Reducing Bias in Lidar-Only Motion Estimation

by

Yuqing (Tim) Tang

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
University of Toronto Institute for Aerospace Studies
University of Toronto

Abstract

Reducing Bias in Lidar-Only Motion Estimation

Yuqing (Tim) Tang
Master of Applied Science
University of Toronto Institute for Aerospace Studies
University of Toronto
2018

Lidar sensors are used extensively in mobile robotics applications. As a moving-while-scanning sensor, motion estimation from lidar alone is challenging since the point-clouds measured are motion distorted. We develop a continuous-time lidar odometry pipeline based on the simultaneous trajectory estimation and mapping (STEAM) framework. Though accurate, there are biases in the estimator that cause the odometry to drift over time. This thesis presents two approaches for bias reduction. First, a learning-based approach is developed, which trains a model using prior data. The model takes in point-cloud information and outputs the predicted biases. The predicted biases are then applied as a correction to improve the odometry. Next, we show that a source of estimator bias occurs when the motion model cannot sufficiently represent the underlying trajectory. We derive a novel white-noise-on-jerk motion model, and show that it outperforms the white-noise-on-acceleration motion model in the existing STEAM framework.

Acknowledgements

First and foremost, I would like to thank my parents for their selfless support over the past many years, and for instilling me to be curious about the world and always have a thirst for knowledge. It is a privilege to study engineering at the University of Toronto, a privilege that would not have been possible without my parents' support and caring. Thank you.

I want to express my deepest gratitude to my thesis advisor and mentor, Professor Tim Barfoot. As my advisor, Tim has provided countless ideas and directions critical to my thesis research. Tim has been absolutely the best resource on the theory and mathematics in state estimation. Most importantly, Tim has taught me how to approach scientific research properly and rigorously, and how to tackle problems efficiently and effectively - ways of thinking which will benefit me for a lifetime as an engineer and a problem solver. As my mentor, I would like to thank Tim for offering a great amount of support and guidance on making choices for my future academic career. Joining ASRL and becoming Tim's student has been one of the best life decisions I made.

UTIAS is where I became interested in robotics, and I would like to thank the faculty members here who guided me in the past. Thank you Professor Angela Schoellig, for introducing me to mobile robotics and for advising my very first research experience in robotics. Thank you Professor Jonathan Kelly, for introducing me to state estimation and for being my undergraduate thesis advisor.

I would like to acknowledge my colleagues at UTIAS for their generous help on my work and student life. I want to thank David, who worked side-by-side with me on the same project, for helping me with algorithms and ideas, debugging my software, proofreading my papers and thesis, and making our office a fun place to be. Thank you Patrick, for your incredible work on hardware and data collection. Thank you Professor François Pomerleau, for patiently teaching me the fine details on working with lidar point-clouds. Thank you Val, for all the hours spent on teaching me how deep learning works. Thank you Jon, Katarina, Michael, Kirk, Mike, Peter, Mona, Hengwei, and Nan for the awesome discussions on research and other things.

Finally, I want to express my sincere gratitude to Andrew, Keith, and Bruno at Applanix Corporation for the great meetings we had, and for providing valuable guidance with your expertise. I want to also thank the Applanix Corporation and NSERC for sponsoring my project.

Contents

Notations

$\boldsymbol{\mathcal{F}}_{\underrightarrow{\ }a}$	A vetrix representing a reference frame in three dimensions
$(\cdot)^{\curlywedge}$	An operator associated with the adjoint of an element from the Lie algebra for poses
$(\cdot)^{\odot}$	An operator associated with the measurement Jacobian term
$(\cdot)^{\wedge}$	An operator associated with the Lie Algebra for rotation and poses
$(\cdot)_k$	The value of a quantity at timestep k
$(\cdot)_{k_1:k_2}$	The value of a quantity from timestep k_1 to timestep k_2
$\bar{(\cdot)}$	The current best estimate of a quantity
$\boldsymbol{A}$	This font is used time-invariant system quantities
$\check{(\cdot)}$	A prior quantity
$\hat{(\cdot)}$	A posterior (estimated) quantity
$\mathbb{R}^{M\times N}$	The space of real $M \times N$ matrices
$\mathbf{0}$	A zero matrix
$\mathbf{1}$	An identity matrix
$\mathbf{n}$	A normal vector in three dimensions
$\mathbf{p}$	This font is used for quantities that are real vectors of $\mathbb{R}^{M\times 1}$
$\mathbf{S}_k$	A point-cloud with end time at t_k
$\mathbf{T}_{b,a}$	A matrix in $\mathbb{R}^{4\times 4}$ which transforms vectors from $\boldsymbol{\mathcal{F}}_{\underrightarrow{\ }a}$ to $\boldsymbol{\mathcal{F}}_{\underrightarrow{\ }b}$.

$\mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\kappa}(t-t')$	A Gaussian process with mean $\boldsymbol{\mu}(t)$ and covariance $\boldsymbol{\kappa}(t-t')$
$\mathfrak{se}(3)$	The Lie Algebra associated with a member of $SE(3)$
$\|(\cdot)\|$	The norm of a vector
$\mathrm{Ad}(\cdot)$	An operator producing the adjoint of an element from Lie Group
p	This font is used for quantities that are real scalars
$SE(3)$	The special Euclidean group, a matrix Lie group used to represent poses

Chapter 1

Introduction

1.1 Background and Motivation

Mobile robots rely on sensors such as lidar or camera for accurate motion estimation. Lidars have several advantages over cameras, in that they are unaffected by lighting condition, and have a larger field-of-view. In lidar-based odometry, scan-matching associates point-cloud data measured along a trajectory to a common reference frame, and computes an estimate for the trajectory.

The performance of odometry is crucial to tasks in autonomous navigation, such as mapping and localization. For lidar-based mapping, the performance of odometry directly influences the quality of the map, as drift in lidar odometry can cause the generated point-cloud map to be misaligned. For localization against the map, estimates from lidar odometry can act as an initial condition for localization. Furthermore, in a route-following system, the accuracy of lidar odometry dictates how well the robot can follow its path without a successful localization.

Lidar-only odometry is challenging for a number of reasons, among them is the problem of motion distortion. As a moving-while-scanning sensor, a lidar takes point measurements *continuously* as it travels through the environment, resulting in the point-clouds produced being motion distorted. This is analogous to a bad rolling-shutter camera. Moreover, a typical lidar sensor such as the Velodyne HDL-64 can measure more than $100,000$ points in one revolution over the span of just 0.1s, making the problem of lidar-based motion estimation highly expensive computationally.

To address these challenges, we develop a lidar-only odometry algorithm that handles motion distortion by using state-of-the-art techniques for continuous-time trajectory estimation. Our odometry algorithm is carefully designed to have good accuracy while being capable of running in real-time on a CPU. The performance of the odometry algorithm is

demonstrated by our high ranking on the popular KITTI odometry leader board [16]. It was observed, however, that the odometry still exhibits consistent biases when evaluated on various real-world datasets.

In an effort to reduce bias in motion estimation, we explore ideas that seek to learn a bias correction based on training data with ground truth, where the learned bias correction is applied to improve the odometry estimates. Specifically, machine learning models that use Gaussian process (GP) regression and convolutional neural networks (CNNs) are studied.

While we can gain reasonable improvements to odometry accuracy with learning-based methods, it is important to understand fundamentally what might be causing the biases in the estimator. With this in mind, we conduct a theoretical analysis on our continuous-time trajectory estimation framework, and show that a bias can appear when the white-noise-on-acceleration (WNOA) motion prior we use does not sufficiently represent the underlying trajectory. Towards the end of this thesis, we derive a white-noise-on-jerk (WNOJ) motion prior, and demonstrate that it outperforms the WNOA prior on simulated datasets resembling urban driving scenarios.

1.2 Related Work

Lidar-based motion estimation is a well-studied subject in robotics. Most existing lidar estimation methods are variants of the Iterative Closet Point (ICP) algorithm, for which Pomerleau et al. [34] provides a comparative study. Other approaches to point-cloud alignment include the use of Renyi's Quadratic Entropy (RQE), such as the work in [45], [40], and [23].

Extensive past efforts were made in the context of improving lidar-based motion estimation. Segal et al. [37] developed the Generalized ICP (GICP) by combining point-to-point ICP and point-to-plane ICP into a single framework. Serafin et al. [38] developed the Normal Iterative Closest Point (NICP) algorithm by considering both the normal and the local surface information for each point, and showed an overall improvement over GICP. Variants of the Normal Distribution Transforms (NDT) algorithms were developed as alternatives to ICP [24], [42], where the point-cloud is discretized and represented by a combination of normal distributions. Magnusson et al. [25], [26] compared NDT and ICP algorithms, and showed NDT is generally more accurate and may converge faster. The state-of-the-art lidar estimation method, LOAM [49], achieves accurate and efficient scan-matching by having two algorithms (odometry and mapping) running in parallel. Odometry runs at a higher frequency to estimate the velocity of the sensor and unwrap

the motion-distorted point-clouds, while mapping runs at a lower frequency but with higher fidelity to cancel the drift in odometry.

Other than using improved scan-matching algorithms, the performance of motion estimation can also be enhanced by choosing keypoints that are stable and provide sufficient constraints to the problem. LOAM selects keypoints on planes and edges. Sefarin et al. [39] used segmentation to extract keypoints on lines and planes after removing ground points, and showed an improvement in the estimated trajectory than the commonly used NARF keypoints [41].

Our method of learning a bias correction using GP regression seeks to relate the geometry of point-clouds to estimation accuracy. With the same starting point, Gelfand et al. [17] showed that in point-to-plane ICP, points with normals perpendicular to a direction provide no constraints to that direction. Similarly, Zhang et al. [48] showed the lack of geometric structures can lead to degeneracy, making the estimation problem ill-conditioned in certain directions.

Various techniques for reducing bias in motion estimation have been proposed. Vega-Brown et al. [46] improved state esimation by predicting for an adaptive covariance matrix rather than using a fixed one. Farboud-Sheshdeh et al. [14] also applied a bias correction to an estimator, but their method does not learn the correction from training data. Rather, by quantifying how bias increases with measurement noise, the work in [14] is able to compute a corrected estimate for a hypothetical noise-free scenario. Peretroukhin et al. [32] reduced drift in visual odometry (VO) by using Convolutional Neural Networks (CNN) to infer sun direction. Related to our work on learning a bias correction is the method proposed by Hidalgo-Carrió et al. [19], in which a GP model was used to predict for errors in wheel odometry, which is part of a simultaneous localization and mapping (SLAM) system. Results in [19] show that by selecting image frames for VO adaptively based on the predicted errors in wheel odometry, the estimated trajectory did not lose significant accuracy while using much less image frames than selecting image frames non-adaptively.

Finally, this thesis is mostly concerned with continuous-time trajectory estimation. Early work on continuous-time estimation represented the trajectory using temporal basis functions [15]. Tong et al. [44] later developed methods where trajectory estimation is done using GP regression. Barfoot et al. [5] showed the introduction of a continuous-time prior results in the estimation being exactly sparse GP regression, which can be done efficiently. Finally, Anderson and Barfoot [3] extended continuous-time trajectory estimation using sparse GP regression to $SE(3)$, and introduced a white-noise-on-acceleration motion prior. Applications of and further work on continuous-time estimation in $SE(3)$

as GP regression were explored by Boots et al. in [11], [10], and [28].

1.3 Contributions

This thesis has made the following main contributions:

- An accurate and efficient continuous-time lidar-only odometry algorithm capable of handling motion-distorted data. Work in this section contributed towards the publication in [27]
- Learning-based methods for reducing bias in lidar-only estimation. Work in this section resulted in the publication in [43]
- The derivation of a white-noise-on-jerk motion model to serve as a substitution to the existing white-noise-on-acceleration motion model

1.4 Overview

Chapter 2 describes the details of the lidar-only odometry pipeline we developed. Chapter 3 shows evaluations of our odometry algorithm on different real-world datasets, and shows how our odometry solution is biased. We demonstrate the use of learning-based methods for reducing bias in lidar-only motion estimation in Chapter 4. In Chapter 5 we conduct a theoretical analysis on $SE(3)$ estimation, identify a cause for the bias, and derive a new motion prior for continuous-time estimation. Finally, in Chapter 6 we give concluding remarks and discuss future work.

Chapter 2

Lidar-only Motion Estimation

2.1 Continuous-time Trajectory Estimation in SE(3)

2.1.1 Motivation

A scanning lidar is moving *continuously* along the trajectory and we are sampling the points discretely, causing the point-cloud to be motion distorted. Therefore, we cannot treat all points in a full revolution as being measured simultaneously at a discrete timestep, as many discrete-time estimation techniques do. More importantly, we cannot assume they are acquired from a single pose of the lidar.

