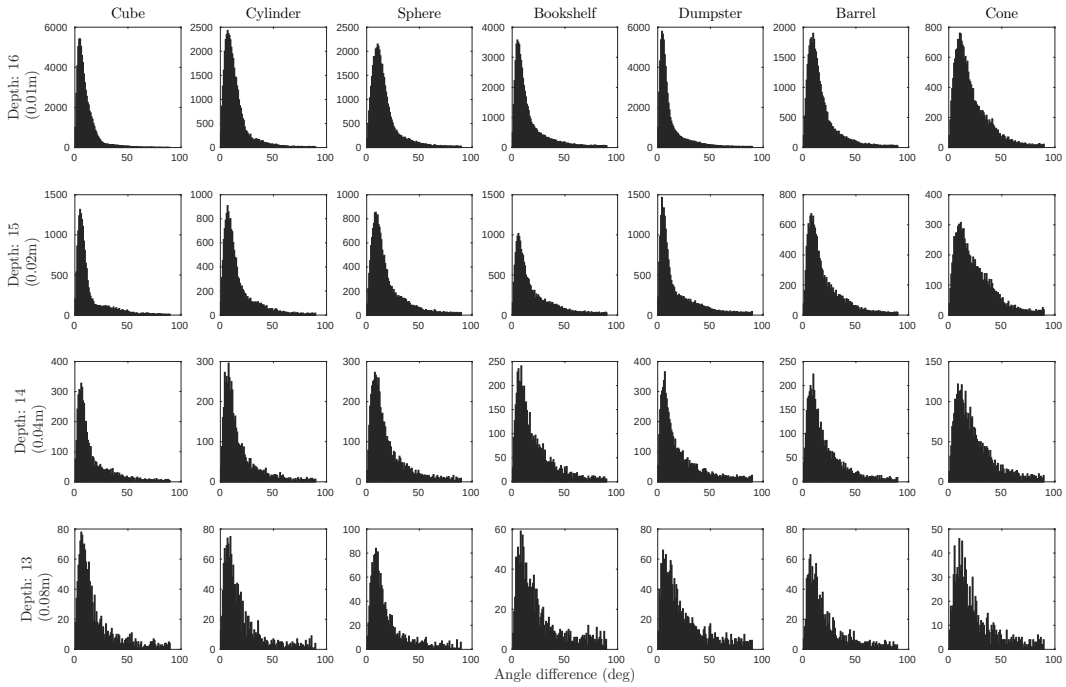
	Number of Valid Cells (-)							
	MR1				MR2			
	Depth 16 (0.01m)	Depth 15 (0.02m)	Depth 14 (0.04m)	Depth 13 (0.08m)	Depth 16 (0.01m)	Depth 15 (0.02m)	Depth 14 (0.04m)	Depth 13 (0.08m)
Cube	73685	19212	6367	1938	73685	12963	2860	605
Cylinder	46159	18652	6022	1986	46159	8601	2124	352
Sphere	47265	20859	7441	2204	47265	7255	1472	150
Bookshelf	63255	23079	6865	2008	63255	12435	2569	517
Dumpster	73998	24871	8471	2313	73998	13515	2840	471
Barrel	42271	17855	6109	1821	42271	7049	1703	36
Cone	23536	12406	4992	1571	23536	2788	377	0
	Surface Normal Estimation Computation Time (s)							
	MR1				MR2			
Cube	219.62	112.83	88.33	83.49	219.62	34.88	9.64	2.19
Cylinder	131.67	82.13	61.83	54.01	131.67	30.26	8.79	1.89
Sphere	136.35	86.66	63.53	54.39	136.35	29.81	8.44	1.97
Bookshelf	181.35	103.64	74.16	69.13	181.35	38.83	8.61	2.06
Dumpster	211.34	120.90	92.71	83.97	211.34	42.56	11.11	2.23
Barrel	120.15	74.64	55.32	48.67	120.15	29.71	8.30	1.64
Cone	63.01	43.77	30.87	25.04	63.01	16.42	5.52	1.31

Table 3.2: Number of Valid Cells and Computation Time Comparison between MR1 and MR2

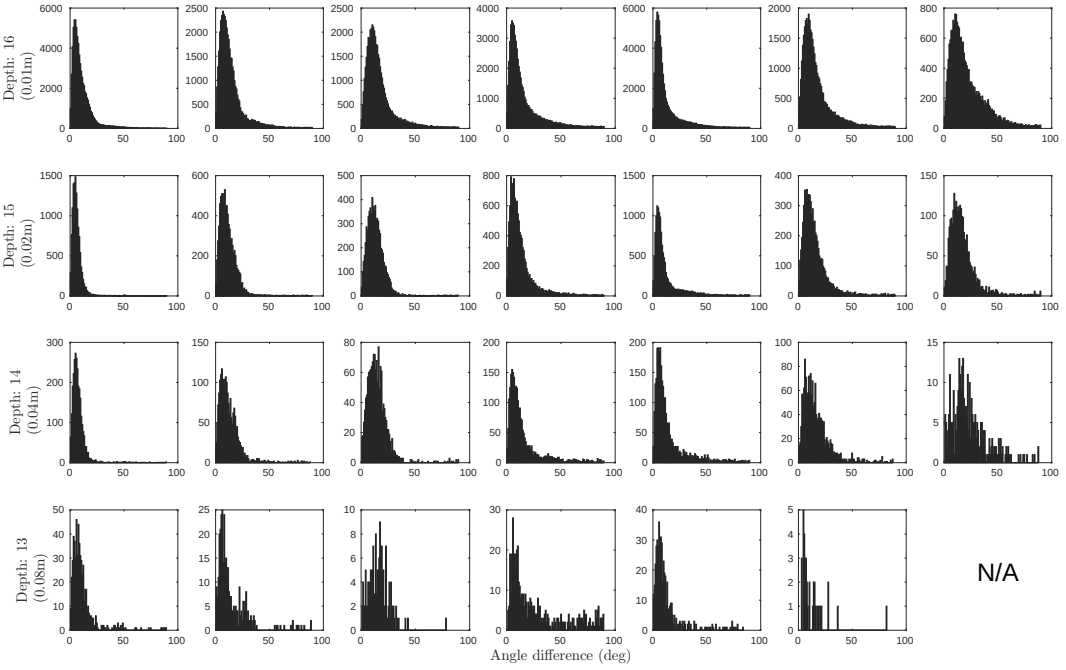
The Octomap used has total of 16 depth levels; the bottom (level 16) depth has resolution of 0.01m, and top (level 1) depth has resolution of 655.36m.

The seven objects presented previously are used for comparison. Among the multiple noise levels, the largest noise ($\{\sigma_x = \sigma_y = \sigma_z\} = 0.005$m) is chosen to demonstrate the robustness of the algorithm as 2σ equals to the cell width. Given the size of the objects, resolution lower than level 13 (resolution of 0.08m) is too large to yield meaningful results; thus, only depth levels of 13–16 are presented. Angle difference histograms are computed for both MR1 and MR2 algorithms in comparison with PCL. The angle difference is computed compared to the nearest normal estimate of PCL using the full dense point clouds (without discretization).

Figure 3.12 shows the binned histograms of angle difference for each object for depth levels 16–13, using MR1 and MR2. As the depth level decreases, the



(a) Angle difference histogram of using MR1



(b) Angle difference histogram of using MR2

Figure 3.12: Angle difference histograms (compared to PCL) for two multi-resolution surface normal estimation algorithms (MR1 in (a) and MR2 in (b)) computed for seven objects for octree depths from 16 (top) to 13 (bottom). Angle difference is computed for the simulation data perturbed noise magnitude of ($\sigma_x = \sigma_y = \sigma_z = 0.005$m), which is large compared to the cell size (2σ equals to cell width).

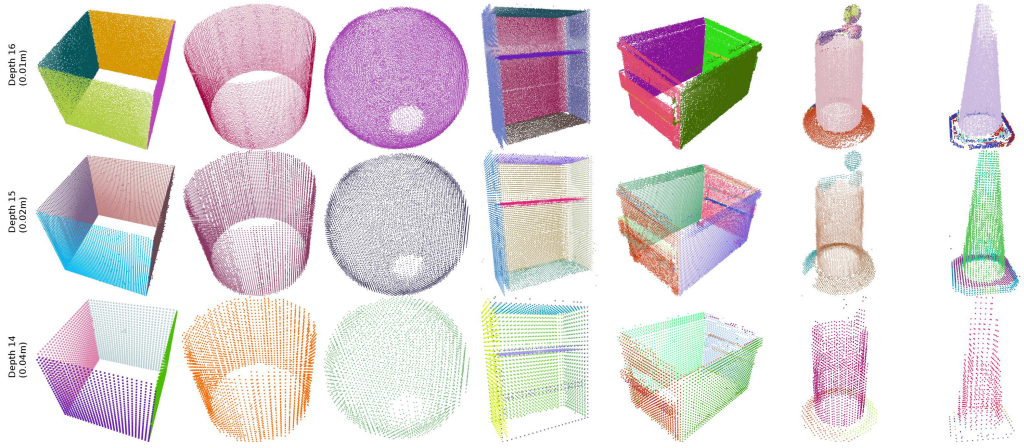
number of occupied cells decreases substantially for both algorithms. This is primarily due to the size of the voxel doubling in each dimension (see Table 3.2 top). It is clear that the decrease in the number of valid cells in MR2 is substantially faster than those of MR1 as the resolution increases.

For MR1 (Fig. 3.12(a)), as the size of grid cells increases, the frequency of larger angle difference increases, as shown in depth 13 and 14; the tails of the distribution for large angle differences become bigger. Intuitively, the angle differences increase as the resolution increases partly because the larger cells can contain vastly different surface shapes. Also, the histogram becomes more varied due to the discretization. The histograms using MR2 (Fig. 3.12(b)), however, create more consistent angle difference distributions among depths 16, 15, and 14, despite the small number of valid cells. Comparing MR1 and MR2 at depth 15 and 14, MR2 has smaller mean angle differences, except the cone object at depth 14. The cone is small compared to other objects; the number of valid cells is 377 for MR2 compared to 4992 for MR1. While MR1 and MR2 are comparable in terms of accuracy, with significantly lower number of valid cells, MR2 is more desirable especially when computational resource is limited. The non-circular objects (cube, bookshelf, dumpster) show similar degradation in accuracy compared to the circular objects as the resolution decreases.

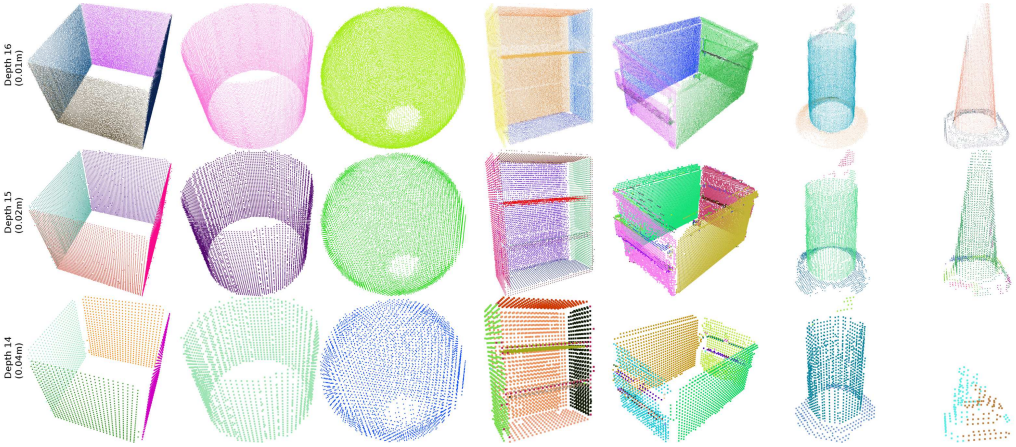
Table 3.2 shows the computation time required to estimate surface normals using the MR1 and MR2 algorithms. More specifically, the time required to compute the average vectors using all children at a given resolution (see (3.31) and (3.32)) ; MR2 is the time required to infer the surface normals using (3.21). Overall, the computation time required by MR2 is significantly smaller than that of MR1. For depth 15 (resolution of 0.02m), MR2 requires half the compu-

tation time compared to MR1, and at least 5.6 times less for depth 14 (resolution of 0.04m). Considering that MR2 shows comparable performance to MR1 as shown in Fig. 3.12, the amount of reduction in computation is substantial. At depth 13 (resolution of 0.06m), the number of valid cells becomes too small for barrel and cone.

Figure 3.13 shows the segmentation results of MR1 and MR2 using three different depth levels for all objects. Each row corresponds to depth 16, 15, and 14 from top to bottom, respectively. Segmentation is performed again using RANSAC. Each segmented shape is colored randomly to enable easy visualization. Segmentation performance using lower resolutions (depths 15 and 14) demonstrates nearly identical results compared to those of using the highest resolution (depth 16). These lower resolution models have significantly lower number of grid cells, yet are able to successfully estimate and classify the shapes. Comparing MR1 and MR2, all synthetic objects are well segmented for MR2 for all depth levels despite the smaller number of valid cells. For the bookshelf, the base is not correctly segmented using MR1 (depth 14), while MR2 is successful. The middle shelf of the bookshelf (depth 15) using MR1 is also missing, while MR2 successfully finds the shelf. Cylindrical shapes are segmented for both the barrel and cone. The body of the barrel is segmented successfully for both MR1 and MR2 over all depth levels, but the bottom support segmented by MR2 is not as clear as MR1. For the cone (depth 14), only small portions of the cone shape are segmented due to very small number of valid cells. Using multiple resolutions, various shapes are successfully segmented, especially when using small number of valid cells and lower resolution.



(a) Segmentation results using MR1



(b) Segmentation results using MR2

Figure 3.13: Segmentation results using different resolutions and different estimation method for each object. Each row in each figure corresponds to the each tree depth: {16, 15, 14} from top to bottom. Level 16, 15, and 14 have grid resolution of 0.01m, 0.02m, 0.04m, respectively. Colors are randomly chosen to better visualize segmented surfaces. Points are enlarged for better visualization as needed.

3.7.3 Experimental Data Results

The multi-resolution surface normal estimation algorithm is tested on Office 2A13 dataset of Univ. of Zurich [45]. This dataset contains 3D LIDAR scans of an office environment. The original dataset has very dense 3D LIDAR readings covering the entire office space, and is therefore trimmed to the more complex portion as shown in Fig. 3.14. The total number of 3D points stored and used for PCL is 8921120, whereas 181995 grid cells are stored for the smallest resolution.