A naive approach would be to add a pose variable to the state vector for *each* point measured, and solve all pose variables together. This is impractical, since a typical Velodyne sensor can scan more than $100,000$ points in a full revolution, imposing a tremendous computational burden. Instead, we treat the trajectory we wish to estimate as a *continuous-time* trajectory, and solve the trajectory estimation problem using Gaussian process regression. This allows us to solve for sensor poses at discrete timesteps (known as knot times of the continuous-time trajectory), which are typically chosen as the end-times of sensor revolutions, but query the pose at any time along the continuous-time trajectory using standard GP interpolation.

There have been similar techniques where poses at discrete timesteps are kept in the state vector, while poses in between two adjacent discrete timesteps are interpolated, as discussed in Section 1.2. However, these techniques often make ad-hoc assumptions about the motion of the sensor in between two adjacent timesteps in order to carry out the interpolation. In fact, the estimation problem we face is *unobservable*, as there is no other information constraining the motion between discrete timesteps.

Our approach adds a GP prior to the optimization problem. The introduction of a

motion prior allows us to use GP interpolation, which provides a principled approach for querying the state without having to choose any ad-hoc assumption about the motion of the sensor.

2.1.2 Optimization Problem

For continuous-time trajectory estimation, we follow the framework of simultaneous trajectory estimation and mapping (STEAM) as in [3]. Under this set-up, the cost function of the optimization problem consists of measurement terms, and a WNOA motion prior. However, for lidar odometry we do not keep landmarks as part of the state vector, as we are only interested in estimating the trajectory of the sensor.

Let $\mathbf{z}$ to be a collection of all state variables:

$$\mathbf{z} = \{\boldsymbol{\varpi}_0, \mathbf{T}_1, \boldsymbol{\varpi}_1, \ldots, \mathbf{T}_K, \boldsymbol{\varpi}_K\}, \tag{2.1}$$

where we use $\mathbf{T}_k$ to denote $\mathbf{T}_{k,0}$ for simplicity. $\mathbf{T}_k \in SE(3)$ transforms vectors from reference frame $\boldsymbol{\mathcal{F}}_0$ to $\boldsymbol{\mathcal{F}}_k$. Here $\boldsymbol{\mathcal{F}}_k$ is a reference frame associated with time t_k, where each timestep t_k is a knot time of the continuous-time trajectory. $\boldsymbol{\varpi}_k$ is the body-centric velocity associated with timestep t_k. See Section 2.6 for a discussion on choosing knot spacing for the continuous-time trajectory.

Given $\mathbf{z}$, we wish to solve the following optimization problem:

$$\hat{\mathbf{z}} = \arg\min_{\mathbf{z}} J(\mathbf{z}), \tag{2.2}$$

If using L2 cost, the full objective function we wish to minimize is

$$\begin{aligned} J &= J_{\text{prior}} + J_{\text{meas}} \\ &= \sum_i \frac{1}{2}\mathbf{e}_i^T \mathbf{Q}_i^{-1} \mathbf{e}_i + \sum_j \frac{1}{2}\mathbf{g}_j^T \mathbf{R}_j^{-1} \mathbf{g}_j, \end{aligned} \tag{2.3}$$

where $\mathbf{g}_j$ is a measurement error term, $\mathbf{e}_i$ is a prior error term, and $\mathbf{R}_j$ and $\mathbf{Q}_i$ are the corresponding covariance matrices. If a point lies on a planar surface, we can formulate a point-to-plane measurement term by setting the inverse covariance matrix using surface normals:

$$\mathbf{R}_j^{-1} = \beta \mathbf{n}\mathbf{n}^T, \tag{2.4}$$

where β is a scalar coefficient, and $\mathbf{n}$ is the surface normal vector associated with the point. In practice we slightly inflate the diagonal terms of $\mathbf{R}_j^{-1}$ to avoid having a singular

matrix.

We use a WNOA motion prior. For this prior, the inverse covariance matrix can be determined as

$$\mathbf{Q}_i^{-1} = \begin{bmatrix} 12\Delta t_i^{-3}\mathbf{Q}_c^{-1} & -6\Delta t_i^{-2}\mathbf{Q}_c^{-1} \\ -6\Delta t_i^{-2}\mathbf{Q}_c^{-1} & 4\Delta t_i^{-1}\mathbf{Q}_c^{-1} \end{bmatrix}, \tag{2.5}$$

where $\mathbf{Q}_c$ is the *power spectral density matrix*, which is a hyperparameter in our algorithm [6].

Define block quantities of stacked error terms:

$$\mathbf{e} = \begin{bmatrix} \mathbf{e}_1 \\ \vdots \\ \mathbf{e}_M \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_M \end{bmatrix}. \tag{2.6}$$

For standard Gauss-Newton techniques, we need to linearize the error terms with respect to our state at the current operating point, $\mathbf{z}_{\text{op}}$,

$$\mathbf{e} \approx \mathbf{e}_{\text{op}} + \mathbf{E}\boldsymbol{\epsilon}, \qquad \mathbf{g} \approx \mathbf{g}_{\text{op}} + \mathbf{G}\boldsymbol{\epsilon}, \tag{2.7}$$

where $\boldsymbol{\epsilon}$ is a vector of perturbations to our state,

$$\boldsymbol{\epsilon} = \begin{bmatrix} \delta\boldsymbol{\varpi}_0^T & \delta\boldsymbol{\xi}_1^T & \delta\boldsymbol{\varpi}_1^T & \dots & \delta\boldsymbol{\xi}_N^T & \delta\boldsymbol{\varpi}_N^T \end{bmatrix}^T. \tag{2.8}$$

Here $\boldsymbol{\xi} \in \mathbb{R}^6$ is a vectorspace representation of pose, computed by taking the log-map of $SE(3)$ [7]:

$$\delta\boldsymbol{\xi}_i = \ln(\delta\mathbf{T}_i)^\vee, \tag{2.9}$$

where the operator $(\cdot)^\vee$ converts a 4×4 member of the *Lie algebra*, $\mathfrak{se}(3)$, to $\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\rho}^T & \boldsymbol{\phi}^T \end{bmatrix}^T \in \mathbb{R}^6$ [7], [6]. Moreover, $\mathbf{E}$ and $\mathbf{G}$ are the prior and measurement Jacobians:

$$\mathbf{E} = \frac{\partial \mathbf{e}}{\partial \mathbf{z}}\Big|_{\mathbf{z}_{\text{op}}}, \quad \mathbf{G} = \frac{\partial \mathbf{g}}{\partial \mathbf{z}}\Big|_{\mathbf{z}_{\text{op}}}. \tag{2.10}$$

At each iteration of Gauss-Newton, setting $\frac{\partial J}{\partial \boldsymbol{\epsilon}}^T = \mathbf{0}$, we arrive at the following system of linear equations:

$$\begin{aligned} \mathbf{A}\boldsymbol{\epsilon} &= \mathbf{b}, \\ \mathbf{A} &= \mathbf{E}^T\mathbf{Q}^{-1}\mathbf{E} + \mathbf{G}^T\mathbf{R}^{-1}\mathbf{G}, \\ \mathbf{b} &= -\mathbf{E}^T\mathbf{Q}^{-1}\mathbf{e}_{\text{op}} - \mathbf{G}^T\mathbf{R}^{-1}\mathbf{g}_{\text{op}}. \end{aligned} \tag{2.11}$$

We can solve for the optimal perturbation $\boldsymbol{\epsilon}^*$ (e.g., using Cholesky decomposition), and apply the update to the state variables using the following perturbation schemes:

$$\mathbf{T}_{\text{op},i} \leftarrow \exp(\delta\boldsymbol{\xi}_i^{\wedge})\mathbf{T}_{\text{op},i}, \tag{2.12}$$

$$\boldsymbol{\varpi}_{\text{op},i} \leftarrow \boldsymbol{\varpi}_{\text{op},i} + \delta\boldsymbol{\varpi}_i. \tag{2.13}$$

In (2.12) we have made use of the exponential map, where $(\cdot)^{\wedge}$ converts $\boldsymbol{\xi} \in \mathbb{R}^6$ to a member of $\mathfrak{se}(3)$ [7], [6]. It should be noted that while the pose perturbations are vectors, $\delta\boldsymbol{\xi} \in \mathbb{R}^6$, we keep the optimal pose as $SE(3)$ matrices, $\mathbf{T}_{\text{op}} \in SE(3)$. This choice guarantees the pose representation is singularity-free, provided that the perturbations $\boldsymbol{\epsilon}$ are small at each iteration of Gauss-Newton.

2.1.3 Error Terms and Jacobians

To formulate error terms for the motion prior, we follow the approach of *local* pose variables in [3]. This results in a motion prior error term between timesteps t_{k+1} and t_k as

$$\mathbf{e}_k = \begin{bmatrix} \ln(\mathbf{T}_{k+1,k})^{\vee} - (t_{k+1} - t_k)\boldsymbol{\varpi}_k \\ \boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\boldsymbol{\varpi}_{k+1} - \boldsymbol{\varpi}_k \end{bmatrix}. \tag{2.14}$$

The prior error Jacobian evaluated at the current best estimate is

$$\begin{aligned} \mathbf{E}_k &= \begin{bmatrix} \frac{\partial \mathbf{e}_k}{\partial \delta\boldsymbol{\xi}_{k,0}} & \frac{\partial \mathbf{e}_k}{\partial \delta\boldsymbol{\varpi}_k} & \frac{\partial \mathbf{e}_k}{\partial \delta\boldsymbol{\xi}_{k+1,0}} & \frac{\partial \mathbf{e}_k}{\partial \delta\boldsymbol{\varpi}_{k+1}} \end{bmatrix} \\ &= \begin{bmatrix} -\boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\boldsymbol{\mathcal{T}}_{k+1,k} & -(t_{k+1}-t_k)\mathbf{1} & \boldsymbol{\mathcal{J}}_{k+1,k}^{-1} & \mathbf{0} \\ -\frac{1}{2}\boldsymbol{\varpi}_{\text{op},k+1}^{\curlywedge}\boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\boldsymbol{\mathcal{T}}_{k+1,k} & -\mathbf{1} & \frac{1}{2}\boldsymbol{\varpi}_{\text{op},k+1}^{\curlywedge}\boldsymbol{\mathcal{J}}_{k+1,k}^{-1} & \boldsymbol{\mathcal{J}}_{k+1,k}^{-1} \end{bmatrix}, \end{aligned} \tag{2.15}$$

where $\boldsymbol{\mathcal{J}}_{k+1,k}$ is the *left Jacobian* of $SE(3)$ [6] evaluated at the current operating point, $\boldsymbol{\mathcal{J}}_{k+1,k} = \boldsymbol{\mathcal{J}}\left(\ln\left(\mathbf{T}_{\text{op},k+1}\mathbf{T}_{\text{op},k}^{-1}\right)^{\vee}\right)$. $\boldsymbol{\mathcal{T}}_{k+1,k}$ is the *adjoint* of $SE(3)$ ([6]), with $\boldsymbol{\mathcal{T}}_{k+1,k} = \text{Ad}(\mathbf{T}_{\text{op},k+1}\mathbf{T}_{\text{op},k}^{-1})$.

For lidar-based motion estimation, we choose the following measurement error equation:

$$\mathbf{g}_{\text{op},j} = \mathbf{D}(\mathbf{p} - \mathbf{T}_\tau\mathbf{q}), \qquad \mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{2.16}$$

where $\mathbf{p}$ is a homogeneous point measured at time τ, and $\mathbf{q}$ is its matched point measured previously, expressed in the reference frame associated with time t_0. $\mathbf{D} \in \mathbb{R}^{3\times4}$ is a projection matrix.

The measurement error Jacobian at the current best estimate can be evaluated as

$$\mathbf{G}_j = \frac{\partial\, \mathbf{g}_j}{\partial\, \delta\boldsymbol{\xi}_\tau}\bigg|_{\mathbf{z}_{\text{op}}} = -\mathbf{D}(\mathbf{T}_{\text{op},\tau}\mathbf{q})^{\odot}, \tag{2.17}$$

where the $(\cdot)^{\odot}$ operator maps a homogeneous point in $\mathbb{R}^4$ to a matrix in $\mathbb{R}^{4\times 6}$, as defined in [7]:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ 1 \end{bmatrix}^{\odot} = \begin{bmatrix} 1 & 0 & 0 & 0 & u_3 & -u_2 \\ 0 & 1 & 0 & -u_3 & 0 & u_1 \\ 0 & 0 & 1 & u_2 & -u_1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.18}$$

2.1.4 Querying the Trajectory

As discussed in 2.1.1, we can query for a pose and velocity at any time along the trajectory using GP interpolation equations. In fact, the pose in the measurement error equation in (2.16) is $\mathbf{T}_\tau$, but $\mathbf{T}_\tau$ does not need to be a pose we keep as part of our state $\mathbf{z}$. Suppose $\mathbf{z}$ contains $\mathbf{T}_k$, $\boldsymbol{\varpi}_k$, $\mathbf{T}_{k+1}$, and $\boldsymbol{\varpi}_{k+1}$, and that $t_k \leq \tau \leq t_{k+1}$, then we can interpolate for $\mathbf{T}_\tau$ and $\boldsymbol{\varpi}_\tau$ using the interpolation equations as derived in [3] and [4]. Again, this allows us to take into account the fact that the sensor travels *continuously* along the trajectory as it takes point measurements, and that the sensor pose is different at the measurement time of each point.