The angle difference histograms compared to PCL are shown in Fig. 3.15 for four different resolutions. At depth 16, the largest two peaks lie at 1.7 degree and 16 degree, while there are small number of outliers with larger than 20 degree difference. The computation time for surface normal estimation by PCL is 2190.17 seconds, whereas S took only 235.76 seconds in CPU time for the highest resolution.

The multi-resolution MR2 method is used as a comparison. The number of valid cells for depth level {16 (resolution of 0.01m), 15 (0.02m), 14 (0.04m), 13 (0.08m)} are {181995, 37707, 10830, 2896}, respectively. The computation time for depth level {16, 15, 14, 13} are {235.76, 187.68, 140.65, 125.86} seconds, respectively. As shown in Fig. 3.15, the angle difference distribution is maintained as the depth decreases from 16 to 14. At depth 13, however, the frequency of larger angle difference increases and becomes more noisy.

Figure 3.16 shows the segmentation results for segmenting plane shapes, for four different resolutions. The wall, desk, monitor, and floor are segmented successfully across all resolutions. Objects on the desk have small height, and are mostly flat; with most of the normal vectors aligned, the objects are considered

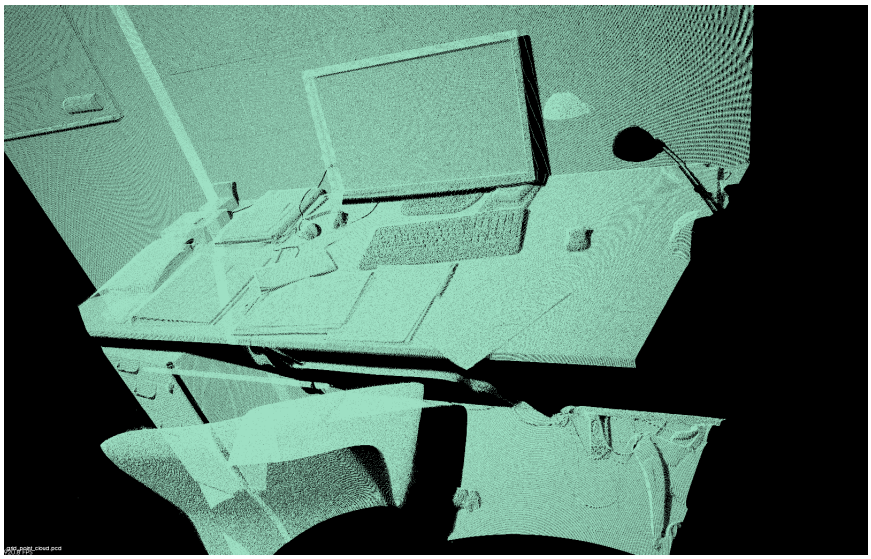


Figure 3.14: Environment used in the experiment from Office 2A13 dataset. Multiple office objects are shown along with a chair and walls.

to be part of the desk. Other objects, such as drawer, and seat and back of chair, are successfully segmented as planes for the highest resolution (depth 16). As the resolution decreases, however, objects with curved surfaces, such as seat and back of chair, are segmented into multiple pieces. The flat surfaces, such as wall, desk, monitor, and drawer are successfully segmented even at the coarest resolution.

3.8 Conclusion

A scalable, efficient probabilistic multi-resolution surface normal estimation method is developed. Probabilistic occupancy grid information at multiple resolutions is used as a basis for the approach. The estimated surface normals can be used to describe spatial features, and thus are an important foundation for high-level segmentation, object detection, and classification tasks. The strengths

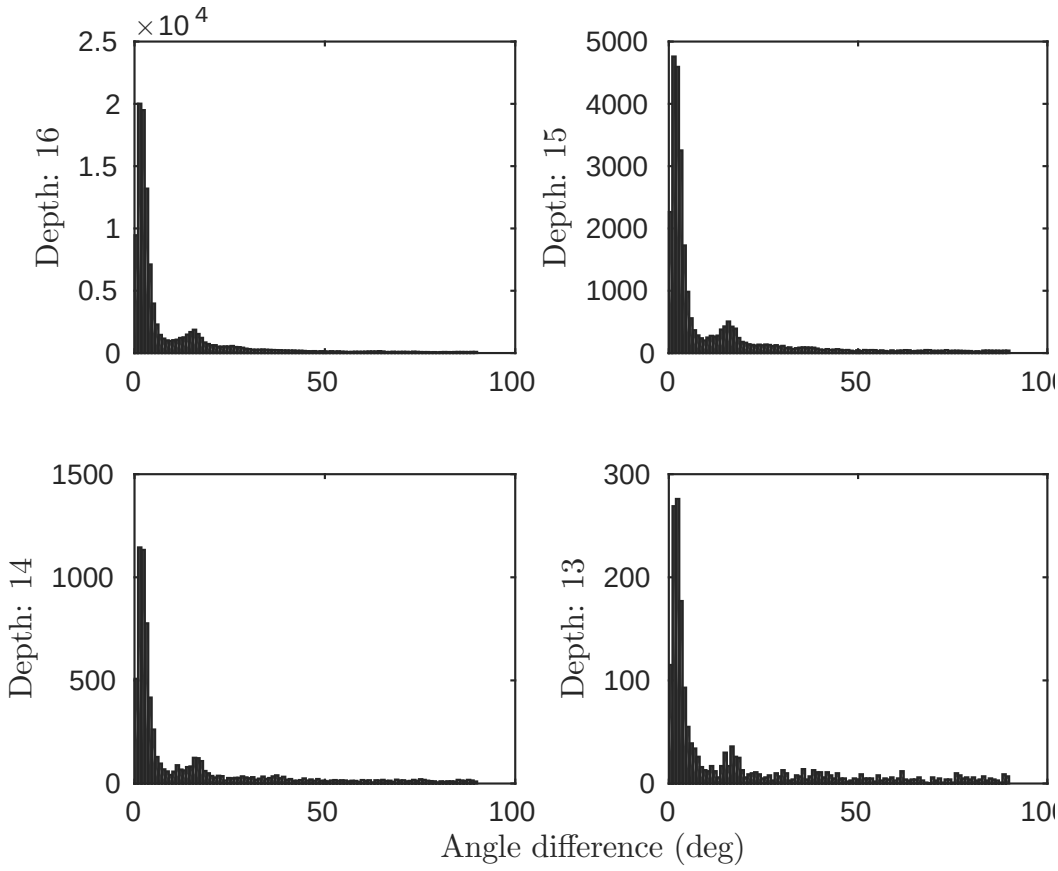
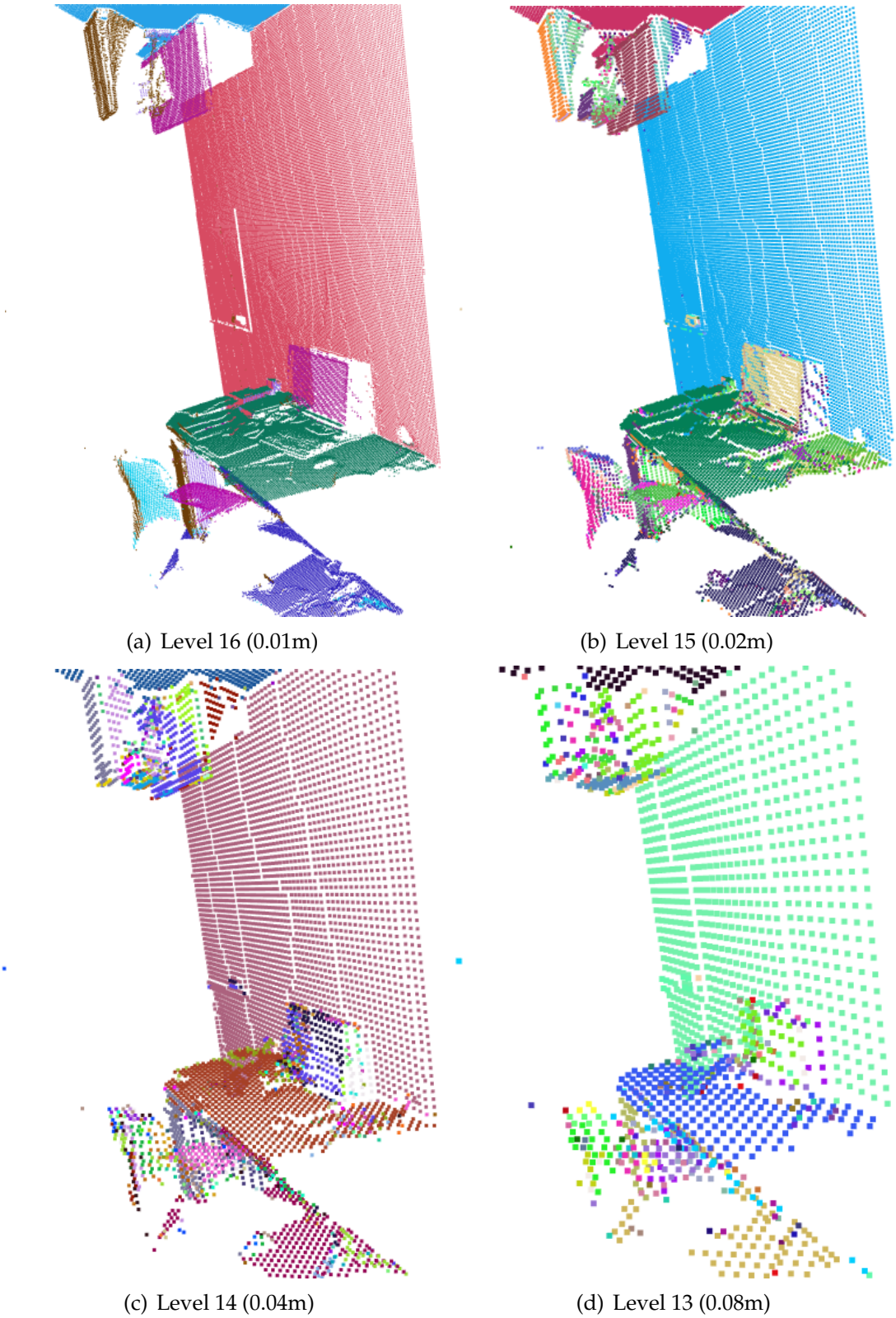


Figure 3.15: Angle difference histograms for 2A13 dataset for four different resolutions

of the proposed algorithm are: 1) a probabilistic algorithm that scales with the number of grid cells, not the number of measurements; 2) reduced computational demands (time and storage) compared to benchmark methods using point clouds; 3) multi-resolution estimates are available using multi-resolution occupancy grid such as Octomap; 4) both occupied (positive information) and empty (negative information) grid cells are used enabling viewpoint ambiguity to be naturally considered; point cloud methods cannot estimate the viewpoint directly; and 5) either raw data, or an already running occupancy grid algorithm can be used, thus further reducing the effort in processing the same data into another representation. While reducing the computation to real-time performance,



(a) Level 16 (0.01m)

(b) Level 15 (0.02m)

(c) Level 14 (0.04m)

(d) Level 13 (0.08m)

Figure 3.16: Segmentation results using multiple resolutions

the algorithm is shown to converge to the full algorithm estimates over time as sufficient number of measurements become available. Results in simulation and using experimental data show that the performance of the scalable surface normal estimation methods are comparable to that of PCL and the truth (mostly under 20 degree of angle difference), with significantly less computation (15–41 times less storage and 96–866 times faster). Two methods developed for surface normal estimation in multiple resolutions show substantial reduction of memory and computation time, thus enhancing real-time performance. Despite much smaller number of grid cells used in lower resolutions, the accuracy of the surface normal vectors are well maintained. The developed surface normal estimation algorithm also enables accurate and fast segmentation throughout various different resolution settings.

CHAPTER 4

BIAS AND UNCERTAINTY ESTIMATION OF MODULAR ROBOTS WITH SENSORS

4.1 Introduction

Modular robots have been developed over last decade. Modular robots are often aimed to be self-reconfigurable which enables adaptive reconfiguration based on the environment or task complexity. Such robustness, along with relatively low cost to manufacture through modern methods such as 3D printing, has led to a very active research over the last decade.

Many modular self-reconfigurable robotics platforms have been developed. A cube-shaped modular robot in a lattice structure that travels and slides on a surface using magnets was developed by An [2]. A distributed flight array was developed where multiple flying platforms (up to six) with single propeller assemble together into a lattice structure [53]. Yim *et al.* developed CKBot whose connections are in a chain-structure [80]. Sambot was developed whose module can individually act as an independent module as well as combine with other modules to reconfigure [78]. UBot was developed using universal joints such that it can achieve two degrees-of-freedom in a unit of one cubic space [82]. SMORES robots were developed to enable 4 degrees-of-freedom and easy reconfiguration [12]. Some modular robots are also commercially available [61].

Given their adaptability, all modular robotics systems are attempt to achieve multiple high-level tasks. In addition to hardware, the ability to achieve these tasks requires accurate and consistent knowledge of itself and the environment.

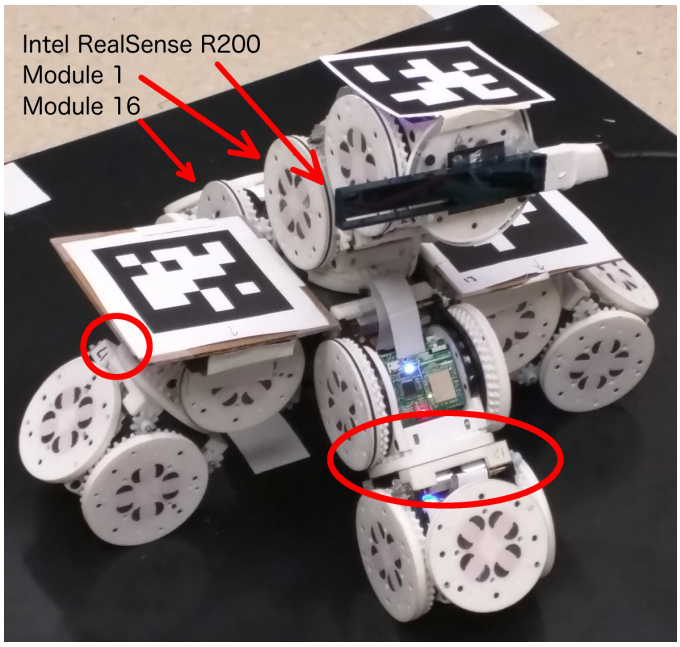
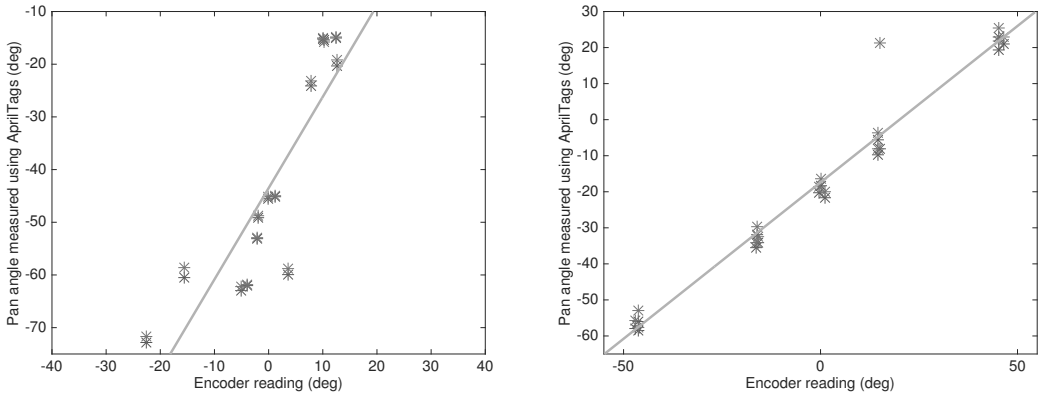


Figure 4.1: A set of SMORES modules used for experiments. Circles denote potential imperfect connections made between modules. Intel RealSense R200 sensor is used for image and depth information acquisition.



(a) Module 1 pan angle

(b) Module 16 tilt angle

Figure 4.2: Encoder values and measured pan and tilt angles using a visual fiducial system (AprilTag) for Modules 1 and 16, respectively. Data points and a linear fit are shown.

Typically, one or multiple modules will have sensors for measuring the environment. The accuracy and consistency of the sensing and perception the environment, therefore, requires an accurate sensor position estimate to measure the environment.

In order to obtain accurate sensor position estimates, a number of challenges arise – encoder measurement biases, manufacturing errors, imperfect mechanical connections between modules, and material bending and twisting due to gravity. Figure 4.1 shows a set of SMORES [12] modules. Each connection between modules is imperfect, where there are 2D position and orientation errors. Figure 4.2 shows the results of a simple experiment. Each module was connected with a cube whose each face has a visual fiducial tag (AprilTag) [52] attached. One camera was mounted on the ceiling looking downward, and another camera was placed on the side. Each camera provides relative position and orientation of the tags with respect to the camera that are seen in the view. The experiment studied the accuracy of reported encoder measurements compared to the true angle measured using AprilTag measurements. Module 1 shows about 45 degree encoder bias for pan angle; zero degree encoder measurement maps to about 45 degree orientation (Fig. 4.2(a)). Module 1 also shows the rate of change (shown with the slope of linear fit line) between encoder values to the measured angle is higher than one. Module 16 shows the similar behavior as well for tilt angle (Fig. 4.2(b)).

Errors such as those in Fig. 4.2 introduce significant estimation error for the sensor position on a specific module. The encoder biases and connection offsets can be constant and not changing throughout an experiment after connections between modules are made, such as manufacturing errors. Depending on the

quality of encoder sensor, however, the error of the encoder measurements to the actual orientation angle can be nonlinear, scaled as shown in Fig. 4.2, or even time varying. In addition, bending and twisting due to gravity can be highly complex depending on configurations, thus difficult to model.

In this work, a set of solutions are presented to create custom error and calibration models for modular robots, with the ultimate goal of improving the performance of a sensor module. Section 4.2 describes the problems of interest and their formulations. Section 4.3 presents the proposed approaches and algorithms in detail. Section 4.4 present the performance of proposed algorithms in both simulation and experiment run with SMORES modules, followed by a conclusion.

4.2 Problem Statement

The problem of estimating sensor position can use an error/uncertainty model in three different ways: 1) a constant bias in the encoder; 2) a constant bias in the encoder, and connections between modules have translation and rotation offsets; 3) nonlinearities between the measured encoder and true angle, to also include bending and twisting due to external forces.

All three approaches/representations seek to solve for the ideal sensor position, which can be written as:

$$p_s = f(u_e) \tag{4.1}$$

where $f(u_e)$ is a function that maps the encoder measurements, u_e, to the true relative sensor position, $p_{s,(\cdot)}$, given a module configuration. The relative sensor

position, $p_{s,(\cdot)}$, can be estimated in three different formulations such as:

$$p_{s,\text{alg1}} \approx f_b(x_b, u_e) + \omega_b \tag{4.2}$$

$$p_{s,\text{alg2}} \approx f_{b,o}(x_{b,o}, u_e) + \omega_{b,o} \tag{4.3}$$

$$p_{s,\text{alg3}} \approx \hat{f}(u_e) + \omega_f \tag{4.4}$$

where each equation represents each of the three different formulations. Each formulation is discussed in Section 4.2 in detail. The function $f_b(x_b, u_e)$ maps the encoder values, u_e, along with the encoder biases, x_b, to a sensor position, $f_{b,o}(x_{b,o}, u_e)$ is a function that maps the encoder values along with the encoder biases and connection offsets in translation and orientation, $x_{b,o}$. Variable $\hat{f}(u_e)$ represents an estimated function that maps the encoder values to a sensor position. Each $\omega_b, \omega_{b,o}$, and ω_f represents noises, where each is distributed with a zero-mean Gaussian distribution. A measurement, y, from the sensor position, p_s, can be described using a function $h(\cdot)$ with a zero-mean Gaussian noise, $\omega_y \sim \mathcal{N}(0, \sigma_y^2)$.

$$y = h(p_{s,(\cdot)}) + \omega_y \tag{4.5}$$

The measurement, $y \in \mathbb{R}^{n_y}$, can be any arbitrary measurement that uses the sensor position, $p_{s,(\cdot)}$. Such examples consist of direct measurements of the sensor position, relative position and orientation measurements to a visual feature, or a visual tag such as AprilTag in the environment from the sensor, or a transformation between two 3D point clouds from two different time steps.

Modules with two actuators are considered in this work. One actuator provides body tilt angle, and the other provides pan angle. Figure 4.3 shows the four degrees-of-freedom of a SMORES module [12] as a reference. The pan angle refers to #3 DoF and body tilt angle refers to #4 in the figure.

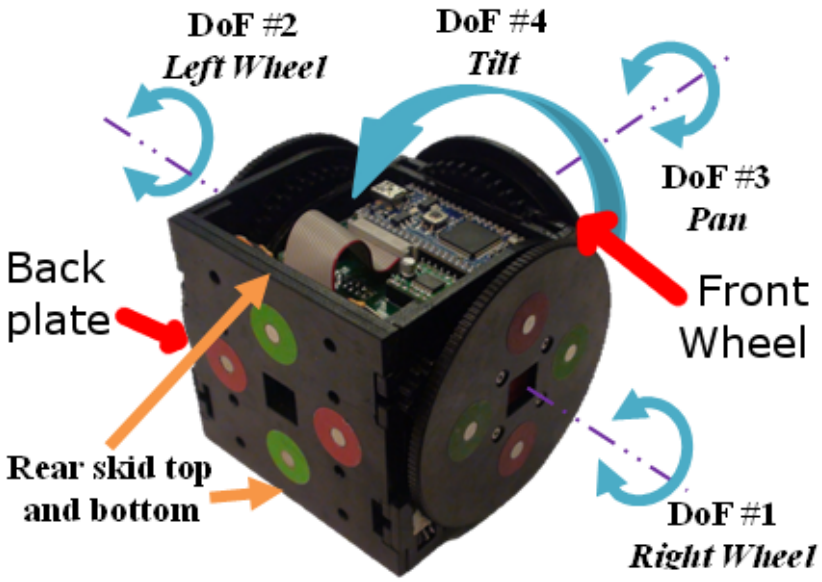


Figure 4.3: Four degrees-of-freedom of SMORES module [12]. This work considers a module with two degrees-of-freedom regarding DoF #3 and #4 shown here.

The goal of this work is to estimate the biases and uncertainty of a modular robot once a set of modules are organized into a specific configuration. The ultimate performance is that of the sensor, and the estimates of the scene is used to help evaluation. The uncertainty model can then be used by the robot as it operates in that configuration. The proposed auto-calibration procedure operates as follows. First, the modules execute a number of carefully predefined motions while taking encoder measurements as well as the sensor module measuring a stationary object location. The data is then used to estimate the sensor position mapping, (4.2) – (4.4). Once the biases and uncertainty are estimated, the sensor position of a given configuration is estimated accurately, and can then operate and perform high-level sensing tasks.

4.3 Auto-Calibration: Bias and Offset Uncertainty Estimation

4.3.1 Observability Analysis

In order to estimate the encoder biases and connection offsets of each module, a set of motions must be carefully chosen to provide the maximum observable information. The degree of observability is computed using observability Gramian [7]. Given a complex configuration of a set of modules, however, it is non-trivial to compute the observability Gramian analytically, especially when the sensor position prediction function, $f_{(\cdot)}(\cdot)$, is highly nonlinear. As an alternative, the observability Gramian for a given time interval $t \in [0, T]$ at a state x^0, is computed numerically [55, 38] as follows:

$$\frac{1}{4\epsilon^2} \int_0^T (y^{+i}(t) - y^{-i}(t))^\mathsf{T} (y^{+j}(t) - y^{-j}(t)) dt \tag{4.6}$$

where $y^{\pm i} \in \mathbb{R}^n$ is defined to be $y^{\pm i} = y(x^0 \pm \epsilon e^i)$ where e^i is the i-th unit vector in $\mathbb{R}^n$, and $\epsilon > 0$ is a small displacement.

The degree of unobservability is measured using the ratio between the largest and the smallest singular values computed on the Gramian. The larger the ratio becomes, the larger the effect on the output caused by a small change of the initial condition becomes. This ratio is called *local estimation condition number* [38].

As the configuration becomes complex, determining the motions to provide maximum observability information is difficult, especially in a closed form. Therefore, a Monte Carlo approach is used, where a large number of encoder inputs to command, u_e, are sampled. Using these encoder input commands, the measurements, y, are simulated, and observability Gramians are computed for

each sample. A set of encoder values with minimum local estimation condition number are selected to be used to obtain the most state observable data.

4.3.2 Encoder Biases (Algorithm 1)

Consider the encoder biases only model (4.2), where only encoder biases are present. The encoder biases state, x_b, given N_m number of modules, can be defined as a $2N_m \times 1$ vector:

$$x_b = [\phi_1, \theta_1, \cdots, \phi_{N_m}, \theta_{N_m}]^\mathsf{T} \tag{4.7}$$

where ϕ_i and θ_i indicate body tilt angle (#4 in Fig. 4.3) and pan angle (#3 in Fig. 4.3) of i-th module, respectively.

Collecting M set of a sensor module measurements, $y^{(\cdot)}$, and encoder values, $u_e^{(\cdot)}$, eqn. (4.7) can be stacked, and a nonlinear least-squares problem can be formulated as:

$$\begin{bmatrix} y^1 \\ \vdots \\ y^M \end{bmatrix} = \begin{bmatrix} h(f_b(x_b, u_e^1)) + \omega^1 \\ \vdots \\ h(f_b(x_b, u_e^M)) + \omega^M \end{bmatrix} \tag{4.8}$$

$$\approxeq \begin{bmatrix} h(f_b(\hat{x}_b, u_e^1)) + J^1(x_b - \hat{x}_b) + \omega^1 \\ \vdots \\ h(f_b(\hat{x}_b, u_e^M)) + J^M(x_b - \hat{x}_b) + \omega^M \end{bmatrix} \tag{4.9}$$

where $\hat{x}_b$ and J^i represent the encoder biases estimate and Jacobian matrix of the measurement function, $h(f(\cdot))$, evaluated around an estimate $\hat{x}_b$, respectively.

An iterative nonlinear least-squares method is used to solve for the biases

estimate at $(i+1)$-th iteration as follows [7]:

$$\hat{x}_{b,i+1} = \hat{x}_{b,i} + \left(\mathbf{J}_i \mathbf{R}^{-1} \mathbf{J}_i\right)^{-1} \mathbf{J}_i^{\mathsf{T}} \mathbf{R}^{-1} \left[\mathbf{y} - \mathbf{h}(\mathbf{f}(\hat{x}_{b,i}, \mathbf{u}_{e,i}))\right] \tag{4.10}$$

where $\mathbf{J}_i$ and $\mathbf{R}_i$ represent a diagonally stacked matrix of Jacobian and a diagonally stacked measurement noise covariance matrix at i-th iteration, respectively. The stacked measurement vector,$\mathbf{y}$, is formulated as $\mathbf{y} = [(y^1)^{\mathsf{T}}, \cdots, (y^M)^{\mathsf{T}}]^{\mathsf{T}}$. Similarly, $\mathbf{h}(\mathbf{f}(\cdot, \mathbf{u}_{e,i}))$ represents the stacked vector of predicted measurements using all encoder values, $\mathbf{u}_{e,i}$ at i-th iteration. The covariance matrix of the estimated biases is computed as:

$$\mathbb{E}\left[(\hat{x}_{b,i+1} - x_b)(\hat{x}_{b,i+1} - x_b)^{\mathsf{T}}\right] = (\mathbf{J}_{i+1} \mathbf{R}^{-1} \mathbf{J}_{i+1})^{-1} \tag{4.11}$$

This least-squares formulation minimizes the quadratic error equation as shown:

$$C = \sum_{n=1}^{M} \left[y^n - h(f_b(x_b, u_e^n))\right]^{\mathsf{T}} R^{-1} \left[y^n - h(f_b(x_b, u_e^n))\right] \tag{4.12}$$

The iteration can stop once the change in quadratic error, C, converges to a sufficiently small value.

4.3.3 Encoder Biases and Imperfect Connection (Algorithm 2)

Modular robots can suffer from imperfect alignments, especially when a non-rigid connection mechanism is used. For example, connecting modules using magnets introduces translation and orientation offsets. A richer representation (4.3) is considered, where position and rotation offset errors on a connecting surface are assumed to be constant once the connections are made. The rotation offset, however, can be embedded in the encoder's pan angle bias, $\theta_{()}$, since the

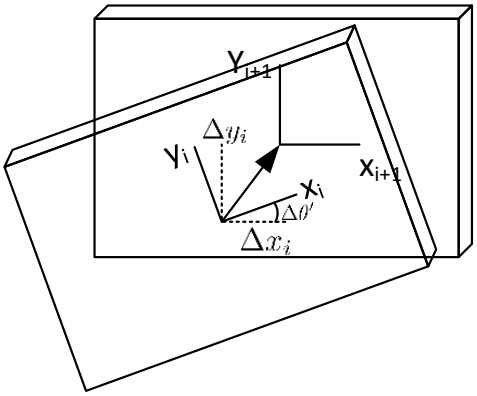


Figure 4.4: Diagram of translation and orientation offsets on a connecting surface between two modules. Δx and Δy represent translation offset, and $\Delta\theta'$ represent the orientation offset.

offset is constant (see Fig. 4.4). Therefore, $x_{b,o}$ represents the encoder biases and connection offsets as:

$$x_{b,o} = [\phi_1, \theta_1, \Delta x_1, \Delta y_1, \cdots, \phi_{N_m}, \theta_{N_m}, \Delta x_{N_m}, \Delta y_{N_m}]^\mathsf{T} \tag{4.13}$$

which is a $4N_m \times 1$ vector.