2.2 Keypoint Extraction

Our lidar odometry algorithm takes in point-clouds as inputs, where each point-cloud is typically assembled by combining all points acquired in a full revolution of the sensor. For a Velodyne HDL-64E lidar, more than 100,000 points can be measured in a revolution, resulting in point-clouds that are very dense. Rather than operating directly on the raw, dense point-clouds, we choose to downsample the point-clouds, or extract keypoints, prior to pose estimation.

There are two main motivations for downsampling the point-clouds. First and most importantly is the computational cost. Processing all points in a dense point-cloud for motion estimation is extremely expensive. As discussed in section 2.3, the cost of point matching is super-linear to the number of points even when we use an efficient implementation such as a k-d tree. Therefore, it is highly challenging to have real-time performance on a CPU while retaining high solution accuracy when all points in a dense point-cloud are used for motion estimation. As such, in order to guarantee

real-time performance while not making significant sacrifices on estimation accuracy, we downsample the point-clouds such that approximately 5% of points are kept and successively used in point matching and motion estimation.

The second motivation for downsampling is that not all points in a point-cloud have the same noise characteristics. Some points have a more noisy range return than other points. Our hope is that, by using prior knowledge of how noise characteristics relate to the geometry of the surface on which each point lies, we can select an informed downsampling strategy to filter out points that we think are more noisy.

We develop two strategies for downsampling. The first strategy looks at the normalized intensity value, and the second strategy looks at a measure of planarity of the surface on which a point lies.

Assuming Lambertian reflectance, the intensity value of a point, I, is inversely proportional to r^2 [21], where r is the range of the point. We define normalized intensity to be Ir^2. The relationship between intensity return and geometry is illustrated in Figure 2.1.

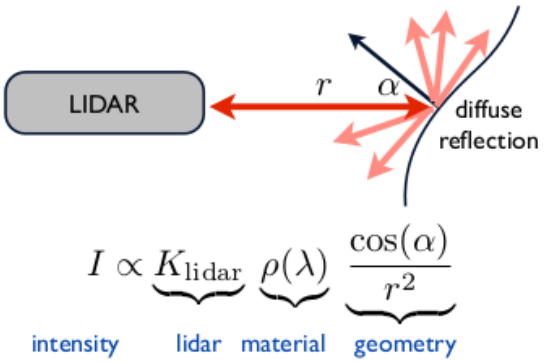

Figure 2.1: Relationship between intensity, lidar constant, surface material, incidence angle, and range.

To determine whether a point lies on a planar surface, similar to [39], we look at the eigenvalues of the covariance matrix of its k-nearest neighbours. This computation is performed using the open-source registration library *libpointmatcher* [34]. The eigenvalues are sorted such that λ_1 is the smallest and λ_3 is the largest. If a point is on a planar surface, then λ_1 will be much smaller than λ_2 and λ_3.

A point may be selected as a keypoint if it satisfies either of the following two conditions:

- Its normalized intensity is greater than a threshold:

$$Ir^2 > \epsilon_I$$

- λ_1 is much smaller than λ_2 and λ_3, which we denote using the following ratio between the eigenvalues:

$$\frac{\lambda_1+\lambda_2+\lambda_3}{\lambda_1} > \epsilon_p$$

As shown in Figure 2.1, Ir^2 varies linearly with $\cos(\alpha)$, where alpha is the incidence angle. Therefore by selecting points with large values of Ir^2, we are automatically rejecting points with large incidence angles, which are known to have noisy range returns [18]. Moreover, since the lidar constant K_{lidar} and material property $\rho(\lambda)$ stays invariant for the same sensor and surface, in terms of geometry, Ir^2 is only a function of the incidence angle α. For each point observed by the sensor, its corresponding α will not change significantly across an interval of a few tenths of a second. Therefore, points with high values of Ir^2 in one revolution of the sensor will also likely to have high values of Ir^2 in the subsequent revolution. This ensures that downsampling by Ir^2 results in a strategy that finds stable keypoints.

We can formulate point-to-plane measurement terms as in (2.4) for points on planar surfaces. In our downsampling strategy, point-to-plane measurement terms are formulated for keypoints satisfying the second condition.

Again, to achieve real-time performances, the thresholds ϵ_I and ϵ_p are chosen such that approximately 5% of points are kept as keypoints and used for odometry. Ground points are ignored when selecting points on planar surfaces; points are treated as ground points based on their z coordinate and surface normal direction. Two conditions are set for selecting keypoints, so that even in environments lacking planar structures, a sufficient number of keypoints can still be selected using the first condition. Figure 2.2 shows a dense point-cloud comparing against its downsampled version. As shown in Figure 2.1, much of the useful structures in the scene such as walls are kept in the downsampled point-clouds, while most of ground points are rejected.

As discussed more in detail in section 3.1, we evaluate our odometry algorithm against the publicly available KITTI dataset [16]. Using the downsampling strategy described in this section, our odometry algorithm reported a translational error of 1.13% on the first 11 sequences of the KITTI dataset. Keeping everything else the same but using random downsampling to 5% of points, our odometry algorithm reported an error of 1.69% on the same sequences. Intuitively, our informed downsampling strategy outperforms random downsampling with the same computational cost.

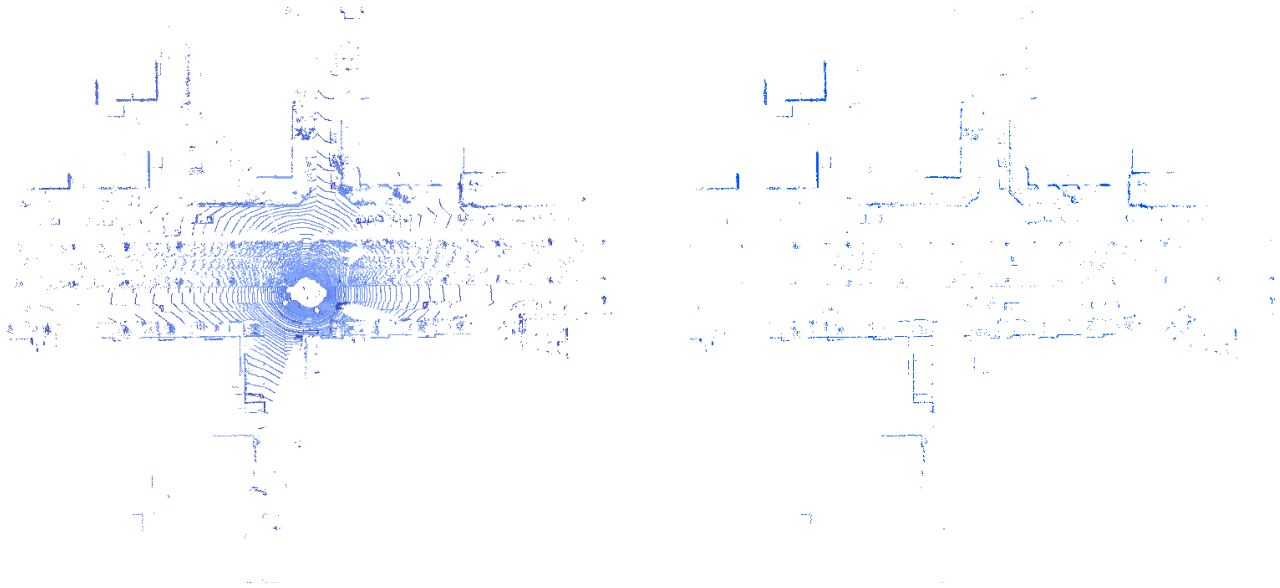

Figure 2.2: Raw, dense point-cloud (left) and downsampled point-cloud (right). The points are downsampled based on their normalized intensity and planarity.

2.3 Finding Point Matches

Similar to standard Iterative Closest Point (ICP) methods, our lidar odometry pipeline is iterative. At the beginning of each iteration, given two downsampled point-clouds, we find point matches based on Euclidean distance to the nearest neighbour. Point matching is handled through the library *libnabo* [2], which implements an efficient nearest-neighbor search using k-d trees. The best case complexity for building a k-d tree is $O(n \log(n))$, where n is the number of points, while the best case for searching a k-d tree is $O(\log(n))$. As such, the cost of point matching is super-linear with the number of points in the point-cloud. Given the matched pair of points, we build the measurement error terms as in (2.16), solve for an optimal perturbation $\boldsymbol{\epsilon}^*$ using Gauss-Newton, and update the trajectory. In the next iteration, given the updated trajectory estimates, we will then recompute the point-matches. The matching and optimization process is repeated until convergence.

Prior to point-matching, the two downsampled point-clouds need to be expressed in the same reference frame given the current best estimate of the trajectory. Standard, discrete-time ICP algorithms typically treat all points in a full revolution as being measured at the exact same timestep, and thus sharing the same sensor pose. These techniques make the ad-hoc assumption that the point-clouds are *rigid*, but this is not true as point-clouds from a moving-while-scanning lidar are motion-distorted. Since we treat the underlying trajectory as *continuous-time*, when transforming a point-cloud to

another reference frame, we query for a pose transform for each point given its timestep, rather than assuming all points share the same pose transform. Therefore, unlike standard, discrete-time ICP algorithms, we do not make the same ad-hoc assumption about point-clouds being *rigid*, but rather handle the inherent motion distortion explicitly.

Figure 2.3 shows a flowchart for our continuous-time odometry. The lidar completes a full revolution between timesteps t_{k-1} and t_k to produce point-cloud $\mathbf{S}_k$, while it completes another full revolution between timesteps t_k and t_{k+1} to produce point-cloud $\mathbf{S}_{k+1}$. To perform trajectory estimation, we first express the two point-clouds in the same reference frame, by querying for a pose transform for each point. Given the two point-clouds expressed in the same reference frame, we find matched pairs of points, build cost terms, update the trajectory, and iterate. Upon convergence we have the posterior estimates $\{\hat{\mathbf{T}}_i, \hat{\boldsymbol{\varpi}}_i\}$. It should be noted that our odometry algorithm has two loops of iterations. There is an outer-loop of iterations for point matching, while within each iteration of point matching, there is an inner-loop of iterations for Gauss-Newton steps to solve for $\boldsymbol{\epsilon}^*$.

2.4 Sliding-window Optimization

For each timestep t_k contained in $\mathbf{z}$, we have an associated pose and velocity, $\mathbf{T}_k$ and $\boldsymbol{\varpi}_k$, except for t_0 where we only have $\boldsymbol{\varpi}_0$. Define each state at t_k to be the pose and a velocity at t_k contained in $\mathbf{z}$, $\{\mathbf{T}_k, \boldsymbol{\varpi}_k\}$. As a middle ground between solution accuracy (batch optimization) and computational speed (single-frame optimization), we use sliding-window optimization [6]. Sliding-window optimization iterates over a window of time-steps and slides this window along to allow for constant-time implementation. The optimization window slides forward by one state when state variables in the current window converge. Please refer to [6] for more details on sliding-window optimization.

We define *locked* states as states that are no longer updated by the mini-batch optimization, and *unlocked* states as ones that are perturbed at each iteration of Gauss-Newton. Furthermore, we can combine point-clouds measured across multiple revolutions into a single point-cloud, and use the combined point-cloud to perform point matching and construct measurement terms.

Figure 2.4 shows a sliding window consisting of two locked states (shown in black) and two unlocked states (shown in grey). The blue dots represent measurement terms, where the black dots represent prior terms between adjacent states. On the top diagram in Figure 2.4, the states associated with t_{k+1} and t_{k+2} are *locked* states in the optimization window. The point-clouds associated with the *locked* states are $\mathbf{S}_{k+1}$ with t_{k+1} as the

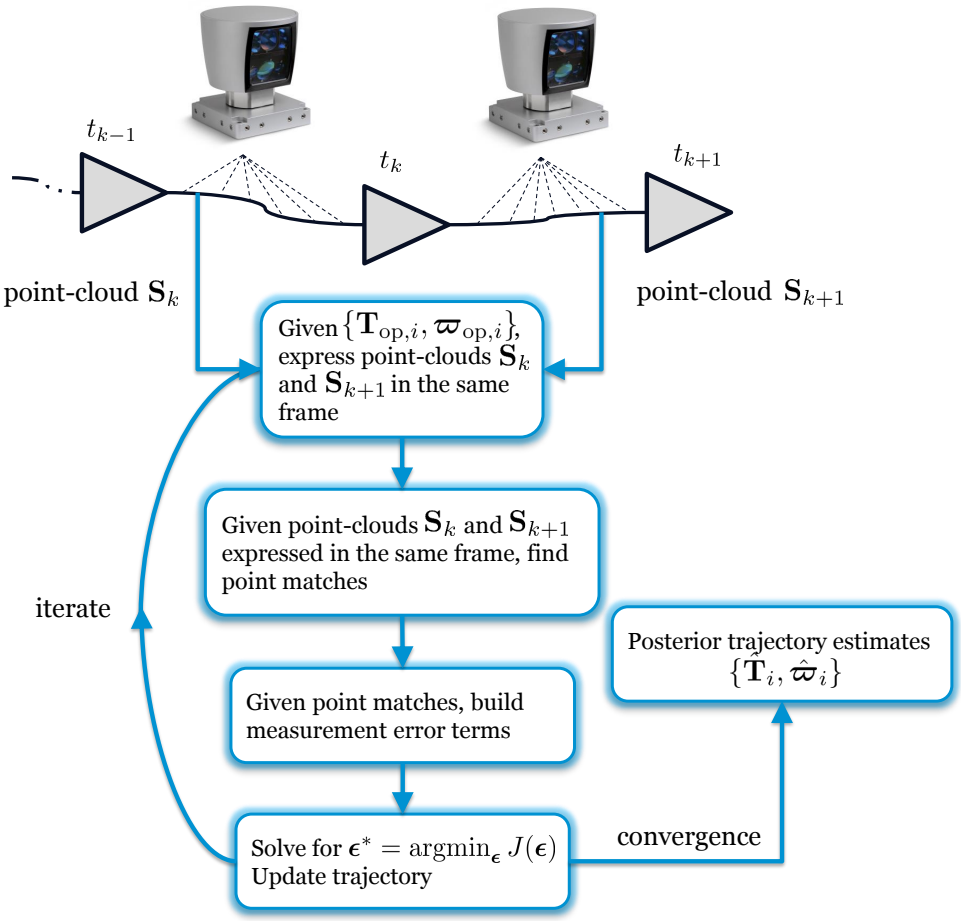

Figure 2.3: Flowchart of the continuous-time odometry pipeline. The triangles represent states, and the blue boxes represent processes in the pipeline. Since we use a continuous-time pipeline, we do not assume all points in $\mathbf{S}_k$ or $\mathbf{S}_{k+1}$ are measured at the same lidar pose.

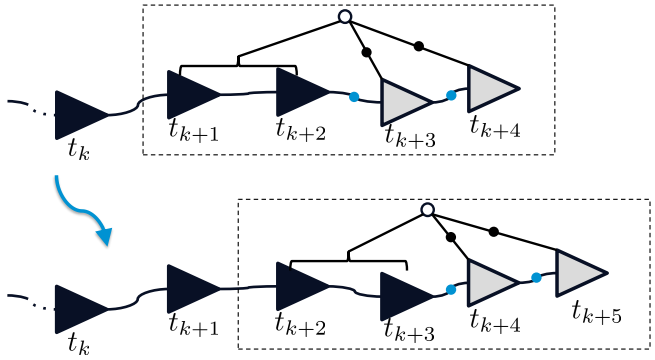

Figure 2.4: Sliding-window optimization, where the window consists of two locked states (black) and two unlocked states (grey). The blue dots are binary motion prior terms between adjacent states. The black dots are measurement terms.

end-time, and $\mathbf{S}_{k+2}$ with t_{k+2} as the end-time. We can combine $\mathbf{S}_{k+1}$ and $\mathbf{S}_{k+2}$ into a single point-cloud, which we denote as $\mathbf{S}'$. There are two point-clouds associated with the *unlocked* states in this window, namely point-cloud $\mathbf{S}_{k+3}$ with end-time t_{k+3} and $\mathbf{S}_{k+4}$ with end-time t_{k+4}. We match $\mathbf{S}_{k+3}$ and $\mathbf{S}_{k+4}$ against $\mathbf{S}'$, and build measurement terms with the point matches.

The mini-batch optimization will compute updates to the *unlocked* states associated with t_{k+3} and t_{k+4}, namely $\{\mathbf{T}_{k+3}, \boldsymbol{\varpi}_{k+3}, \mathbf{T}_{k+4}, \boldsymbol{\varpi}_{k+4}\}$. When we add a new knot to the trajectory at time t_{k+5}, as shown by the bottom diagram in Figure 2.4, the optimization window slides forward by one state. Now $\mathbf{T}_{k+3}$ and $\boldsymbol{\varpi}_{k+3}$ will no longer be updated, but a new *unlocked* state associated with t_{k+5} is added to the optimization window.

Up to a certain point, the solution accuracy for sliding window optimization increases for larger window size, at the expense of an increased computational cost. For our lidar odometry algorithm, we typically choose a window size of three *locked* states and two *unlocked* states, as this is the largest optimization window our algorithm can afford to solve while maintaining real-time performance.

2.5 Robust Cost Functions

Outliers in lidar-only motion estimation can result from different sources, such as incorrectly matched pairs of points, points on dynamic objects, or points with large incidence angles having noisy range returns. Outlier points typically have high residual errors in motion estimation, as shown in Figure 2.5. Outliers are handled through robust cost functions in the odometry algorithm, which downplay their weights in the overall objective function.

The cost functions we analyzed are L2, Cauchy, Geman McClure, and dynamic covariance cost functions. See [22] for a thorough comparison of cost functions in state estimation. Keeping all other parameters the same, we evaluated each cost function on the first 11 sequences of the KITTI dataset. Table 2.1 shows the equation and resulting error on KITTI for the cost functions studied.

In Table 2.1, u is the whitened error norm:

$$u_j = \sqrt{\mathbf{g}_j^T \mathbf{R}_j^{-1} \mathbf{g}_j}. \tag{2.19}$$

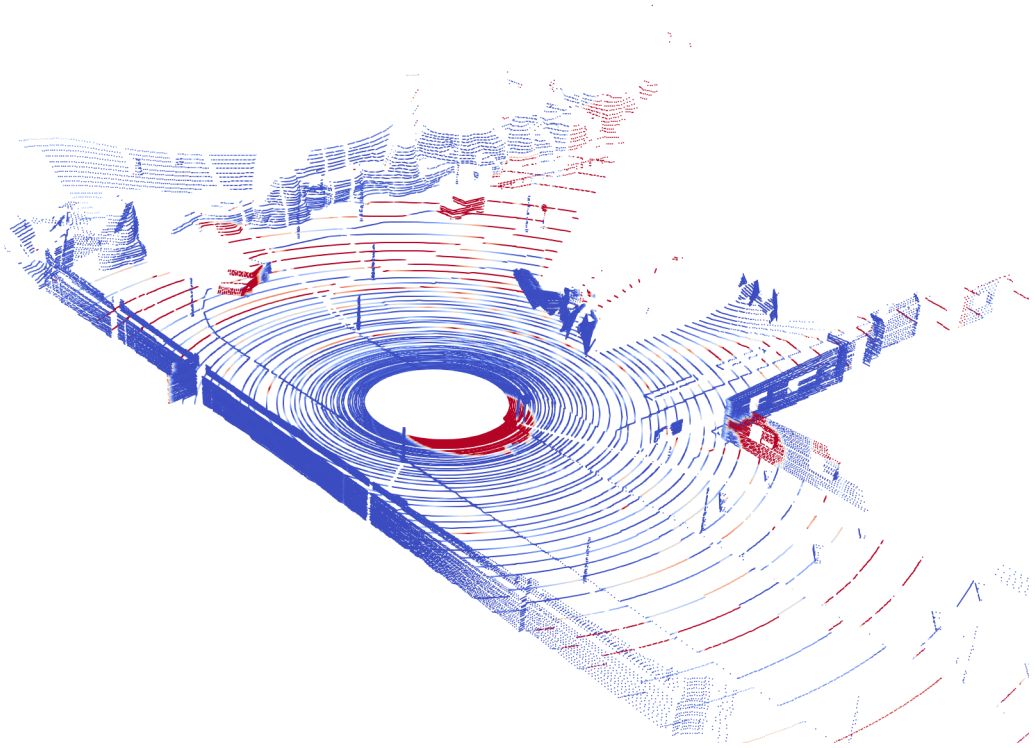

Figure 2.5: Outliers such as points landing on dynamic objects can result in high residual error after the point-clouds are aligned. Points with high residual errors are labelled in red, while points with low residual errors are labelled in blue.

Table 2.1: Error evaluated on the training sequences of KITTI for each cost function

cost function	equation	error(%)
L2	$\frac{1}{2}u^2$	15.6
Cauchy	$\frac{1}{2}\ln(1+u^2)$	2.35
Geman McClure	$\frac{1}{2}\frac{u^2}{1+u^2}$	**1.13**
dynamic covariance	$\frac{1}{2}u^2$ if $u \leq 1$, $\frac{2u^2}{1+2u^2} - \frac{1}{2}$ otherwise	1.20

2.6 Knot Spacing

As described previously in the chapter, knots are state variables stored at discrete timesteps used for representing the underlying continuous-time trajectory. For parametric representations of the trajectory, proper knot spacing is required to model smoothness, and decisions concerning knot spacing have direct impact on the fidelity of the model [4]. The work by Dubé et al. [13] is an example of designing appropriate knot spacing for using spline function to represent the trajectory. An advantage of the GP representation over parametric representation is that less micromanagament regarding knot spacing is needed, as the trajectory smoothness is handled through the continuous-time prior distribution [4]. Therefore, for our odometry algorithm, which represents the trajectory as a Gaussian process, we simply need to choose a knot spacing that allows for real-time performance, but not one that is too large to over-smooth the trajectory.

One particular choice of knot spacing for lidar-based odometry is to add a knot to the trajectory for every sensor revolution. Under this choice, we would add a corresponding pose and velocity pair to the collection of all state variables $\mathbf{z}$, for every revolution of the sensor:

$$\mathbf{z} = \{\boldsymbol{\varpi}_0, \mathbf{T}_1, \boldsymbol{\varpi}_1, \ldots, \mathbf{T}_N, \boldsymbol{\varpi}_N\}, \tag{2.20}$$

where we have state variables at each timestep t_n, with t_n corresponding to the end time of the nth revolution of the sensor, and N is the total number of sensor revolutions in the data sequence. This approach requires the odometry to process data collected by every single revolution of the sensor.

While processing every scan available would be ideal as the odometry is utilizing all measurement information available, it is nevertheless expensive to do so. To reduce computational effort, instead of adding a new knot for every revolution of the sensor, we choose the knot spacing adaptively. For our odometry algorithm, adding a new knot with knot time t_k to the trajectory is essentially adding $\mathbf{T}_k$ and $\boldsymbol{\varpi}_k$ to the collection of state variables $\mathbf{z}$.

We add new knots based on our estimates of the current body-centric velocity of the sensor. Define a translation threshold λ_ρ, a rotation threshold λ_θ, and a time threshold λ_t. Let t_m be the current latest knot time of the continuous-time trajectory, with corresponding state variables $\mathbf{T}_m$ and $\boldsymbol{\varpi}_m$. Further, let $\boldsymbol{\varpi}_m = \begin{bmatrix} \boldsymbol{\varpi}_{m,\rho}^T & \boldsymbol{\varpi}_{m,\theta}^T \end{bmatrix}^T$, where $\boldsymbol{\varpi}_{m,\rho} \in \mathbb{R}^3$ and $\boldsymbol{\varpi}_{m,\theta} \in \mathbb{R}^3$ are the translational and rotational components of $\boldsymbol{\varpi}_m$, respectively. For each new sensor revolution with end time t_k and $t_k > t_m$, we add a new knot at t_k to the end of the trajectory if t_k satisfies any of the following conditions:

- The robot has moved translationally greater than a threshold from the last knot:

$$t_k - t_m > \frac{\|\lambda_\rho\|}{\|\boldsymbol{\varpi}_{m,\rho}\|}$$

- The robot has moved rotationally greater than a threshold from the last knot:

$$t_k - t_m > \frac{\|\lambda_\theta\|}{\|\boldsymbol{\varpi}_{m,\theta}\|}$$

- Some period of time has elapsed since the last knot:

$$t_k - t_m > \lambda_t$$

The first two conditions ensure that the knots are not spaced too far apart in terms of distance or orientation, to avoid over-smoothing the trajectory. The third condition guarantees that we are consistently adding new knots to the trajectory even when our estimates of the current velocity is inaccurate, or if the vehicle has no motion. In our odometry pipeline, λ_ρ is chosen to be 4m, λ_θ is chosen to be 0.1rad, while λ_t is chosen to be 0.3s, such that adjacent knots are no more than 300ms apart.

Again, since we use a continuous-time approach, we can still query the trajectory at any timestep regardless of our choice for knot spacing. We are, however, introducing additional smoothing to the trajectory if we choose a sparser knot spacing, as illustrated in Figure 2.6.