Given a sufficient number of accurate measurements, the encoder biases and the position and orientation offsets can be estimated using iterative nonlinear least-squares method similar to (4.10). Note that this formulation includes the model of Algorithm 1, and more parameters to estimate. Because of the increase of the number of parameters to estimate, obtaining enough data is important, and observability could become more of an issue.

4.3.4 Nonlinear Encoder Mapping and Material Compliance (Algorithm 3)

While the encoder biases and connection offset errors can be estimated via a set of measurements and a nonlinear iterative least-squares method because they are assumed to be constant, if the uncertainties are nonlinear from encoder measurements to the physical position of each module, the approach cannot be used. An alternative approach is to model each module's biases and uncertainty. However, if given a large number of modules, and a large number of complex configurations, this is undesirable.

A Gaussian process (GP) uncertainty model is proposed to capture the nonlinear function that maps the encoder measurements to a relative sensor position. A GP is implemented for each measurement state, resulting in n_y GPs. For each n-th measurement state element, the GP is modeled as:

$$y_n(u_e) \sim GP(m(u_e), \kappa(u_e, u'_e)) \tag{4.14}$$

where $m(u_e)$ is the mean function and $\kappa(u_e, u'_e)$ is the covariance function.

$$m(u_e) = \mathbb{E}[y_n(u_e)] \tag{4.15}$$

$$\kappa(u_e, u'_e) = \mathbb{E}[(y_n(u_e) - m(u_e))(y_n(u'_e) - m(u'_e))^{\mathsf{T}}] \tag{4.16}$$

Choosing the suitable parameters, $\mathbf{\Theta}$, for the covariance function plays a major role in GP performance. For squared exponential kernel that is used in this work, for an example,

$$\kappa(u_e, u'_e) = \sigma_f^2 \exp(-\frac{1}{2}(u_e - u'_e)^T \mathbf{M}(u_e - u'_e)) + \sigma_y^2 \delta_{u_e, u'_e} \tag{4.17}$$

consists of three parameters, $\mathbf{\Theta} = \sigma_f, l, \sigma_y$, where $\mathbf{M} = l^{-2}\mathbf{I}$. The optimal parameters given a set of training data can be found by maximizing the marginal likelihood of the data, $p(\mathbf{y}|\mathbf{X})$:

$$\log p(\mathbf{y}|\mathbf{X}) = \log \mathcal{N}(\mathbf{y}|0, \mathbf{K}) \tag{4.18}$$

$$= -\frac{1}{2}\mathbf{y}^\top \mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}| - \frac{N_t}{2}\log(2\pi) \tag{4.19}$$

where $\mathbf{y}$ represents a stacked vector of measurements, $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]$ represents N training input data, and $\mathbf{K}$ is a matrix where (i, j)-th element is computed using covariance function, $\mathbf{K}_{i,j} = \kappa(u_e^i, u_e^j)$. The partial derivative of the log likelihood with respect to the covariance function parameters, $\mathbf{\Theta}$, can be computed [50]:

$$\frac{\partial}{\partial \Theta}\log p(\mathbf{y}|\mathbf{X}) = \frac{1}{2}\mathrm{tr}\left(\left(\alpha\alpha^\top - \mathbf{K}^{-1}\right)\frac{\partial \mathbf{K}}{\partial \Theta}\right) \tag{4.20}$$

where $\alpha = \mathbf{K}^{-1}\mathbf{y}$. The maximum likelihood estimate of $\mathbf{\Theta}$ is found using gradient descent algorithm.

Once the optimal parameters, $\mathbf{\Theta}$, are found, the sensor position, y_*, for a single encoder input, $u_{e,*}$, given the training data $\mathbf{X}$ and $\mathbf{y}$, can be estimated as a Gaussian distribution:

$$p(y_*|u_{e,*}, \mathbf{X}, \mathbf{y}) = \mathcal{N}(y_*|\mu_*, \sigma_*) \tag{4.21}$$

$$\mu_* = \mathbf{k}_*^\top \mathbf{K}_y^{-1}\mathbf{y} \tag{4.22}$$

$$\sigma_* = k_{**} - \mathbf{k}_*^\top \mathbf{K}_y^{-1}\mathbf{k}_* \tag{4.23}$$

where $\mathbf{k}_* = [\kappa(u_{e,*}, \mathbf{x}_1), \cdots, \kappa(u_{e,*}, \mathbf{x}_N)]$, and $k_{**} = \kappa(u_{e,*}, u_{e,*})$.

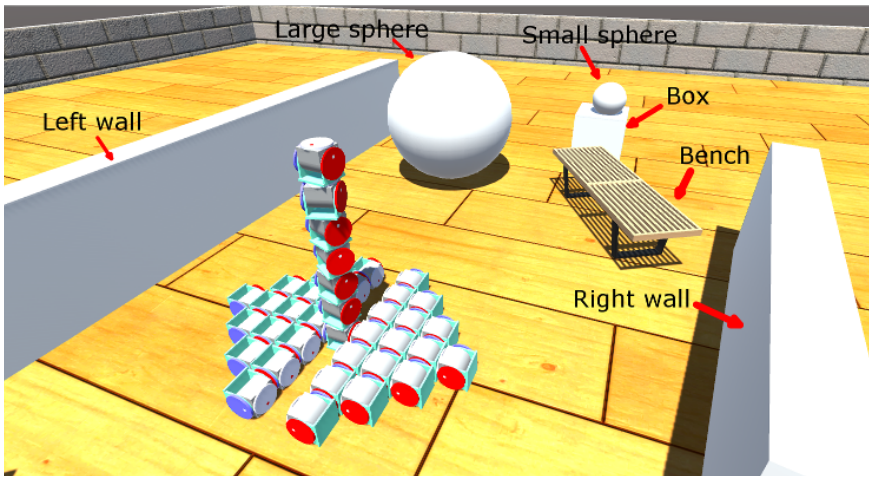


Figure 4.5: Unity simulation environment with object annotations

4.4 Simulation and Experiment

The performance of proposed bias and uncertainty models is evaluated in both simulation and experiment.

4.4.1 Simulation Results

Unity [69] is used to simulate modular robotics behavior along with sensor measurements. Modular robots are simulated based on SMORES robots [12]. Gravity, and bending and torsion due to material compliance are simulated as well using Unity's built-in physics engine.

Figure 4.5 shows the simulation environment as well as the set of modules used. Total of 39 modules are used, where 32 are used to construct a stable base in order to enable an arm configuration. The arm consists of seven modules connected in a serial fashion; a back plate is connected with a front wheel of the other module (see Fig. 4.3). The module at the top of the arm is set to be a

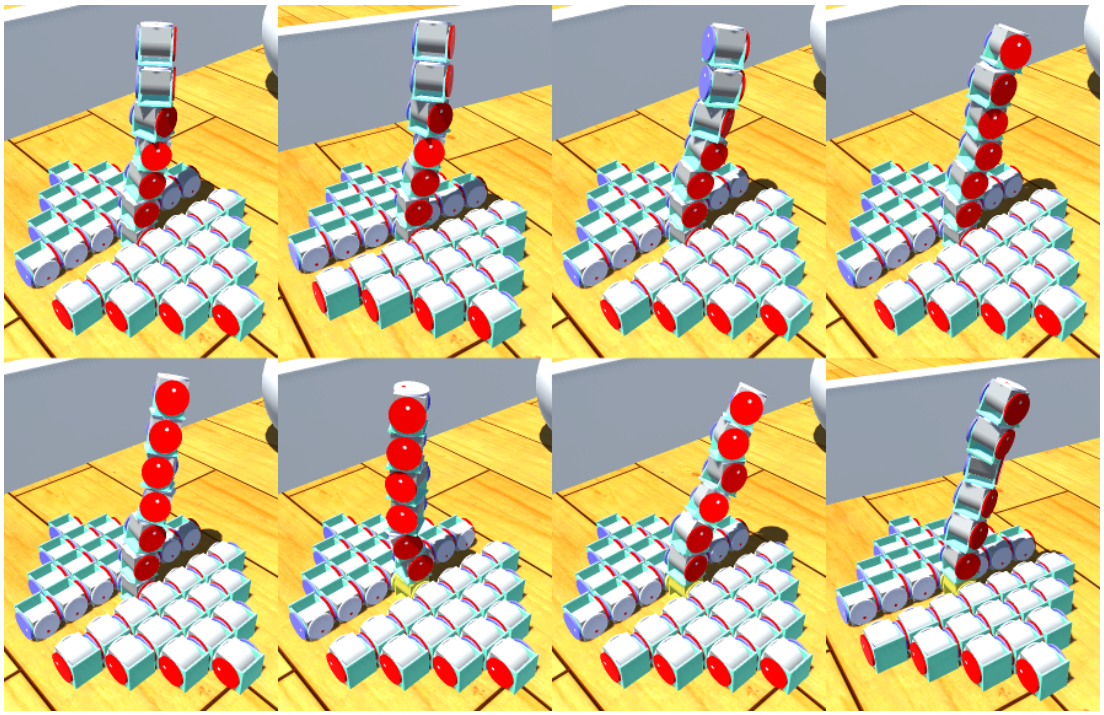


Figure 4.6: Module configurations used for algorithm evaluations

sensor module, and its actuators are not used; therefore six *moving* modules are present. A 3D depth sensor is simulated from the sensor module with 640 × 480 resolution. Captured 3D depth measurements are used to reconstruct the environment for qualitative evaluation.

The algorithms are evaluated using eight arbitrarily chosen sets of configurations (see Fig. 4.6). Note that these eight sets of configurations are for evaluation only, and not used for encoder biases estimation, encoder biases and connection offset estimation, or Gaussian process model. The 3D depth measurements and each module's noisy encoder measurements are stored along with the true position of all modules. An estimate of the sensor position, $p_{s,(\cdot)}$, is computed for each set of encoder measurements using each of the three proposed algorithms ((4.2) – (4.4)), and is used to project the 3D depth measurements in a global frame. Quantitative and qualitative results are studied to show the effective-

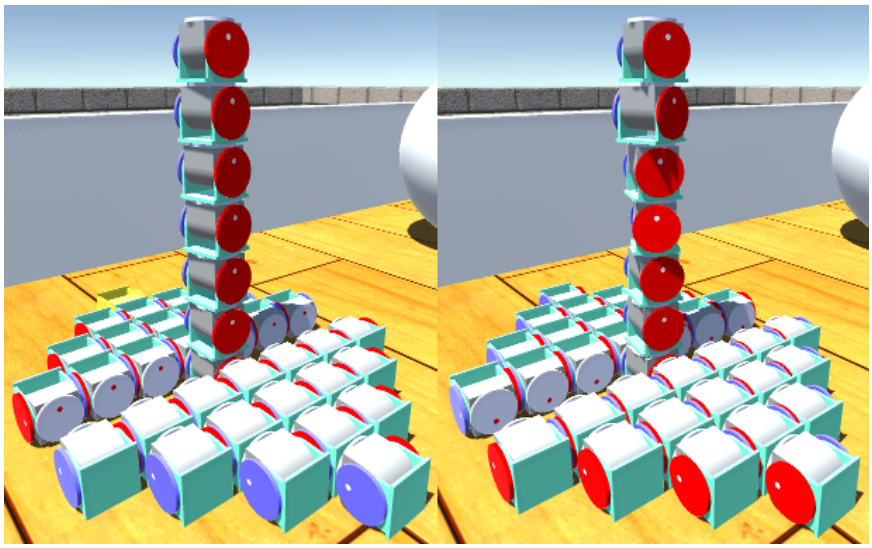


Figure 4.7: Configuration used for perfect connection (left) and imperfect alignment (right). Imperfect connections have position and orientation offsets in 2D plane.

ness of the proposed algorithms. Qualitative results compare the consistency of the reconstructed environment using the 3D depth measurements and the estimated positions from the three algorithms. Each 3D depth measurement has partial view of the environment. Therefore, the alignment of these depth measurements demonstrate the consistency and accuracy of the estimated sensor position. Estimated sensor position is directly compared to the true position for qualitative evaluation.

Encoder Bias with Perfect Connection

The goal of the first set of simulations is to estimate encoder biases, when the modules are perfectly aligned. Intuitively, all three algorithms should perform well because the encoder biases are observable and constant. The seven modules for the arm configuration are connected with no position and rotation offset

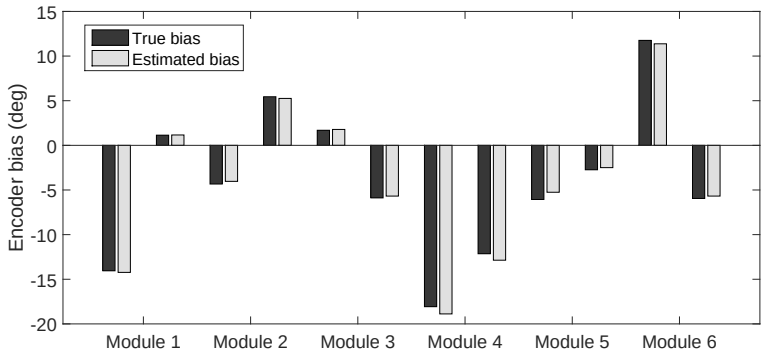


Figure 4.8: Encoder bias comparison for perfect connection. Each module shows two encoder values: body tilt angle ϕ (left) and pan angle, θ (right).

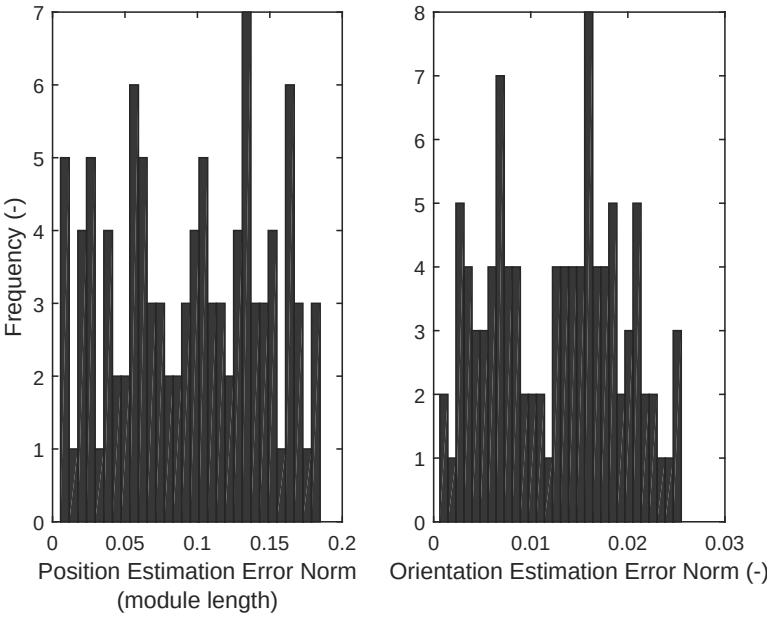


Figure 4.9: [Biases only case] Histogram of norm position estimation errors (left) and orientation estimation errors (right) of estimated sensor positions over 100 configurations using Algorithm 1. Norm of quaternion vector difference is used for orientation error comparison. The position estimation error is in module unit size.

in 2D plane (Fig. 4.7 left). The modules have made total of 100 configurations that are obtained by Monte Carlo observability analysis described in Section 4.3.1. Total of 100 sets of encoder measurements and noisy relative sensor position to the base of the configuration are collected, which are used to estimate the encoder biases. The simulated encoder biases are chosen within $[-20, 12]$ degree range, and added to the encoder measurements with noise.

Figure 4.8 shows the estimated biases of each module compared to the true bias values simulated. The body tilt angle ϕ (left) and pan angle θ (right) are shown for each module. Estimating the encoder biases using the collected measurements from the 100 configurations using (4.10), the encoder biases are accurately estimated. Algorithm 3 does not estimate the encoder biases directly.

Figure 4.9 shows a histogram of the predicted sensor position differences over the 100 configurations that were used for encoder biases estimation. The sensor positions are predicted using the encoder measurements with the estimated encoder biases using Algorithm 1, and the prediction error from the true positions are shown using a histogram. The norm 3D position and orientation estimation errors are presented. All position errors are lower than 0.2 module size. The norm of orientation estimation errors, which are computed as norm difference between the estimated and true orientation quaternion vector, are all within 0.03, which indicates within 3% orientation error.

The first row of Fig. 4.13 shows the environment reconstructed using the eight estimated sensor positions using Algorithm 1 (first row, third column) and 3 (first row, fourth column). Algorithm 2 is not used here because the connection offsets are not introduced here; however, Algorithm 2 will perform the same as Algorithm 1. With biases estimated, the sensor position is accurately

estimated (first row, third column), and the environment is very consistent with the truth (first row, first column). Using Gaussian process (Algorithm 3) (first row, fourth column), however, does not show as consistent alignments of the depth measurements as using encoder biases estimation (Algorithm 1). This result suggests that a parametric model performs better than a non-parametric model, when the parametric model accurately represents the system (such as, when there are primarily encoder biases errors).

Encoder Bias with Imperfect Connection

The goal of the second simulation study is to estimate both encoder biases and offsets (imperfect alignment). Intuitively, Algorithm 1 should perform poorly because the prediction function, f_b, does not consider connection offsets. However, Algorithm 2 and 3 should perform well. The connections between all seven modules are disturbed with position and rotation offsets, as shown in Fig. 4.7 right. Total of 200 different configurations were created, and encoder measurements and noisy relative sensor positions are collected for each configuration.

The connection offsets are constant, and thus are estimated using iterative nonlinear least-squares, as shown in Algorithm 2. All encoder biases and 2D position offsets are estimated (as shown in (4.13)) using the collected measurements. These results are computed using the sensor position prediction function, $f_{b,o}$, incorporating the estimated encoder biases and 2D connection offsets, $x_{b,o}$.

Figure 4.10 and 4.11 show the estimated encoder biases and connection offsets using Algorithm 2. Note that the encoder biases shown in Fig. 4.10 include

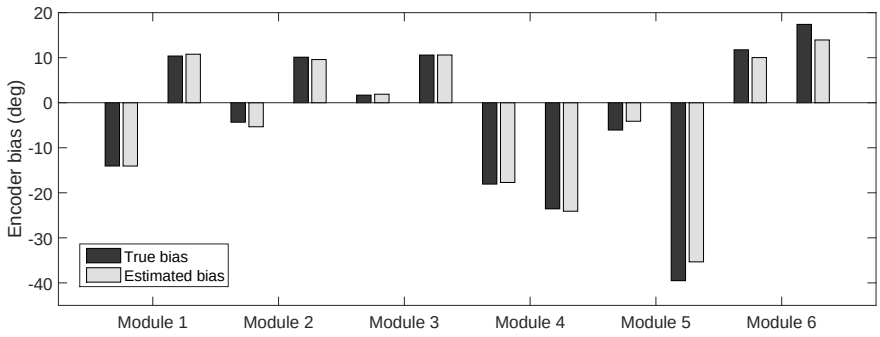


Figure 4.10: Encoder biases comparison for imperfect connection. Each module shows two encoder biases values: body tilt angle ϕ (left) and pan angle, θ (right). The pan angle biases shown include the orientation offsets.

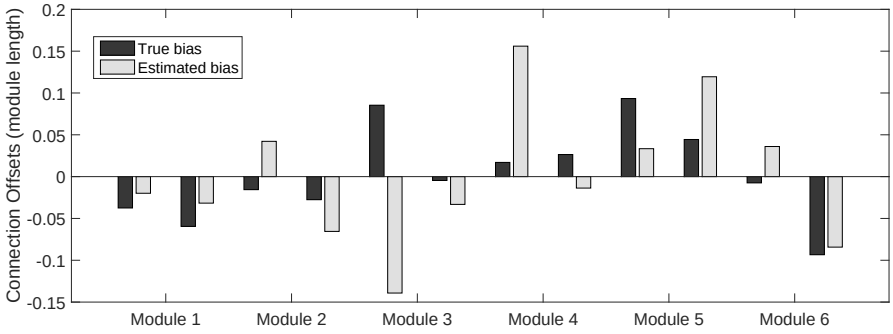


Figure 4.11: Connection offsets comparison for imperfect connection. Each module shows two connection offsets: Δx (left) and Δy (right).

the orientation offsets caused from imperfect connections. All encoder biases are accurately estimated using 200 configurations with about 5 degree maximum estimation error. The connection offsets are less accurately estimated with maximum approximately 0.25 module length errors. Algorithm 3 does not estimate the encoder biases connection offsets directly.

Figure 4.12 shows a histogram of the predicted sensor position differences over the 200 configurations that were used for encoder biases and connection offsets estimation. The sensor positions are predicted using the encoder mea-

surements with the estimated encoder biases and estimated connection offsets using Algorithm 2, and the prediction error from the true positions are shown using a histogram. All position errors are lower than 0.2 module size. The norm of orientation estimation errors are all within 0.04, which indicates within 4% orientation error.

The second row of Fig. 4.13 shows the environment reconstructed using estimated sensor positions for the eight different input configurations. The environment created with measured encoder data (with biases) (second row, second column) shows inconsistent left and right wall segments, and inconsistent segments of bench object (see Fig. 4.5 for reference). Once the estimated biases and offsets (Algorithm 2) are applied (second row, third column), the constructed scene matches well with the truth (second row, first column) and all segments are consistent to one another. The environment reconstructed using Gaussian process (Algorithm 3) (second row, fourth column) is also presented. However, left and right walls are not aligned as well as using Algorithm 2, and bench object also shows a misalignment. Similar to Algorithm 1, the sensor position prediction function $f_{b,o}$ represents the true system accurately given correct encoder biases and connection offsets. In such cases, a parametric model shows better estimation performance than using non-parametric model.

Encoder Bias, Imperfect Connection, and Scaled Encoder Measurements

The third set of simulations increases complexity by using encoder biases, position and rotation offsets for imperfect alignment. In addition, nonlinearities are simulated by scaling each module's encoder measurement with unknown factors. Although more complex nonlinear functions can be implemented, a

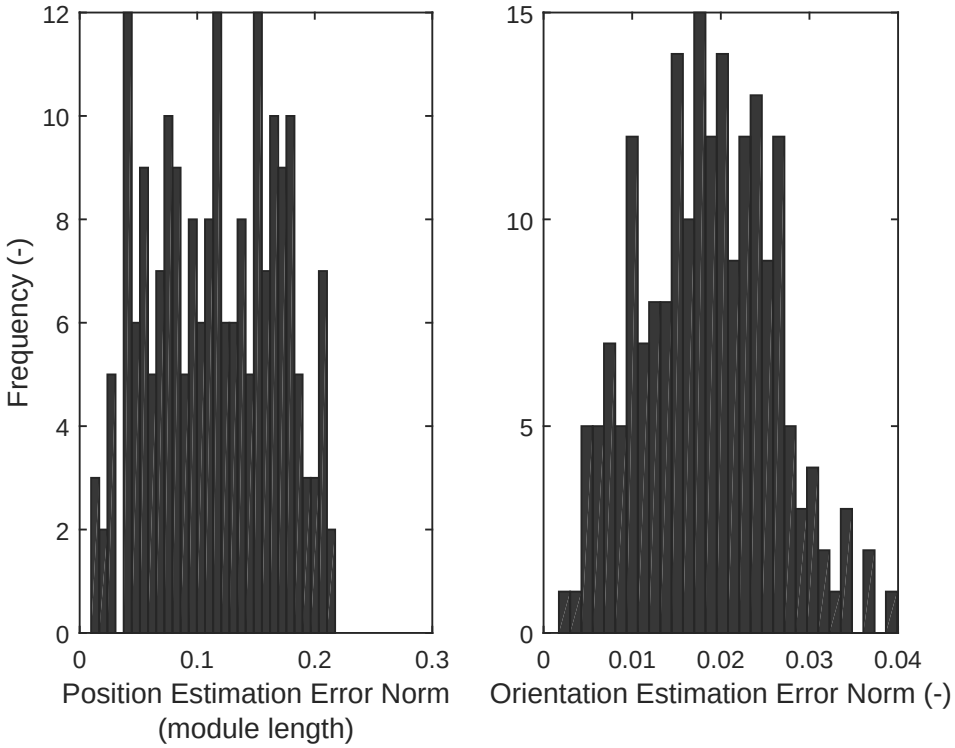


Figure 4.12: [Biases and imperfect connection case]Histogram of norm position estimation errors (left) and orientation estimation errors (right) computed for 200 sensor positions using Algorithm 2. Norm of quaternion vector difference is used for orientation error comparison. Note that the position estimation error is in module unit size.

simple scaled data already disturbs the encoder measurement enough that the output of the sensor position prediction function, $f_{(\cdot)}$, is highly inaccurate. The last row of Fig. 4.13 shows the reconstructed scene of the environment using Algorithm 2 (third row, third column), where encoder biases and connection offsets are estimated, and Algorithm 3 (fourth column), where a Gaussian process is used. The encoder measurements no longer possess significant meaning because the function $f_{(\cdot)}$ does not consider a scaling factor, and thus create poor environment representation (third row, second column). Consequently, estimating encoder biases and connection offsets does not show much significant

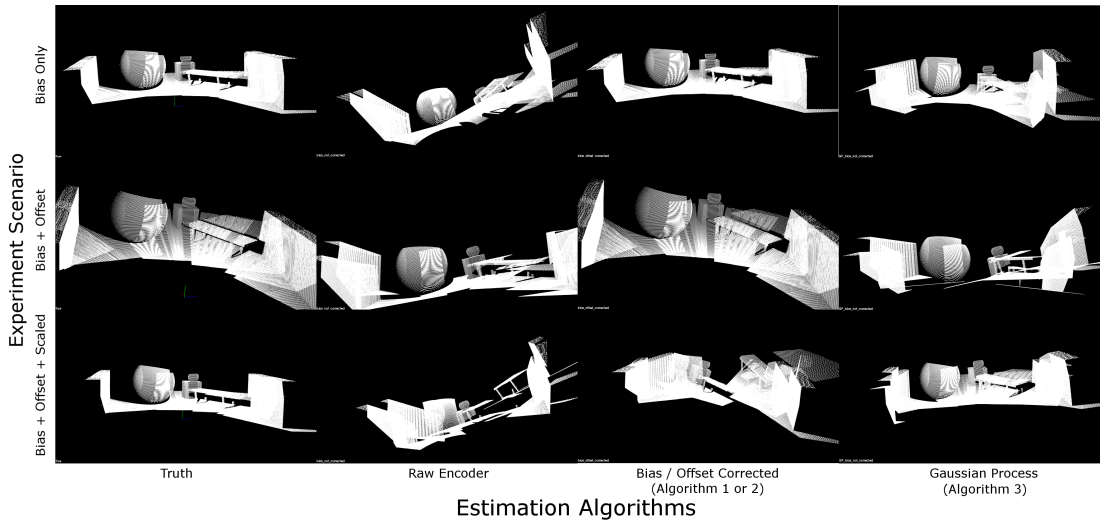


Figure 4.13: Environment reconstructed using sensor position estimates with different algorithms for comparison: {true, using only encoder measurements, using encoder biases (and offsets if applicable) estimation, Gaussian processes} from left to right.

improvement (third row, third column). Lastly, however, using Gaussian processes (third row, fourth column) shows significant improvement over the other two algorithms. Left and right wall segments show some imperfect alignments, but other objects such as spheres, a cube, and a bench show good alignments, compared to using Algorithm 2.

Figure 4.14 and 4.15 show the estimated encoder biases and connection offsets using Algorithm 2. Note that the encoder biases shown in Fig. 4.14 include the orientation offsets caused from imperfect connections. All encoder biases are poorly estimated using 200 configurations with about 45 degree maximum estimation error. The connection offsets are also poorly estimated with maximum approximately 1.2 module length errors. This is expected because the sensor position prediction function does not take account that the encoder measurements are scaled. Therefore, using an incorrect and inaccurate parametric model causes severe estimation errors.