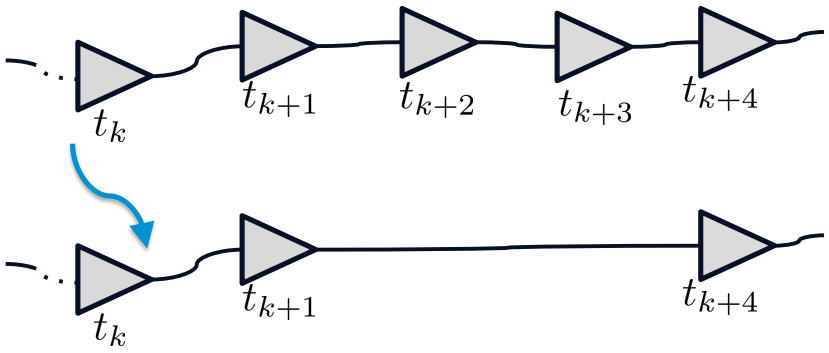

Figure 2.6: Trajectory with tighter knot spacing (top) vs. trajectory with sparser knot spacing (bottom). When the knots at t_{k+2} and t_{k+3} are removed from the trajectory, we are essentially introducing additional smoothing between t_{k+1} and t_{k+4}.

2.7 Parameters of the Odometry Algorithm

See Table 2.2 for the list of parameters in our lidar odometry algorithm. The power spectral density matrix, $\mathbf{Q}_c$, is tuned on the training sequences of the KITTI odometry benchmark, as explained in Section 3.1.

Table 2.2: Parameters in the lidar odometry algorithm

Parameter	Description	Value
$\mathbf{Q}_c$	Power spectral density matrix for WNOA prior	$\begin{bmatrix} 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.001 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 \end{bmatrix}$
ϵ_I	Threshold for downsampling based on normalized intensity	97th percentile of Ir^2 of all points in a dense point-cloud
ϵ_p	Threshold for downsampling based on planarity	97th percentile of $\frac{\lambda_1+\lambda_2+\lambda_3}{\lambda_1}$ of all points in a dense point-cloud
λ_ρ	Translational threshold for adding knots	4m
λ_θ	Rotational threshold for adding knots	0.1rad
λ_t	Time threshold for adding knots	0.3s

2.8 Software and System Implementation

The lidar odometry algorithm is implemented in C++. OpenMP was used to parallelize the construction of cost terms. As mentioned previously, we use *libpointmatcher* [34] for surface normal computation and *libnabo* [2] for point matching. Our lidar odometry algorithm is capable of running in real-time on a Lenovo ThinkPad P90 laptop, without having to use GPUs.

We have implemented a lighter version of the odometry algorithm on the Robot Operating System (ROS), which is used to demonstrate live lidar-only motion estimation on our mapping vehicle equipped with a Velodyne sensor, shown in Figure 2.7. The lighter version of the odometry software implements frame-to-frame optimization on the live data stream, and uses a knot spacing the same as the spinning frequency of the lidar (0.1s for a Velodyne HDL-64E). To ensure online performance, we always terminate the processing the current point-cloud within 100 milliseconds, so the system is ready for processing the next point-cloud. During operation, the algorithm runs on a laptop mounted inside the mapping vehicle, which is connected to the Velodyne sensor via an Ethernet connection.

Figure 2.7: Our Buick Encore mapping vehicle, equipped with a Velodyne HDL-64E lidar and Applanix POS system.

2.9 Summary

In this chapter we described the details of the lidar odometry algorithm we developed. Specifically, our algorithm performs continuous-time trajectory estimation using lidar as the only sensor, and the estimation is done in an iterative fashion, where matched pairs of points are found at the beginning of each iteration to build measurement cost terms for the objective function. On top of measurement cost terms, the objective function also contains motion prior cost terms, as in the set-up of simultaneous trajectory estimation and mapping (STEAM).

To reduce the computational cost, we downsample the point-cloud prior to motion estimation, rather than operating directly on the raw, dense point-clouds. We use an informed downsampling approach based on normalized intensity and surface planarity. The lidar odometry algorithm operates using sliding-window optimization, and handles outliers with robust cost functions. Finally, we use an adaptive scheme to choose the knot spacing in order to reduce computational cost, but avoid over-smoothing the trajectory.

Chapter 3

Evaluation on Lidar Datasets

3.1 KITTI Dataset

Our lidar odometry pipeline is evaluated on the publicly available KITTI odometry benchmark [16]. The KITTI odometry dataset contains 39.2km of Velodyne data divided in 22 sequences, covering scenes including city, residential, highway, and campus. The lidar data as given in the KITTI odometry dataset have been post-processed by the dataset authors to compensate motion distortion. Sequences 0-10 are the training sequences, where the ground truth trajectories are available to users, while sequences 11-21 are the test sequences where the ground truths are not publicly available. The benchmark reports a translational error as a percentage of distance over path segments of $100, 200, \ldots, 800$ meters. Submissions are ranked on the KITTI leader board based on the average translational error on the test sequences.

The hyperparameter $\mathbf{Q}_c$ for our algorithm is tuned to achieve the smallest error over the 11 training sequences, shown in Table 2.2. The results on the training sequences are shown in Table 3.1. We have a total error of 1.26% on the 11 test sequences, and ranked 3rd on the leader board among lidar-only methods at the time of submission.

In general, our algorithm has decent accuracy, with error on many sequences less than 1%. Examples of the estimated trajectory compared against ground truth trajectory are shown in Figures 3.1 and 3.2. As shown, the estimated trajectory aligns closely to the ground truth trajectory from a top-down perspective for these sequences.

However, in some sequences the estimated trajectory demonstrates a pronounced bias, particularly in the directions of translational z (ρ_3), roll (ϕ_1), and pitch (ϕ_2). This causes the estimated position of the vehicle to drift away from the ground truth, usually in the negative z direction. Examples of sequences where the biases are clearly observable are shown in Figures 3.3 and 3.4. Such sequences usually have higher translational errors.

Table 3.1: Odometry errors before and after the correction for sequences 0 to 10 of the KITTI dataset

Sequence no.	Number of frames	Translational error(%)
0	4541	1.5465
1	1101	2.2232
2	4661	0.9862
3	801	0.7296
4	271	0.6206
5	2761	0.6064
6	1101	0.5013
7	1101	0.6795
8	4071	1.0585
9	1591	0.9478
10	1201	1.7572
Total		1.1288

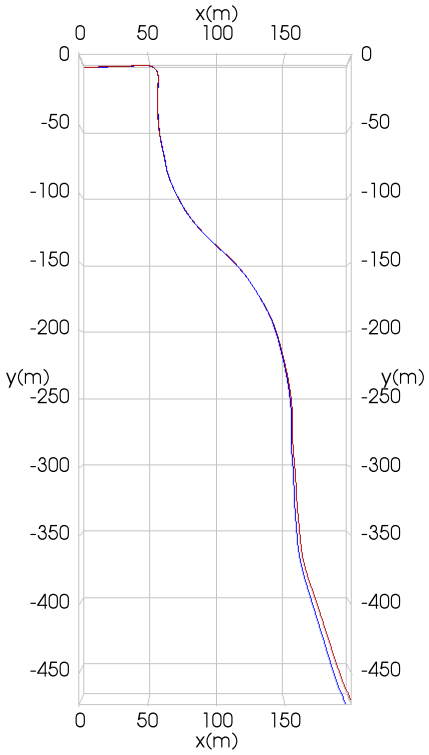

Figure 3.1: Top-down view of the estimated trajectory (blue) vs. ground truth trajectory (red) for sequence 3 of KITTI.

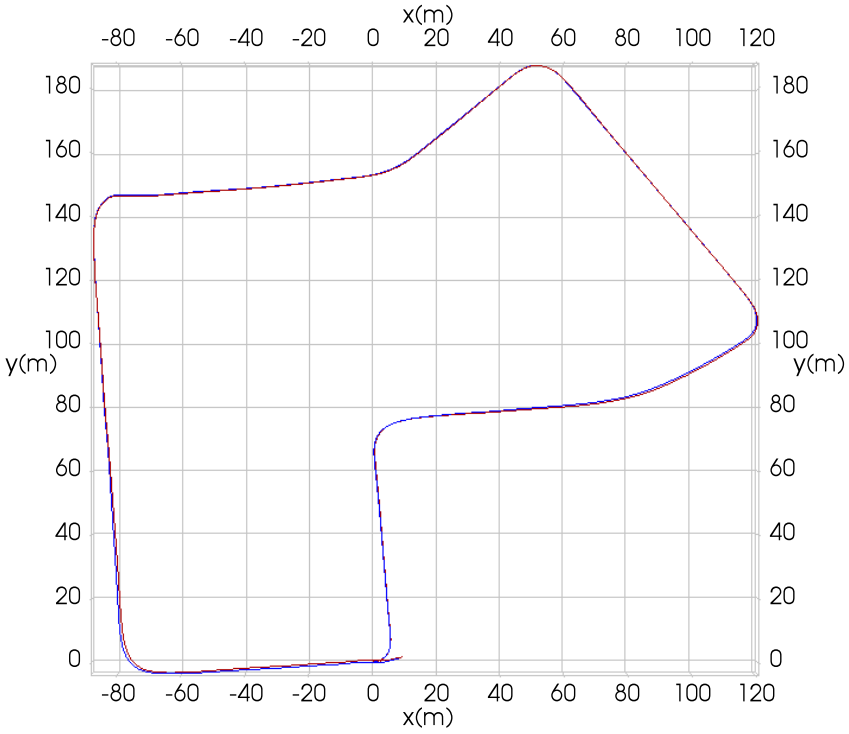

Figure 3.2: Top-down view of the estimated trajectory (blue) vs. ground truth trajectory (red) for sequence 7 of KITTI.

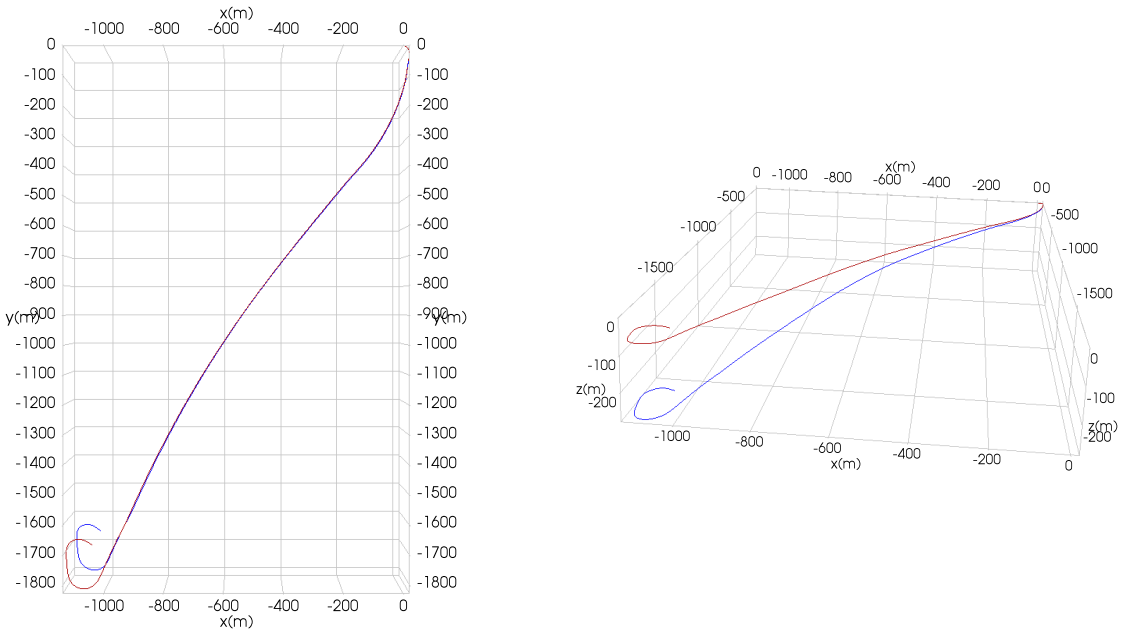

Figure 3.3: Top-down view (left) and 3D view (right) of the estimated trajectory (blue) vs. ground truth trajectory (red) for sequence 1 of KITTI.

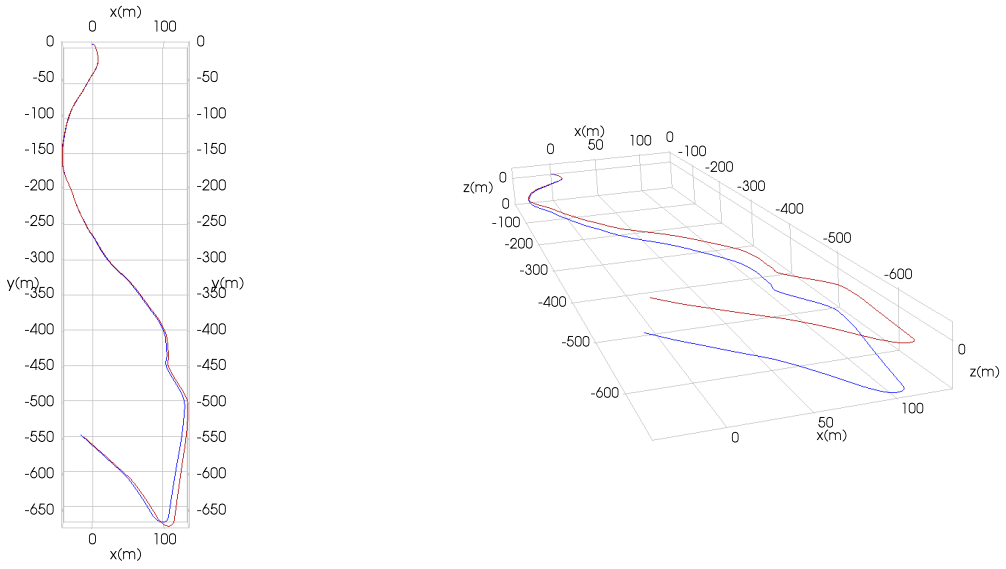

Figure 3.4: Top-down view (left) and 3D view (right) of the estimated trajectory (blue) vs. ground truth trajectory (red) for sequence 10 of KITTI.

3.2 University of Toronto Campus Dataset

A dataset was collected using our test vehicle (Figure 2.7) along different routes around the University of Toronto (U of T). This resulted in 7 sequences of Velodyne data over 18km of traversal. Similar to the KITTI dataset, 6 DOF ground truth is available for the University of Toronto campus dataset, where the ground truth is obtained using an Applanix POS system. However, unlike the KITTI dataset, we do not process the point-clouds to compensate motion distortion, but rather rely on our continuous-time estimation framework, as described in Chapter 2, to account for the effect of motion distortion. For consistency, we evaluate the odometry error in the same fashion as KITTI, where a translational error is reported as a percentage of distance travelled over path segments of length $100, 200, \ldots, 800$ meters.

The odometry errors are shown in Table 3.2. In general the errors are slightly greater than the results for the KITTI dataset, mostly due to the fact that the point-clouds have not been post-processed to compensate for motion distortion.

Examples of estimated trajectory compared against ground truth trajectory are shown in Figures 3.5 and 3.6. From the 3D view plots in Figures 3.5 and 3.6, the biases in ρ_3, ϕ_1, and ϕ_2 are also clearly visible in the estimated trajectories for this dataset.

Table 3.2: Odometry errors before and after the correction for the University of Toronto campus dataset

Sequence no.	Number of frames	Translational error(%)
0	5000	1.6748
1	5000	1.9166
2	6000	1.6252
3	6000	2.2462
4	3000	2.1262
5	6600	1.5629
6	7650	1.7962
Total		1.8146

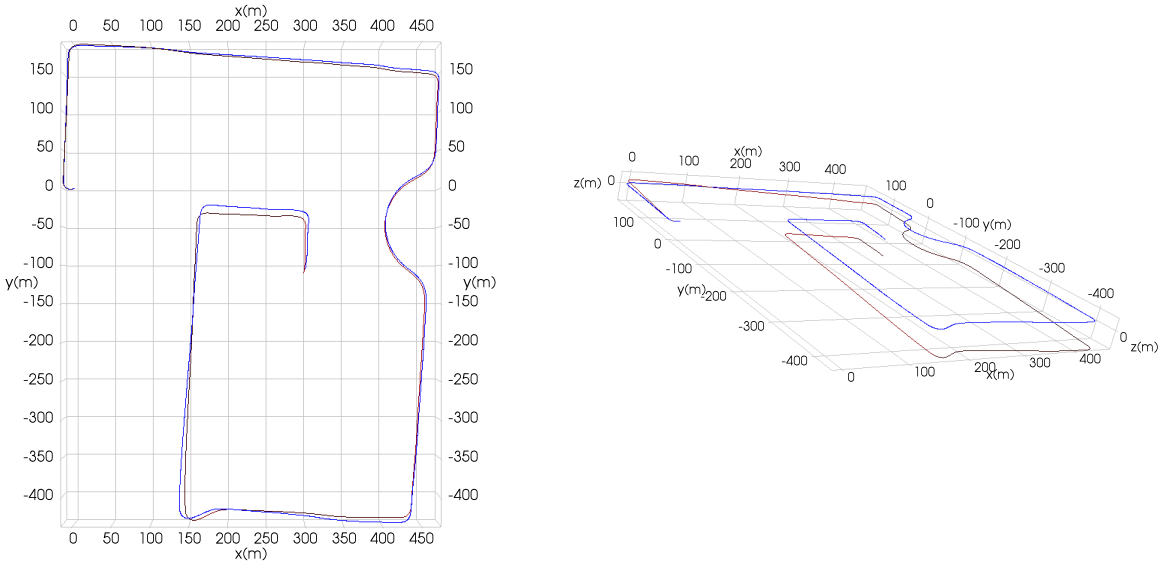

Figure 3.5: Top-down view (left) and 3D view (right) of the estimated trajectory (blue) vs. ground truth trajectory (red) for sequence 1 of the University of Toronto campus dataset.

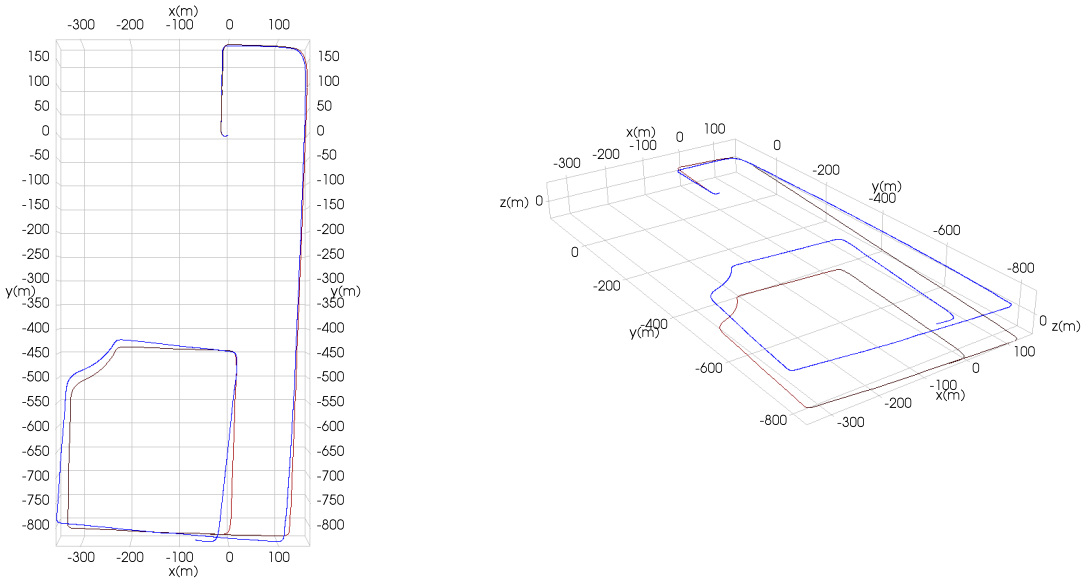

Figure 3.6: Top-down view (left) and 3D view (right) of the estimated trajectory (blue) vs. ground truth trajectory (red) for sequence 6 of the University of Toronto campus dataset.

Table 3.3: Odometry errors before and after the correction for the University of Toronto campus dataset

Sequence no.	Distance(m)	Translational error(%)
0	17097	1.5887
1	7493.8	2.3229
2	35012	2.3449
Total		2.1180

3.3 Applanix Dataset

A dataset was collected in areas near Applanix Corporation in Richmond Hill, Ontario, North of Toronto, using the same mapping vehicle as in Figure 2.7. A total of 60.4km of data was collected, divided into three sequences. While the University of Toronto campus dataset features the same sensor suite as the Applanix dataset, the campus dataset only features well-structured, urban environments while driving at less than 50km/h. The Applanix dataset, however, includes driving in suburbs and rural areas, which lack useful geometry and structure. Moreover, a large portion of the Applanix dataset was collected while driving at more than 70km/h, with parts on the highway as fast as over 100km/h. As such, the Applanix dataset constitutes a much more challenging dataset than the University of Toronto campus dataset.

The odometry errors are shown in Table 3.3. Again, the errors are higher than the U of T campus dataset, because the dataset was collected in more challenging environments while driving at a faster speed.

Example plots of the estimated trajectory compared against ground truth are shown in Figures 3.7 and 3.8. It can be seen that the estimated trajectory exhibits biases in roll and pitch more significantly than in the KITTI dataset or the U of T campus dataset.

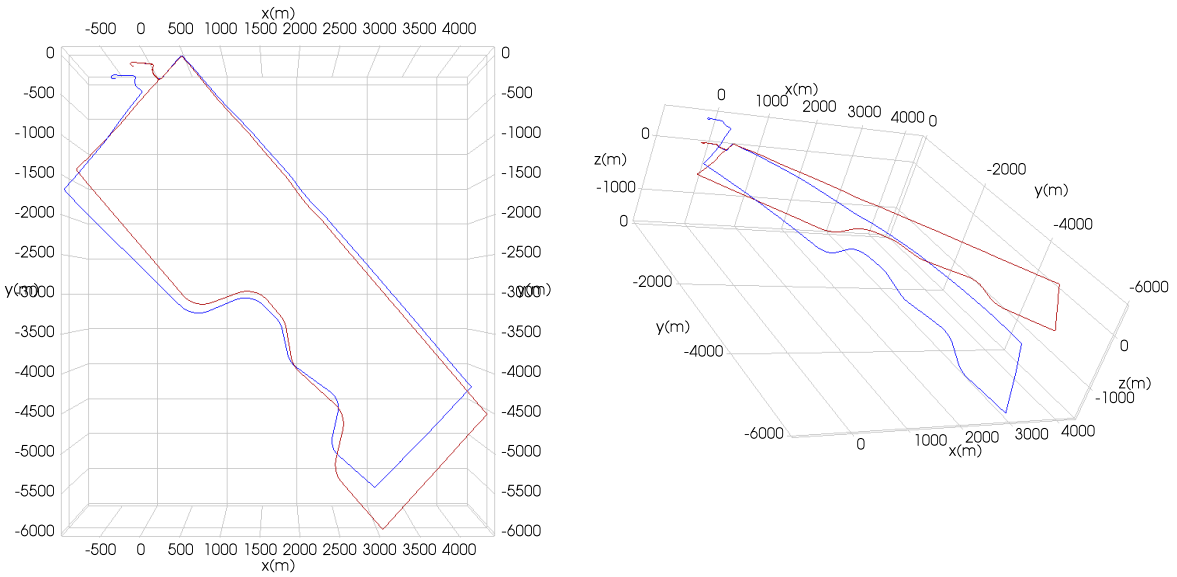

Figure 3.7: Top-down view (left) and 3D view (right) of the estimated trajectory (blue) vs. ground truth trajectory (red) for sequence 0 of the Applanix dataset.

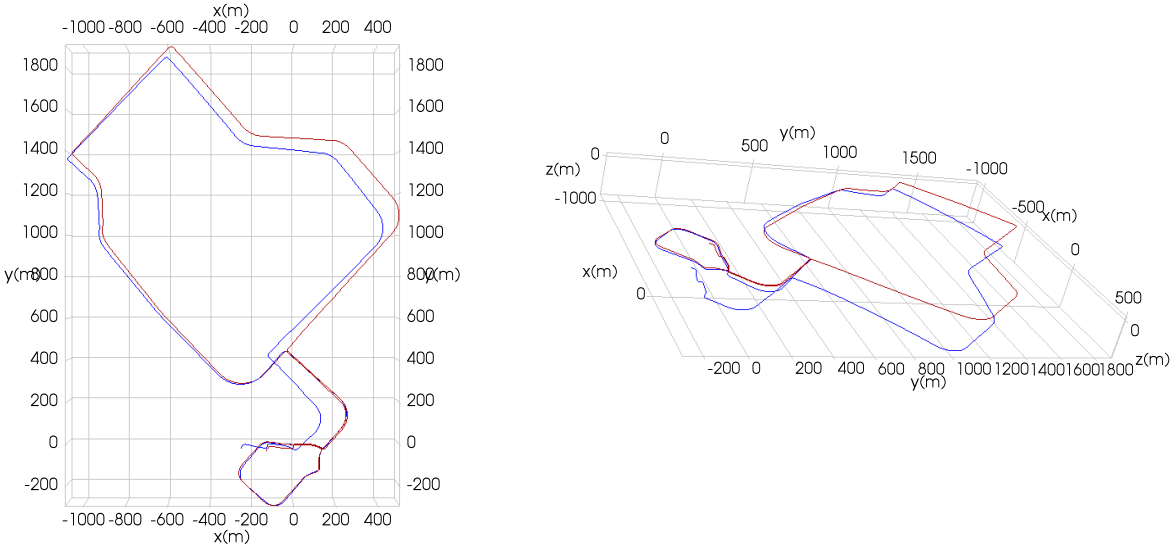

Figure 3.8: Top-down view (left) and 3D view (right) of the estimated trajectory (blue) vs. ground truth trajectory (red) for sequence 1 of the Applanix dataset.