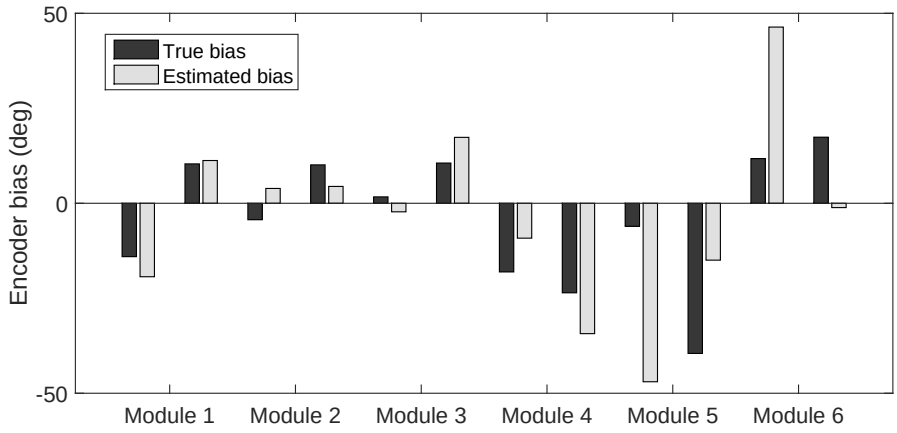


Figure 4.14: Encoder biases comparison for encoder biases, imperfect connection, and scaled encoder simulation. Each module shows two encoder biases values: body tilt angle ϕ (left) and pan angle, θ (right). The pan angle biases shown include the orientation offsets.

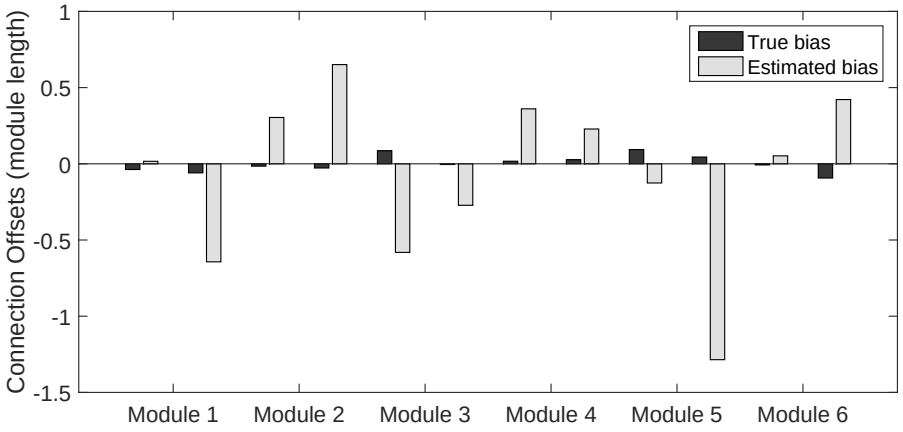


Figure 4.15: Connection offsets comparison for encoder biases, imperfect connection, and scaled encoder simulation. Each module shows two connection offsets: Δx (left) and Δy (right).

Figure 4.16 shows a histogram of the predicted sensor position differences over the 200 configurations that were used for Gaussian process model. The sensor positions are predicted using the encoder measurements with the estimated encoder biases and estimated connection offsets using Algorithm 2, and the prediction error from the true positions are shown using a histogram. Note that both position and orientation error are large; norm position estimation error spans over 2 module size, and norm orientation error spans over 0.35, which indicates 35% error. Algorithm 2 uses the sensor position prediction function $f_{b,o}$ to predict the sensor positions without realizing that the encoder measurements are scaled. Therefore, the sensor positions predicted using Algorithm 2 are incorrect with rather large amount of error. Unlike the two previous cases where the sensor position prediction functions, f_b and $f_{b,o}$ accurately describes the true system, these parametric models f_b and $f_{b,o}$ no longer represent the true system accurately. Even with the best estimates of encoder biases and/or connection offsets, the underlying parametric model hasn't changed and thus the estimated sensor positions are not accurate. In such case, using a non-parametric model – Gaussian process here – performs better because it depends on the data rather than a parametric model.

4.4.2 Experimental Results

An experiment was conducted using a set of ten SMORES modules to demonstrate the performance of the proposed algorithms. Figure 4.1 shows the four-legged configuration of the SMORES robot. Each leg consists of two SMORES modules and is connected to a passive module in the center which can be connected using all six faces; SMORES has five surfaces that can be connected. Two

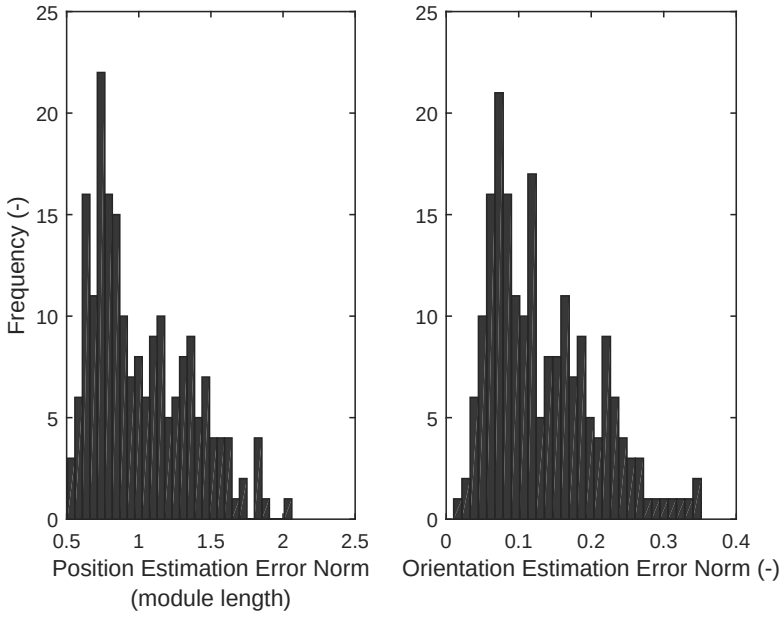


Figure 4.16: [Bias, Imperfect connection, and scaled case] Histogram of norm position estimation errors (left) and orientation estimation errors (right) computed for 200 sensor positions using Algorithm 2 (biases and offset estimation). Norm of quaternion vector difference is used for orientation error comparison. The position estimation error is in module unit size.

additional modules are connected to the top of the passive module to act as an arm. The Intel RealSense R200 sensor is attached at the top module's face and is used as a main external sensor of the robot. The R200 is a RGB-D sensor which provides depth images using stereo IR vision (640×480 resolution with $70 \times 59 \times 46$ degree of field of view), and high-resolution color images (1920×1024 resolution).

Figure 4.17 shows the experiment setup. A box with one AprilTag [52] attached on each side is used as an environmental object to estimate the shape, and also provides external environmental measurements. An AprilTag measurement provides relative position and orientation measurements of the tag

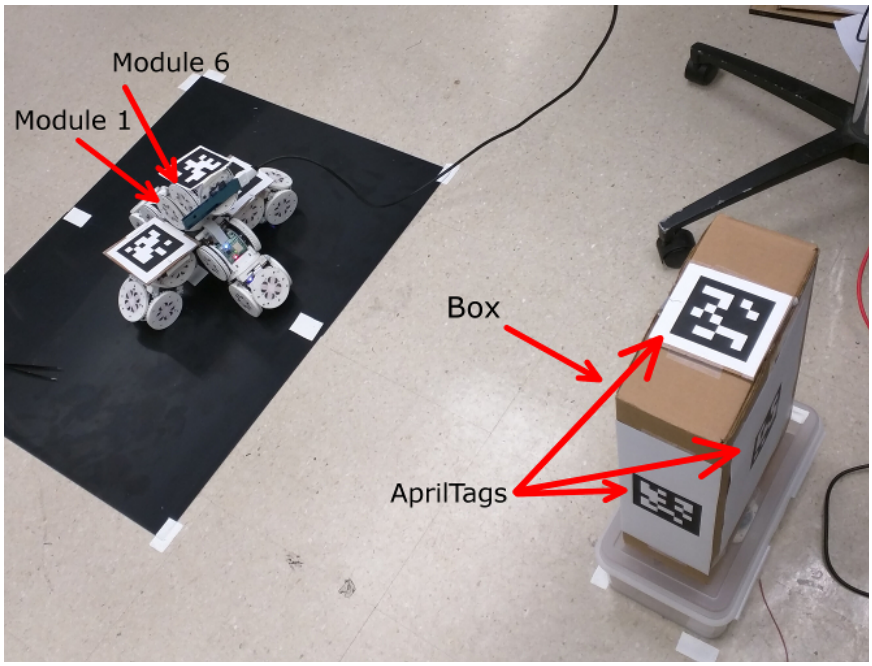


Figure 4.17: Experiment setup

with respect to the camera.

The modular robot's arm portion was moved in a pre-defined way, creating measurements to the external box object in different viewpoints. The arm portion moved to $\{-20, 0, 20\}$ degree horizontally by commanding Module 1's pan angle, while commanding Module 6's tilt angle to $\{-20, 0, 20\}$ degree at the same time; the arm portion moved to all possible nine positions. These movements were repeated four times resulting 36 attempts. However, 18 of the positions did not have valid AprilTag measurements because the tag was out of R200's camera view; only valid ones are selected (22 of them). The AprilTags on the robot (Fig. 4.17) are not used.

All three algorithms (Algorithm 1, 2, and 3) are evaluated within this experiment. The encoder biases, and encoder biases and connection offsets of the arm portion are estimated using Algorithm 1 and 2, respectively, with the 22

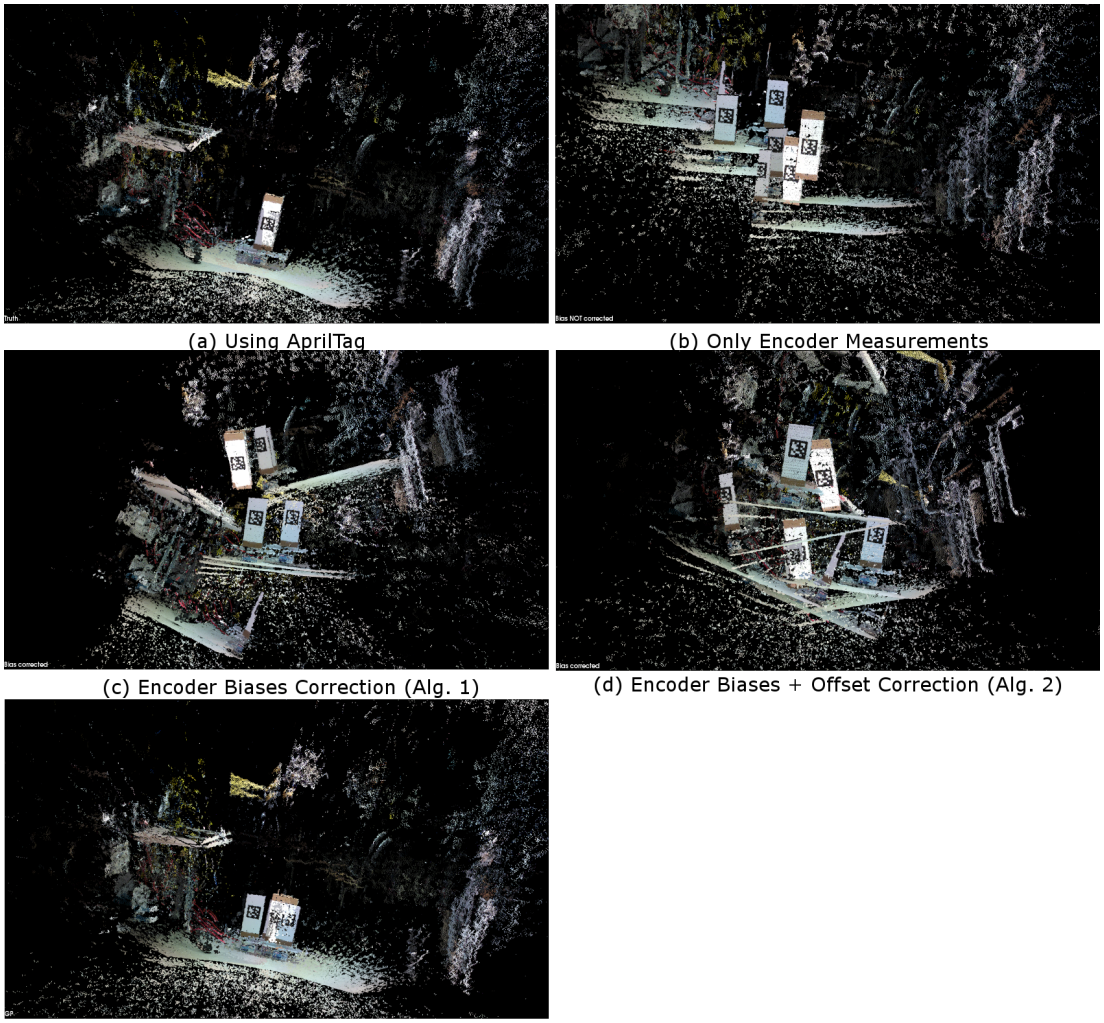


Figure 4.18: Qualitative comparison of the estimates of an environment using different methods of predicting the sensor position. The experiment environment is constructed using the sensor position estimated using {AprilTag measurements, only encoder measurements, with encoder biases correction (Algorithm 1), with encoder biases and offset correction (Algorithm 2), Gaussian process (Algorithm 3)}.

measurements. The estimated encoder biases and connection offsets are used to predict the relative sensor position of the robot again using Algorithm 1 and 2. The same measurements are used for Gaussian process (Algorithm 3).

Figure 4.18 shows the environment reconstructed using 11 testing module configurations with R200 measurements (colored point clouds) and the estimated sensor positions using different algorithms. Figure 4.18 (a) shows the best possible scene created using AprilTag measurements, which provide the R200's positions and orientations for each testing configuration. All 11 point clouds are aligned well to create one consistent box object and the surrounding environment. The scenes reconstructed using only encoder measurements (no biases estimation) (Fig. 4.18 (b)) shows fairly inconsistent alignment, where multiple parts of the box object and the surrounding environments are scattered. Similarly, however, the scene reconstructed using the sensor positions using corrections via encoder biases estimation (Algorithm 1) (Fig. 4.18 (c)), and the sensor positions using corrections via encoder biases and connection offset estimation (Algorithm 2), do not show consistent alignments of R200 measurements as well. Most of 11 color point clouds are rotated with incorrect orientations, and thus the point clouds showing the box object and surrounding environment are poorly aligned. Figure 4.18 (e) shows the environment reconstructed using the sensor positions estimated with Gaussian process (Algorithm 3). The scene created using the estimated function, $\hat{f}(u_e)$, via Gaussian process shows relatively good alignments between the scenes, and look similar to the best possible scene as shown in Fig. 4.18 (a).