3.4 Discussion

Our odometry algorithm is reasonably accurate as demonstrated by evaluation on multiple datasets, including motion-distorted datasets, and datasets with post-processed measurements to remove motion distortion. The accuracy of our odometry algorithm is lower for datasets where the vehicle undergoes high speed, aggressive motion.

Our odometry algorithm exhibits significant bias, particularly in the directions of z, pitch, and roll. Such bias can be seen in all types of data, whether motion-distorted or not. The main goal for this thesis is thus to explore methods to reduce the bias in our estimator and improve the solution accuracy. The next two chapters discuss ideas for reducing bias in estimation.

Chapter 4

Learning-based Methods for Bias Correction

4.1 Motivation

In Chapter 3 we have shown that our lidar odometry algorithm consistently produces a bias mostly in the negative z direction, when evaluated on real-world data. In Chapter 2 we showed various methods for improving the accuracy of state estimation which we already implemented, such as finding stable and low-noise keypoints, choosing a larger sliding-window size, and using robust cost functions. While these design decisions all resulted in an improvement in solution accuracy, they do not eliminate the issue of bias in the estimator.

Given that the bias in the odometry is somewhat systematic, our hope is that using training data with ground truth, we can predict for the bias on test data, and apply the predicted bias as a correction to improve the odometry. This chapter therefore explores methods that seek to learn a bias correction for lidar-only motion estimation.

Two approaches to learning-based methods are explored. In Section 4.2 we use a Gaussian process model for the learning, while in Section 4.3 we look at using deep learning with convolutional neural networks (CNNs). Specifically, for the GP approach we choose inputs to the model based on the geometry of the point-cloud, as we believe estimator biases are related to the geometry in the point measurements. Also, it should be noted that the GP model for learning the bias is different and separate from the GP regression in the lidar odometry algorithm.

4.2 Gaussian Process Regression

4.2.1 Odometry Error Evaluation

We define a frame as a full sweep (360°) of the lidar, and we query our continuous-time trajectory for each frame. For a data sequence, this results in N queried poses, $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_N$, where N is the number of frames.

To evaluate odometry errors we define a window of κ frames. For each frame τ, where $\tau \geq \kappa$, we calculate the relative pose change from frame $\tau - \kappa$, and compare that against ground truth to compute an error:

$$\mathbf{T}_{\text{odom},\tau,\tau-\kappa} = \mathbf{T}_{\text{odom},\tau}\mathbf{T}^{-1}_{\text{odom},\tau-\kappa}, \tag{4.1}$$

$$\mathbf{T}_{\text{gt},\tau,\tau-\kappa} = \mathbf{T}_{\text{gt},\tau}\mathbf{T}^{-1}_{\text{gt},\tau-\kappa}, \tag{4.2}$$

$$\mathbf{T}_{\text{err},\tau,\tau-\kappa} = \mathbf{T}_{\text{gt},\tau,\tau-\kappa}\mathbf{T}^{-1}_{\text{odom},\tau,\tau-\kappa}, \tag{4.3}$$

where $\mathbf{T}_{\text{odom}}$ is pose estimated by lidar odometry, $\mathbf{T}_{\text{gt}}$ is the ground truth pose, and $\mathbf{T}_{\text{err}}$ is the odometry error. We evaluate the odometry error for $\tau = \kappa, \kappa+1, \ldots, N$.

Rather than always evaluating pose change from the previous frame, $\mathbf{T}_{\tau,\tau-1}$, we evaluate $\mathbf{T}_{\tau,\tau-\kappa}$ and leave κ as a parameter of choice. This is due to the high frame rate of the sensor. The Velodyne lidar spins at 10 Hz, therefore $\mathbf{T}_{\tau,\tau-1}$ is the pose change of the sensor over 0.1 seconds, corresponding to a very short section of the trajectory. The imprecision in GPS measurements might make ground truth over such a short trajectory section noisy.

For the error to be a valid output of a GP model, we convert $\mathbf{T}_{\text{err}} \in SE(3)$ to a vectorspace representation:

$$\boldsymbol{\xi}_{\text{err},\tau,\tau-\kappa} = \ln(\mathbf{T}_{\text{err},\tau,\tau-\kappa})^{\vee}, \tag{4.4}$$

where $\boldsymbol{\xi}_{\text{err},\tau,\tau-\kappa} \in \mathbb{R}^6$ is the vectorspace representation of the error. The operator $(\cdot)^{\vee}$ converts a 4×4 member of the *Lie algebra*, $\mathfrak{se}(3)$, to $\boldsymbol{\xi} = \begin{bmatrix}\boldsymbol{\rho}^T & \boldsymbol{\phi}^T\end{bmatrix}^T \in \mathbb{R}^6$ [7], [6], where $\boldsymbol{\rho} = \begin{bmatrix}\rho_1 & \rho_2 & \rho_3\end{bmatrix}^T$, and $\boldsymbol{\phi} = \begin{bmatrix}\phi_1 & \phi_2 & \phi_3\end{bmatrix}^T$. We can convert $\boldsymbol{\xi}_{\text{err},\tau,\tau-\kappa}$ back to $SE(3)$ via the exponential map:

$$\mathbf{T}_{\text{err},\tau,\tau-\kappa} = \exp(\boldsymbol{\xi}_{\text{err},\tau,\tau-\kappa}{}^{\wedge}), \tag{4.5}$$

where $(\cdot)^{\wedge}$ converts $\boldsymbol{\xi} \in \mathbb{R}^6$ to a member of $\mathfrak{se}(3)$ [7], [6].

4.2.2 Applying Learned Correction

Given odometry estimates from new data, for each frame $k \geq m$, we predict for the odometry error between frames k and $k-m$, which we denote by $\boldsymbol{\xi}^*_{\text{err},k,k-m}$. We then apply the predicted errors as corrections to the estimates, by converting the predicted error from vectorspace back to $SE(3)$ using (4.5). We assume the error is accumulated uniformly from frame $k-m$ to k, as shown in Algorithm 1. The prediction and correction step is applied to the poses for frames $k = m, m+1, \ldots, N$, where N is the total number of frames.

Algorithm 1 Applying correction to odometry

Let $\mathbf{T}_{\text{corr},m-1} = \mathbf{T}_{\text{odom},m-1}$
for $k = m, m+1, \ldots, N$ **do**
    predict for odometry error $\boldsymbol{\xi}^*_{\text{err},k,k-m}$
    $\delta\boldsymbol{\xi}^*_{\text{err}} = \frac{1}{m}\boldsymbol{\xi}^*_{\text{err},k,k-m}$
    $\delta\mathbf{T}^*_{\text{err}} = \exp(\delta\boldsymbol{\xi}^{*\wedge}_{\text{err}})$
    $\mathbf{T}_{\text{odom},k,k-1} = \mathbf{T}_{\text{odom},k}\mathbf{T}^{-1}_{\text{odom},k-1}$
    $\mathbf{T}_{\text{corr},k,k-1} = \delta\mathbf{T}^*_{\text{err}}\mathbf{T}_{\text{odom},k,k-1}$
    $\mathbf{T}_{\text{corr,k}} = \mathbf{T}_{\text{corr},k,k-1}\mathbf{T}_{\text{corr},k-1}$
end for

Shown in Algorithm 1, $\mathbf{T}_{\text{odom},k}$ is the pose for frame k before applying the correction, and $\mathbf{T}_{\text{corr},k}$ is the pose for frame k after correction is applied. In practice, m is set between 2 to 20. The majority of odometry errors for our odometry algorithm are in the directions of $z\,(\rho_3)$, pitch (ϕ_2), and roll (ϕ_1), while our algorithm is relatively accurate in the directions of $x\,(\rho_1)$, $y\,(\rho_2)$, and yaw (ϕ_3) (see Figure 4.1 for the coordinate system our odometry algorithm uses). Therefore, we only make predictions and corrections in 3-DOF for z, pitch, and roll.

Since we only make predictions in z, pitch, and roll, we have

$$\boldsymbol{\xi}^*_{\text{err}} = \begin{bmatrix} 0 & 0 & e^*_{\rho 3} & e^*_{\phi 1} & e^*_{\phi 2} & 0 \end{bmatrix}^T. \tag{4.6}$$

4.2.3 GP Model

GP models offer a principled approach for learning from noisy observations, and are much easier to design and cheaper than some of the other machine learning techniques such as CNNs. Gaussian processes have been widely used in robotics, such as in terrain assessment [8], building occupancy grid maps [30], and trajectory estimation [3]. Again, it should be noted that the GP model for bias correction we present here is separate and

Figure 4.1: The coordinate system used by our odometry algorithm. Roll, pitch, and yaw are rotations about the x, y, and z axes, respectively.

different from the GP regression we use for trajectory estimation, which is presented in Chapter 2 of this document.

We wish to predict the odometry error, $\boldsymbol{\xi}_{\text{err}}$, which is a vector in $\mathbb{R}^6$. While there are methods to handle GP regression with vector outputs, for simplicity we model each degree of freedom (DOF) of $\boldsymbol{\xi}_{\text{err}}$ separately. In other words, we fit a separate model for each element of the 6-DOF error vector.

For n observations in a dataset, we have n D-dimensional inputs $\mathbf{X} = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_n\end{bmatrix}^T$, where $\mathbf{x}_i \in \mathbb{R}^D$, and $\mathbf{X} \in \mathbb{R}^{n \times D}$. We also have n scalar outputs $\mathbf{y} = \begin{bmatrix}y_1 & y_2 & \dots & y_n\end{bmatrix}^T$, where $\mathbf{y} \in \mathbb{R}^n$. Let f be the underlying relation between input and output, but the observations are noisy. Therefore we have $y_i = f(\mathbf{x}_i) + \epsilon$, where ϵ is the noise, which we assume to be Gaussian with variance σ_n^2. In our problem, the inputs $\mathbf{x}_i$ are features derived from geometry of the point-clouds. The scalar output y_i is an element of the error vector $\boldsymbol{\xi}_{err}$, as shown in Section 4.2.4.

Given $\mathbf{X}$ and $\mathbf{y}$, to make predictions $\mathbf{f}^*$ on new data $\mathbf{X}^*$, the predictive distribution for Gaussian process regression [36] can be formulated as

$$p(\mathbf{f}^*|\mathbf{X}, \mathbf{y}, \mathbf{X}^*) \sim \mathcal{N}(\bar{\mathbf{f}}^*, \text{cov}(\mathbf{f}^*)), \tag{4.7}$$

$$\bar{\mathbf{f}}^* = \mathbf{K}(\mathbf{X}^*, \mathbf{X})[(\mathbf{K}(\mathbf{X}, \mathbf{X}) + (\sigma_n)^2\mathbf{I})]^{-1}\mathbf{y}, \tag{4.8}$$

$$\text{cov}(\mathbf{f}^*) = \mathbf{K}(\mathbf{X}^*, \mathbf{X}^*) - \mathbf{K}(\mathbf{X}^*, \mathbf{X})[(\mathbf{K}(\mathbf{X}, \mathbf{X}) + (\sigma_n)^2\mathbf{I})]^{-1}\mathbf{K}(\mathbf{X}, \mathbf{X}^*), \tag{4.9}$$

where $\bar{\mathbf{f}}^*$ is the mean prediction and $\text{cov}(\mathbf{f}^*)$ is the variance. $\mathbf{K}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{n \times n}$ is the kernel matrix with $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, where $k(\cdot, \cdot)$ is the kernel function. We use the squared

exponential kernel function with a separate length scale for each input dimension [36]:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j) \right), \tag{4.10}$$

where $\mathbf{M} \in \mathbb{R}^{D \times D}$ is a diagonal matrix with entries $l_1^{-2}, \ldots, l_D^{-2}$, and $l_1, \ldots, l_D$ are the characteristic length scales for each of the D input dimensions. σ_f is the signal standard deviation, and σ_n is the noise standard deviation.

$\Theta = \{l_1, \ldots, l_D, \sigma_f, \sigma_n\}$ form the set of hyperparameters for the GP model. Define the log marginal likelihood [36]:

$$\ln\ p(\mathbf{y}|\mathbf{X}, \Theta) = -\frac{1}{2}\mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y} - \frac{1}{2}\ln|\mathbf{K}_y| - \frac{n}{2}\ln 2\pi, \tag{4.11}$$

where $\mathbf{K}_y = \mathbf{K}(\mathbf{X}, \mathbf{X}) + (\sigma_n)^2 \mathbf{I}$. In practice the hyperparamters Θ are chosen by maximizing the log marginal likelihood in (4.11) with respect to Θ.

4.2.4 Input Features

In Gaussian process regression, model selection is the process of making choices about the details of the model. For our problem, model selection involves choosing the input to the model, as well as setting the hyperparamters (4.11).