In order to quantify the accuracy of the estimated sensor positions, relative position differences of the estimated sensor positions are compared with that

	Position (m)	Orientation (-)
Using Only Encoder Measurements	0.3594 ± 0.1348	0.4149 ± 0.1534
Encoder Biases Correction (Algorithm 1)	0.4451 ± 0.1594	0.3795 ± 0.1387
Encoder Biases & Offset Correction (Algorithm 2)	0.4007 ± 0.1138	0.3977 ± 0.1485
Gaussian process (Algorithm 3)	0.1213 ± 0.0625	0.0866 ± 0.0440

Table 4.1: Relative position and orientation error

of using AprilTag measurements, which serve as the truth. The norm 3D position difference and the norm orientation difference in quaternion are used to represent the accuracy. Table 4.1 shows the mean and standard deviation of the relative camera position difference for using all three different estimation algorithms. The positions estimated using Gaussian process (Algorithm 3) are three times more accurate than using only encoder measurements, whereas the orientation is about five times more accurate than using only encoder measurements. The differences in orientation especially play a significant role in aligning the point clouds, and thus the impact on the key task of environment estimation is high. Using Algorithm 1 and 2 do not show position improvement over using only encoder measurements. There exist little improvement of using Algorithm 1 and 2 for orientation estimation. The orientation differences using Algorithm 1 and 2 show rather a large magnitude of the orientation error, while the results using Gaussian process (Algorithm 3) show significantly better accuracy; the mean of the orientation difference is four times more accurate, and the standard deviation is three times smaller.

If given a complex modular robot shape, which becomes very difficult to analytically model the system accurately, using parametric models such as f_b and $f_{b,o}$ in Algorithm 1 and 2 suffer due to the models' incapability to accurately and correctly represent the system. In addition, physical hardware, especially when manufacturing processes are inconsistent, cause difficulty in modeling;

each module can have different encoder scaling factor (as shown in Fig. 4.2), or different degree of nonlinearities. In this experiment, both Algorithm 1 and 2 do not show improvement over using only encoder measurements, while Algorithm 3 shows improvement over using only encoder measurements. This suggests that the parametric model that was built to express the modular robot's shape may be inaccurate, but the non-parametric model based on the observed data performs well.

4.4.3 Without Using Visual Fiducial System

A visual fiducial system is not always available. In such cases, estimating relative movement of modules (and thus biases) must be done using the measurements of external objects assuming they are stationary. To consider this problem, a 3D point cloud (depth measurements) registration algorithm from Point Cloud Library [62] is used; this registration algorithm estimates the relative transformation between two point clouds using RANSAC [21].

Total of 15 sets of point clouds along with the encoder measurements were collected for estimating encoder biases and Gaussian process model without using AprilTags. Figure 4.19 shows the environment reconstructed using 11 R200 measurements (colored point clouds) and the estimated sensor positions without using AprilTags. Figure 4.19 (a) shows the best possible scene using AprilTag measurements of the module positions and orientations (labeled truth). The scene created using only encoder measurements (Fig. 4.19 (b)) shows inconsistent alignments; multiple parts of the box object and the surrounding environments are scattered. The scene reconstructed using the sensor position estimates

using the encoder measurements with correction via encoder biases estimation (Fig. 4.19 (c)) shows improved alignments of the R200 measurements compared to that of using only encoder measurements. Similarly, Fig. 4.19 (d) shows the R200 measurements using Algorithm 2. Although the R200 measurements look consistent, the one of top module's offsets are estimated to about 25m on the connection surface. Figure 4.19 (e) shows the scene reconstructed using the sensor positions estimated using Gaussian process, which shows good alignments.

Qualitatively, the environments reconstructed using Algorithm 1 and 2 show reasonable consistencies in terms of the R200 depth measurement alignments. The environment, however, such as table and yellow cabinet boxes, shows multiple segments. The results using Gaussian process (Algorithm 3) show the most consistent alignment performance; the table and yellow cabinet left to the box object are aligned well. The alignment of 11 R200 depth measurements using Gaussian process is not perfect as compared to the true alignment (Fig. 4.19 (a)), but shows the closest scene reconstructed to the true environment.

Table 4.2 shows the mean and standard deviation of relative sensor position differences compared to the true sensor positions (using AprilTags). The mean and standard deviation of 3D position and orientation norms are computed for all 11 testing sensor positions. Using Gaussian process (Algorithm 3), the position difference is about two times smaller than those of using Algorithm 1 or 2. In terms of the orientation estimates, the Algorithm 3 also shows better estimation errors over the other two algorithms, but not significantly. This could be due to the challenges of measuring the environmental elements.

The quality of external measurements (to the stationary box object in this case) determines the performance of estimator, especially when the number of

	Position (m)	Rotation (-)
Using Only Encoder Measurements	0.3594 ± 0.1348	0.4149 ± 0.1534
Encoder Biases Correction (Algorithm 1)	0.2694 ± 0.1157	0.4144 ± 0.1564
Encoder Biases & Offset Correction (Algorithm 2)	0.2382 ± 0.1124	0.4252 ± 0.1613
Gaussian process (Algorithm 3)	0.1357 ± 0.0607	0.3766 ± 0.1321

Table 4.2: Relative position and rotation error without using a visual fiducial system

measurement is relatively low; 100 or 200 sets of measurements in simulation, while 22 or 11 sets in experiment. Using a visual fiducial system, AprilTags, provides highly accurate measurements. Using a registration method with point clouds, however, provides less accurate and noisy data. Registration performance depends on the tidiness of the environment, possible occlusion, sensor noise, and environmental changes. Comparing Table 4.1 and 4.2, it is evident that the performance of Gaussian process using a visual fiducial system is superior. The position estimate differences are about the same. However, the orientation estimates using a visual fiducial system perform at least four times better. This could be potentially caused by the smaller number of measurements as well when not using a visual fiducial system.

Considering the quality of external measurements using registration is not as good as using a fiducial system, the performance of using Gaussian process is reasonably good, especially compared to using only encoder measurements. The consistency and accuracy of the sensor position estimates using the Gaussian process model (or Gaussian Process algorithm) is better than encoder biases correction method, and encoder biases and connection offset correction method. Improved performance is expected once the measurements can be refined to provide less noisy data, and a large number of measurement sets become available.

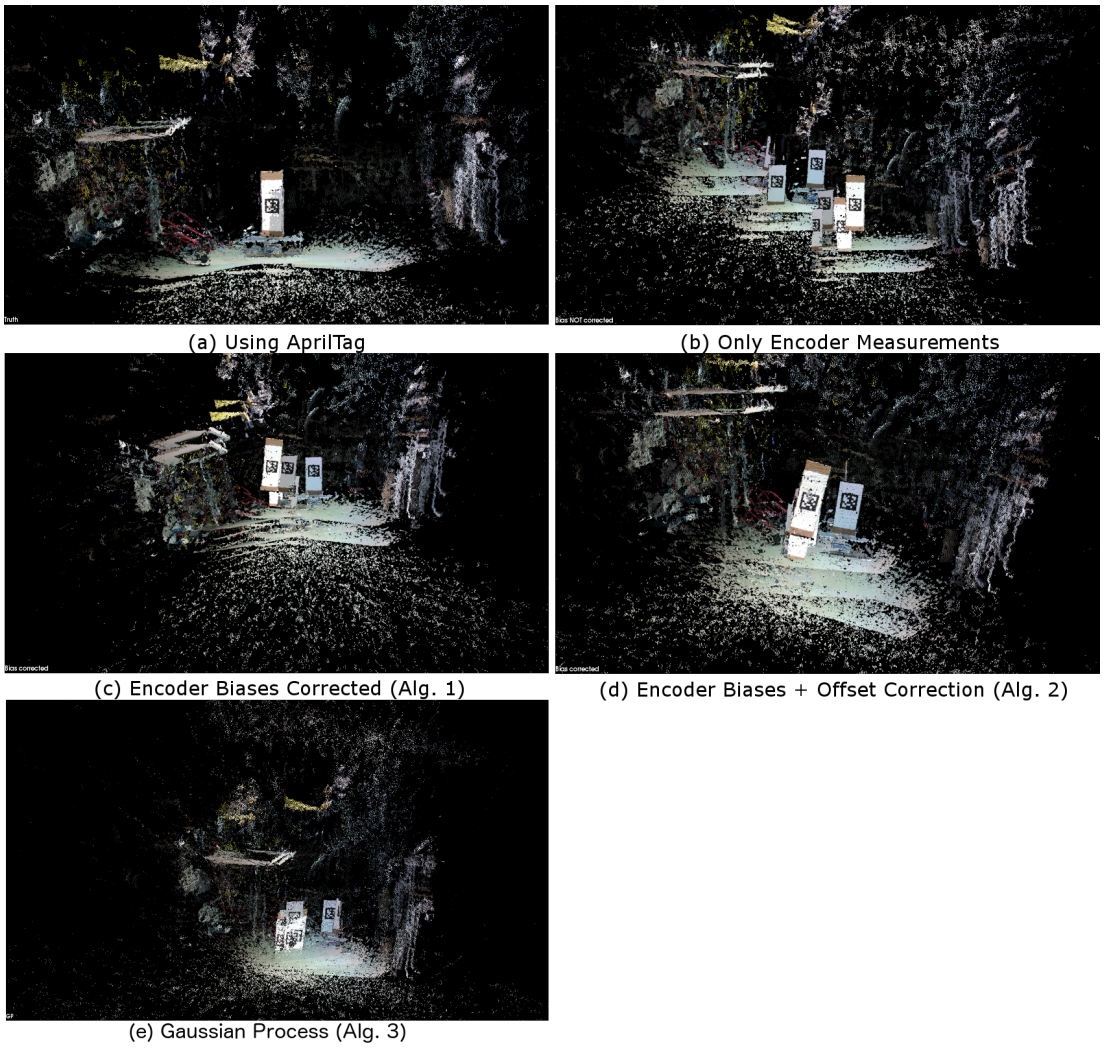


Figure 4.19: Qualitative comparison of using different methods of predicting the sensor position. The experiment environment is recreated using the sensor position estimated using {true, raw encoder data, prediction using bias correction, Gaussian process} without using a visual fudicial system.

These experiments using SMORES modules possess all of the suggested challenges using modular robots; the true sensor position prediction function is not easy to model, connections are not perfect, and mapping of the encoder values is nonlinear and has biases (Fig. 4.2). The use of a Gaussian process model could overcome these challenges better than parametric models using encoder biases estimation, when encoders with nonlinearities or other nonlinearities are present.

The Gaussian process model is recently developed, and very active research is ongoing. The inability to estimate for multiple outputs currently is a limitation and tends to make the estimates conservative, because it does not take account of the correlations between the multiple outputs for training. This is rather critical because 3D position and orientation are usually highly correlated. The estimation performance using Gaussian process for the challenges presented here is expected to improve significantly once multiple outputs are considered.

4.5 Conclusion

A set of bias and uncertainty estimation methods for modular robots using internal and external sensor measurements is presented. Estimation of bias and uncertainties is important in order to perform high-level perception tasks with confidence and consistency. Modular robots can suffer from manufacturing inconsistencies, potential encoder biases, nonlinearities in the encoder, imperfect connections between modules, and other external forces such as bending and twisting from gravity. Three estimation algorithms/formulations are presented;

a nonlinear iterative least-squares approach for encoder biases estimation, a nonlinear iterative least-squares approach for encoder biases and connection offsets, and Gaussian process are presented. These algorithms are validated in both simulation and an experiment using SMORES modular robots. In simulation, a long-arm configuration is studied. Constant encoder biases and offsets are accurately estimated using sensor position prediction model and 100 sets of encoder measurements and measured sensor position information. When nonlinearities and external forces are present, the errors could not be captured by least-squares approaches because the underlying parametric model to represent the system no longer captures the errors; however, the results using the GP model are better. Experimental results on the SMORES robots show the encoder biases estimation does not improve the performance of sensor position estimation. Using Gaussian processes, on the other hand, provide consistent sensor position estimates that reconstruct the environment well.

Modeling a module configuration well is critical to estimate encoder biases and connection offsets. From these simulation and experiment studies, it is shown that having a good parametric model that describes the system's dynamic correctly and accurately can handle the encoder biases and connection offset problems well. A non-parametric model, which does not consider the underlying physics model and solely estimates based on available data, may not perform as well as the parametric model. However, expressing a module configuration's dynamics as a parametric model can become difficult if the configuration consists of many complex connections. In addition, if unknown nonlinear mapping and varying external forces exist, modeling the true system with an accurate parametric model can become very challenging.

CHAPTER 5

CONCLUSION AND CONTRIBUTIONS

A few estimation methods are presented for two different types of robotics platforms. Large and complex platforms suffer from highly nonlinear dynamics and measurement models with non-Gaussian multi-modal noises. Chapter 2 presents an adaptive Gaussian mixture smoothing algorithm, which adaptively changes the number of mixands as the degree of nonlinearity changes. Experimental results successfully demonstrate that the smoothed estimates of using the adaptive Gaussian mixture smoother show significant improvement over the forward filtered estimates. Also, as the number of maximum number of mixands to use for Gaussian mixture increases, the accuracy and consistency of smoothed estimates improve as well.

List of contributions:

- Development of nonlinear Gaussian mixture model smoothing algorithm via linearization; a method to estimate the linearized backward corrector between two time steps is developed to reduce computational complexity while maintaining performance.
- Development of an adaptive Gaussian mixture model smoother using a tree structure; Children collapsing and parent splitting methods are developed to handle varying number of nodes and updating the weights of mixands properly
- Numerical study of estimated backward corrector's accuracy and consistency in simulation using univariate nonstationary growth model. The estimated backward corrector's probability distribution is compared to the

true backward corrector.

- Large scale experiment in application of robot navigation where measurements are sparse; both quantitative and qualitative mapping and localization performance of two proposed methods are compared to the true estimates of the environment and robot position. The effect of the number of maximum mixands for the adaptive Gaussian mixture smoother is evaluated in mapping and localization performance.

A multi-resolution scalable and efficient surface normal estimation method is presented in Chapter 3 for computationally limited platforms. The capability of estimating surface normal vectors efficiently in multiple resolutions enables robust and fast high-level object recognition, classification, and segmentation. Utilizing Octree representation, two multi-resolution surface normal estimation methods show comparable results to that of using dense 3D point clouds in terms of accuracy and segmentation performance, while reducing computational needs significantly. Simulation and experimental results demonstrate the scenes and objects being accurately segmented using the proposed algorithms, with multiple order of magnitude reduction in computation time compared to that of using point clouds.

List of contributions:

- Development of probabilistic framework to estimate surface normal vectors which uses a discrete representation instead of point clouds to enable scalable and computationally tractable implementation
- Incorporation of occupancy grid representation which enables easy implementation and ease of sharing information among other algorithms be-

cause of its common use

- Development of two surface normal estimation formulations: recursive and batch formulations
- Development and validation via a numerical study of sufficient measurement assumption which reduces the computational need by orders of magnitude
- Development of two probabilistic frameworks to infer surface normal vectors in multiple resolutions while leveraging the Octree structure
- Study of estimated surface normal vectors' accuracy and consistency compared to the truth (when available) and using dense point clouds in Gazebo 3D simulation; the surface normal vectors' accuracy and computational cost between using proposed estimation algorithms and using principal component analysis of dense point clouds are studied in multiple noise levels and resolutions. A high-level object segmentation task in multiple noise levels and resolutions for the proposed methods are successfully demonstrated.
- Study of proposed algorithms' performance in multiple resolutions using experimental data

Lastly, estimation methods to tackle modular robots' inherent challenges from making connections are presented in Chapter 4. For their capability to self-reconfigure, modular robots suffer from encoder biases and nonlinear mapping, imperfect connections between modules, material compliance and other external forces. These challenges are solved using iterative nonlinear least-squares and using Gaussian process. The simulation and experimental results show that the parametric models for a given configuration can estimate the position and

orientation of sensor modules if biases are correctly captured, and the prediction model represents the true system accurately. However, if given nonlinear mapping of encoder data and other external forces, these parametric models cannot estimate the sensor positions well. Using Gaussian process given enough sets of measurements, however, the sensor positions and orientations are accurately and consistently estimated. Gaussian process also provides a benefit that a parametric model is unnecessary because the correlations among training set of measurements can provide good estimates for an arbitrarily given input. Therefore, Gaussian process is recommended to be used for complex configurations where accurate parametric models are difficult to obtain.

List of contributions:

- Identify and solve the inherent estimation challenges in using modular robots
- Formulate and solve encoder biases estimation problem using iterative nonlinear least-squares method
- Formulate and solve problems of nonlinear encoder mapping and other external forces using Gaussian process
- Development of an automatic calibration procedure for modular robots so that the sensor positions are accurately estimated
- Study the encoder biases, connection offset, and nonlinear mapping problems in both simulation and experiment using SMORES modular robots
- Demonstrate that the automatic calibration procedure can provide accurate and consistent sensor position estimates using a visual fiducial system

- Study the performance of automatic calibration procedure without using a visual fiducial system; uses raw point cloud data from sensors

All estimation methods presented cover wide variety of topics – state estimation for highly complex nonlinear systems using smoothing algorithm, accurate and scalable multi-resolution surface normal estimations for computationally limited platforms, and a non-parametric and data-driven model for platforms that are difficult to model. While these methods are motivated by two different types of robotics platforms, but can be applied to any other platforms.

BIBLIOGRAPHY

[1] D. Alspach and H. Sorenson. Nonlinear bayesian estimation using gaussian sum approximations. 17(4):439–448, August 1972.

[2] Byoung Kwon An. Em-cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3149–3155, May 2008.

[3] Brian D. O. Anderson and John B. Moore. *Optimal Filtering*, pages 37–61. Dover Publications, January 2005.

[4] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Signal Processing, IEEE Transactions on*, 50(2):174–188, 2002.

[5] A. Asif. Fast rauch-tung-striebel smoother-based image restoration for noncausal images. *Signal Processing Letters, IEEE*, 11(3):371–374, 2004.

[6] Y Bar-Shalom. *Estimation With Applications to Tracking and Navigation*, pages 55–56. Wiley, 2001.

[7] Yaakov Bar-Shalom, Thiagalingam Kirubarajan, and X.-Rong Li. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[8] Christopher Bishop. *Pattern Recignotion and Machine Learning*. Springer, 2006.

[9] Shelby Brunke and Mark Campbell. Square root sigma point filtering for real-time, nonlinear estimation. *Journal of Guidance, Control, and Dynamics*, 27(2):314–317, March 2004.

[10] Owen Carmichael, Daniel Huber, and Martial Hebert. Large data sets and confusing scenes in 3-d surface matching and recognition. In *3-D Digital Imaging and Modeling, 1999. Proceedings. Second International Conference on*, pages 358–367. IEEE, 1999.

[11] T. Darom and Y. Keller. Scale-invariant features for 3-d mesh models. *Image Processing, IEEE Transactions on*, 21(5):2758–2769, May 2012.

[12] J. Davey, N. Kwok, and M. Yim. Emulating self-reconfigurable robots - design of the smores system. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4464–4469, Oct 2012.

[13] Kyle J DeMars, MK Jah, Y Cheng, and RH Bishop. Methods for splitting gaussian distributions and applications within the aegis filter. In *Proceedings of the 22nd AAS/AIAA Space Flight Mechanics Meeting,(Charleston, SC)*, 2012.

[14] H Quynh Dinh and Steven Kropac. Multi-resolution spin-images. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 863–870. IEEE, 2006.

[15] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[16] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12:656–704, 2009.

[17] F. Faubel and D. Klakow. An adaptive level of detail approach to nonlinear estimation. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 3958–3961, March 2010.

[18] F. Faubel, J. McDonough, and D. Klakow. The split and merge unscented gaussian mixture filter. *Signal Processing Letters, IEEE*, 16:786–789, 2009.

[19] Friedrich Faubel and Dietrich Klakow. Further improvement of the adaptive level of detail transform: Splitting in direction of the nonlinearity. In *Proceedings of the 18th European Signal Processing Conference (EUSIPCO)*, pages 850–854, August 2010.

[20] M. Fiala. Artag, a fiducial marker system using digital techniques. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 590–596 vol. 2, June 2005.

[21] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[22] T. Furuyama and M. Niitsuma. Building a multi-resolution map for au-

tonomous mobile robot navigation in living environments. In *Cognitive Infocommunications (CogInfoCom), 2013 IEEE 4th International Conference on*, pages 261–266, Dec 2013.

[23] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, 140:107–113, April 1993.

[24] D. Hahnel, W. Burgard, D. Fox, and S. Thrun. An efficient fastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings*, volume 1, pages 206–211 vol.1, 2003.

[25] F. Havlak and M. Campbell. Discrete and continuous, probabilistic anticipation for autonomous robots in urban environments. *Robotics, IEEE Transactions on*, PP(99):1–14, 2013.

[26] Frank Havlak and Mark Campbell. Discrete and continuous, probabilistic anticipation for autonomous robots in urban environments. In *Proc. SPIE*, volume 7833, pages 78330H–78330H–13, 2010.

[27] Stefan Holzer, Radu Bogdan Rusu, M Dixon, Suat Gedikli, and Nassir Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *IROS*, pages 2684–2689. IEEE, 2012.

[28] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[29] Andrew E Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(5):433–449, 1999.

[30] Krzysztof Jordan and Philippos Mordohai. A quantitative evaluation of surface normal estimation in point clouds. In *IROS*, pages 4220–4226. IEEE, 2014.

[31] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[32] John L. Junkins. von karman lecture: Adventures on the interface of dynamics and control. *Journal of Guidance, Control, and Dynamics*, 20(6):1058–1071, Nov 1997.

[33] S. Khan, D. Wollherr, and M. Buss. Adaptive rectangular cuboids for 3d mapping. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2132–2139, May 2015.

[34] Genshiro Kitagawa. The two-filter formula for smoothing and an implementation of the gaussian-sum smoother. *Annals of the Institute of Statistical Mathematics*, 46(4):605–623, 1994.

[35] Klaas Klasing, Daniel Althoff, Dirk Wollherr, and Martin Buss. Comparison of surface normal estimation methods for range sensing applications. In *ICRA*, pages 3206–3211. IEEE, 2009.

[36] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IROS*, volume 3, pages 2149–2154 vol.3, Sept 2004.

[37] Gerhard K. Kraetzschmar, Guillem Pags Gassull, Klaus Uhl, Guillem Pags, and Gassull Klaus Uhl. Probabilistic quadtrees for variable-resolution mapping of large environments. In *Eds.), Proceedings of the 5th IFAC/EURON*, 2004.

[38] A. J. Krener and K. Ide. Measures of unobservability. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 6401–6406, Dec 2009.

[39] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, Mar 1951.

[40] Daniel J. Lee and Mark E Campbell. An efficient probabilistic surface normal estimator. In *ICRA*, 2016, in print.

[41] D.J. Lee and M.E. Campbell. Iterative smoothing approach using gaussian mixture models for nonlinear estimation. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2498–2503, Oct 2012.

[42] Xinju Li and Igor Guskov. Multiscale features for approximate alignment

of point-based surfaces. In *Symposium on geometry processing*, volume 255, pages 217–226. Citeseer, 2005.

[43] Ming Liu, François Pomerleau, Francis Colas, and Roland Siegwart. Normal estimation for pointcloud using gpu based sparse tensor voting. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 91–96. IEEE, 2012.

[44] Takeshi Masuda. Log-polar height maps for multiple range image registration. *Computer Vision and Image Understanding*, 113(11):1158–1169, 2009.

[45] Oliver Mattausch, Daniele Panozzo, Claudio Mura, Olga Sorkine-Hornung, and Renato Pajarola. Object detection and classification from large-scale cluttered indoor scans. *Computer Graphics Forum*, 33(2):11–21, 2014.

[46] Donald Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129 – 147, 1982.

[47] I. Miller and M. Campbell. Rao-blackwellized particle filtering for mapping dynamic environments. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3862–3869, April 2007.

[48] Isaac Miller, Mark Campbell, Dan Huttenlocher, Frank-Robert Kline, Aaron Nathan, Sergei Lupashin, Jason Catlin, Brian Schimpf, Pete Moran, Noah Zych, Ephrahim Garcia, Mike Kurdziel, and Hikaru Fujishima. Team cornell's skynet: Robust perception and planning in an urban environment. *Journal of Field Robotics*, 25(8):493–527, 2008.

[49] Michael Montemerlo and Sebastian Thrun. A multi-resolution pyramid for outdoor robot terrain perception. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, pages 464–469. AAAI Press, 2004.

[50] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[51] N. Nadarajah, R. Tharmarasa, M. McDonald, and T. Kirubarajan. IMM forward filtering and backward smoothing for maneuvering target tracking. *Aerospace and Electronic Systems, IEEE Transactions on*, 48(3):2673–2678, 2012.

[52] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. In

Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pages 3400–3407. IEEE, May 2011.

[53] R. Oung, F. Bourgault, M. Donovan, and R. D'Andrea. The distributed flight array. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 601–607, May 2010.

[54] Giuliano Pasqualotto, Pietro Zanuttigh, and Guido M Cortelazzo. Combining color and shape descriptors for 3d model retrieval. *Signal Processing: Image Communication*, 28(6):608–623, 2013.

[55] N. D. Powel and K. A. Morgansen. Empirical observability gramian rank condition for weak observability of nonlinear systems with control. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 6342–6348, Dec 2015.

[56] Mark Psiaki. The super-iterated extended kalman filter. American Institute of Aeronautics and Astronautics, Aug 2004.

[57] Mark Psiaki. Backward-smoothing extended kalman filter. *Journal of Guidance, Control, and Dynamics*, 28:885–894, September 2005.

[58] Mark L. Psiaki and Massaki Wada. Derivation and simulation testing of a sigma-points smoother. *Journal of Guidance, Control, and Dynamics*, 30(1):78–86, 2007.

[59] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[60] HE Rauch, CT Striebel, and F Tung. Maximum likelihood estimates of linear dynamic systems. *Journal of the American Institute of Aeronautics and Astronautics*, 3(8):1445–1450, 1965.

[61] Modular Robotics. *Modular Robotics — Moss & Cubelets — Robot Toys*, [Online] http://www.modrobotics.com, 2012 (accessed May 18, 2016).

[62] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *ICRA*, Shanghai, China, May 9-13 2011.

[63] Julian Ryde and Huosheng Hu. 3d mapping with multi-resolution occupied voxel lists. *Auton. Robots*, 28(2):169–185, February 2010.

[64] S. Sarkka, V.V. Viikari, M. Huusko, and K. Jaakkola. Phase-based UHF RFID tracking with nonlinear kalman filtering and smoothing. *Sensors Journal, IEEE*, 12(5):904–910, 2012.

[65] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. Narf: 3d range image features for object recognition. In *IROS*, volume 44, 2010.

[66] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. Point feature extraction on 3d range scans taking into account object boundaries. In *ICRA*, pages 2601–2608. IEEE, 2011.

[67] Fridtjof Stein and Gérard Medioni. Structural indexing: Efficient 3-d object recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):125–145, 1992.

[68] Y. Sun and M.A. Abidi. Surface matching by 3d point's fingerprint. In *ICCV 2001*, volume 2, pages 263–269 vol.2, 2001.

[69] Unity Technologies. *Unity - Game Engine*, [Online] http://unity3d.com, 2004 (accessed May 18, 2016).

[70] Gabriel Terejanu. An adaptive split-merge scheme for uncertainty propagation using gaussian mixture models. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. American Institute of Aeronautics and Astronautics, Jan 2011.

[71] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[72] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*, pages 39–79. The MIT Press, August 2005.

[73] R. Tse, N.R. Ahmed, and M. Campbell. Unified terrain mapping model with markov random fields. *Robotics, IEEE Transactions on*, 31(2):290–306, April 2015.

[74] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001.

[75] Kumar Vishwajeet and Puneet Singla. Adaptive splitting technique for gaussian mixture models to solve kolmogorov equation. In *American Control Conference (ACC), 2014*, pages 5186–5191, June 2014.

[76] Ba-Ngu Vo, Ba-Tuong Vo, and R. P.S Mahler. Closed-form solutions to Forward-Backward smoothing. *Signal Processing, IEEE Transactions on*, 60(1):2–17, January 2012.

[77] E. A Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *AS-SPCC. The IEEE 2000*. IEEE, 2000.

[78] H. Wei, Y. Cai, H. Li, D. Li, and T. Wang. Sambot: A self-assembly modular robot for swarm robot. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 66–71, May 2010.

[79] A.B. Willumsen and O. Hegrenaes. The joys of smoothing. In *OCEANS 2009 - EUROPE*, pages 1–7, May 2009.

[80] M. Yim, P. J. White, M. Park, and J. Sastra. Modular self-reconfigurable robots. In *Encyclopedia of Complexity and Systems Science*, pages 5618–5631. 2009.

[81] Haitao Zhang, Jian Rong, Xiaochun Zhong, Huajun Yang, Liuping Xiao, and Ling Zhang. The application and design of EKF smoother based on GPS/DR integration for land vehicle navigation. In *Pacific-Asia Workshop on Computational Intelligence and Industrial Application, 2008. PACIIA '08*, volume 1, pages 704–707, 2008.

[82] Jie Zhao, Xindan Cui, Yanhe Zhu, and Shufeng Tang. A new self-reconfigurable modular robotic system ubot: Multi-mode locomotion and self-reconfiguration. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1020–1025, May 2011.