Choosing correct inputs for the GP model is crucial to its predictive capabilities. However, this is a non-trivial task if the output of the model is error in odometry. The method in [19] used orientation angles, speed and position of joints, measurements of gyroscopes, and IMU measurements as inputs to model errors in wheel odometry. Selecting inputs is less obvious in our situation, however, as we do not have measurements from any sensors other than the lidar. In our method, the inputs are selected based on high-level features derived from the geometry of points in a point-cloud. We show that we can achieve significant reduction in odometry error, using only inputs derived from the point-clouds and no other measurements.

We fit a GP model to each DOF of odometry error. This requires choosing a set of input features for each element of the error vector $\boldsymbol{\xi}_{\text{err}}$ we wish to model.

z and pitch

We use the same input to predict for errors in z and pitch. For z, the output of the GP model is

$$e_{\rho_3} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{\xi}_{\text{err}}, \tag{4.12}$$

and for pitch:

$$e_{\phi_2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{\xi}_{\text{err}}. \tag{4.13}$$

Our choices for candidate input features are inspired by the work in [17], where the authors argue that in an ICP problem, the distribution of normal vectors affects how well-constrained the solution is in each degree of freedom. A number of input features based on the distribution of surface normals were tested, and we selected input features based on evaluation on the training sequences of the KITTI benchmark, as described in Section 4.2.5. Here we present our final choice for input features, which resulted in the greatest reduction in odometry errors after applying correction on the KITTI training sequences, among all input features tested.

For the 3×1 normal vector $\mathbf{n} = \begin{bmatrix} n_x & n_y & n_z \end{bmatrix}^T$, we sum each component of the normal over all points lying on planar surfaces to form our 3-dimensional input feature, and normalize by the number of points. Points not on planar surfaces are ignored, since their normal estimates are noisy. If the downsampled point-cloud associated with frame i has M points, in which P of them are on planar surfaces, then the input can be calculated as

$$\mathbf{x}_i = \frac{1}{M} \begin{bmatrix} \sum_{p=1}^{P} \|n_{p,x}\| & \sum_{p=1}^{P} \|n_{p,y}\| & \sum_{p=1}^{P} \|n_{p,z}\| \end{bmatrix}^T, \tag{4.14}$$

where $\mathbf{n} = \begin{bmatrix} n_{p,x} & n_{p,y} & n_{p,z} \end{bmatrix}^T$ is the normal for point p.

roll

We discretize the downsampled point-cloud into 16 equally spaced "slices" based on azimuth, as shown in Figure 4.2. A drawing of a car is included for reference. For each "slice", we calculate the number of points on planar surfaces with normal vector pointing in the z-direction, and normalize by the total number of points. This forms our 16-dimensional input. The output of the GP model for roll is

$$e_{\phi_1} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \boldsymbol{\xi}_{\text{err}}. \tag{4.15}$$

4.2.5 Results on the Lidar Datasets

Evaluating on KITTI Training Sequences

We use the first 11 sequences of the KITTI dataset for selecting the best input features since ground truth is available. To evaluate how well a specific set of inputs predict the

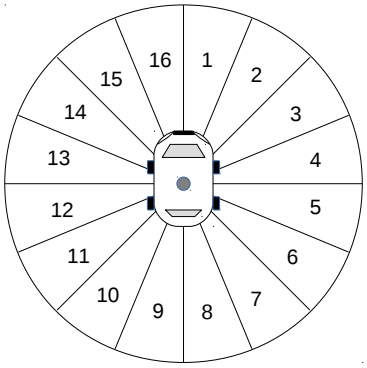

Figure 4.2: We divide the point-cloud into 16 "slices" based on azimuth. For each "slice", we calculate the number of points with normals in the z-direction and normalize. This forms our 16-dimensional input $\mathbf{x}_i$ for roll.

odometry error, we use cross-validation on the training data. Specifically, we leave 1 sequence out as the validation sequence, and fit the model on the other 10 sequences. The fitted model is used to make predictions on the unseen validation sequence, and the predictions are used to correct for the odometry estimates of the validation sequence. We cross-validate by repeating this process for every sequence, such that we have the corrected odometry estimates for all 11 sequences. The KITTI benchmark evaluates percentage errors across path segments of lengths $100, 200, \ldots, 800$ meters, and an average over all path segments is computed. A total error averaged over path segments evaluated for all 11 sequences is also reported. The total error before and after the correction are used to quantify the improvements from applying the learned bias correction.

The training is done offline using the Gaussian Process Regression tool of the MATLAB Statistics and Machine Learning Toolbox [1]. To set the hyperparameters, the log marginal likelihood in (4.11) is maximized with respect to the hyperparameters using gradient-based optimization.

We have experimented with a number of input features for predicting odometry errors in z, pitch, and roll. The set of input features resulting in the greatest reduction in total odometry errors were selected, namely the input features shown in Section 4.2.4. Using these input features, the odometry error before and after the correction on the first 11 sequences of KITTI are shown in Table 4.1. Examples showing improvements to the trajectory are shown in Figures 4.3 and 4.4. The corrections improved the odometry for 8 of the 11 sequences. Over all 11 sequences, the error is reduced from 1.13% to 1.03%, accounting for a 9% reduction. The prediction and correction steps cost only 10s of computational time, which is approximately 0.5% of the total computational time.

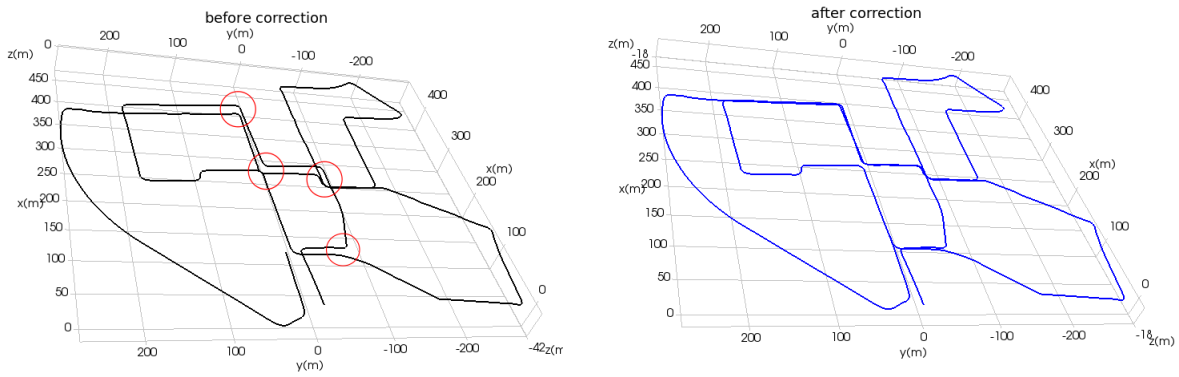

Figure 4.3: 3D plots of odometry estimates for sequence 10: uncorrected odometry estimates (black) vs. corrected odometry estimates (blue) when compared against ground truth (red). The corrections brought noticeable improvements over z and pitch.

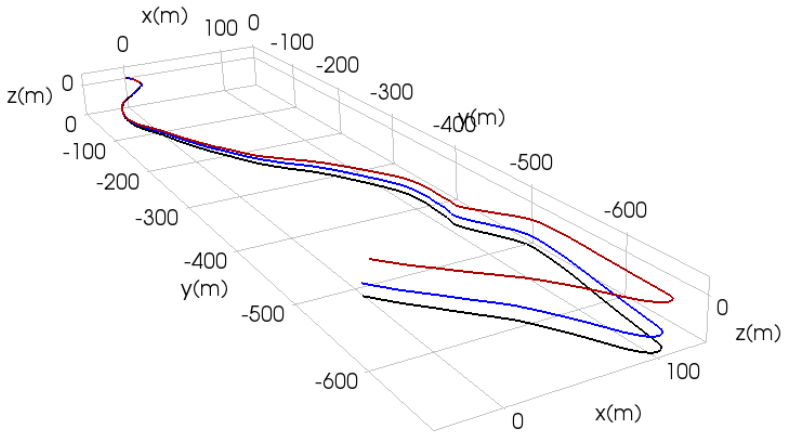

Figure 4.4: 3D plots of odometry estimates for sequence 10: uncorrected odometry estimates (black) vs. corrected odometry estimates (blue) when compared against ground truth (red). The corrections brought noticeable improvements over z and pitch.

Table 4.1: Odometry errors before and after GP correction for sequences 0 to 10 of the KITTI dataset.

Sequence no.	Number of frames	Uncorrected odometry error (%)	Corrected odometry error(%)
0	4541	1.5465	**1.427**
1	1101	2.2232	**1.9627**
2	4661	0.9862	**0.8613**
3	801	**0.7296**	0.8381
4	271	0.6206	**0.4802**
5	2761	0.6064	**0.5316**
6	1101	0.5013	**0.4106**
7	1101	**0.6795**	0.7579
8	4071	1.0585	**1.0007**
9	1591	**0.9478**	1.0023
10	1201	1.7572	**1.3501**
Total		1.1288	**1.0294**

Evaluating on KITTI Test Sequences

Sequences 11-21 are the test sequences of KITTI, and ground truth is unavailable for these sequences. For evaluating against the 11 test sequences, we fit a model using all training sequences 0-10 with input features described in Section 4.2.4. The predicted error is applied as a correction to each of sequences 11-21. Figure 4.5 shows the odometry error over all test sequences before and after the correction. Our method showed significant reduction in odometry error for path segments of all lengths from 100m to 800m. The improvement is more pronounced the longer the path segment. For path segments of 800m, the error is reduced from 1.51% to 1.30%, equivalent to a 14% reduction.

The total error over all path segments for all test sequences was reduced from 1.26% down to 1.16%, accounting for an 8% reduction. Our odometry algorithm is accurate even before applying any corrections, ranking 3rd on the KITTI leader board at the time of submission among methods that use lidar only. Our uncorrected result currently ranks 4th among lidar-only methods as our corrected result now ranks 3rd. In fact, the top two methods for lidar only algorithm either performs SLAM (IMLS-SLAM), or has a mapping algorithm running in parallel to cancel the drift in odometry (LOAM). In contrast, our (uncorrected) method is strictly only odometry, and does not use loop closures or a second estimation algorithm to reduce the drift. By predicting errors using GP regression and applying them back as a bias correction, we achieved significant performance improvements over an already accurate odometry algorithm.

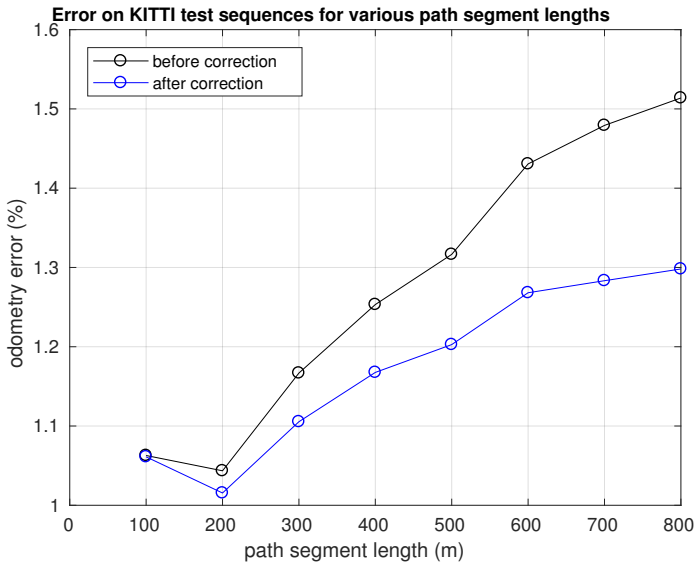

Figure 4.5: Error before and after correction on the KITTI test sequences. Our method produced improvements for path segments of all lengths between 100 to 800 m. Specifically, for path segments of 800m, the odometry error decreased from 1.51% to 1.30%, resulting in a 14% improvement.

Evaluating on University of Toronto Campus Dataset

Here we evaluate our GP learning approach on the University of Toronto Campus Dataset, introduced in Section 3.2. First, we attempted to fit a model using the training sequences of KITTI, and predict for odometry corrections on the University of Toronto campus dataset. However, this did not improve the odometry due to the considerable differences between the two datasets, including different calibration parameters for the Velodyne sensor, different systems for obtaining ground truth, and whether the lidar data is post-processed. Rather, we use cross-validation among sequences of the University of Toronto campus dataset to demonstrate the effectiveness of our method on this dataset. For each sequence, we leave it out as validation sequence and fit a model on the other 6 sequences, and the fitted model is used to make predictions on the unseen validation sequence. We do, however, use the same input features as selected by evaluating on the KITTI training sequences (Section 4.2.4), as opposed to choosing another set of input features specifically for the University of Toronto campus dataset.

The odometry errors before and after the correction are shown in Table 4.2. The corrections improved 6 out of the 7 sequences, while the total error is reduced from 1.81% to 1.56%, accounting for a 14% reduction in odometry errors. Figure 4.6 shows plots of the odometry estimates for sequence 2 before and after the correction is applied. Before the correction, the estimated path does not overlap with itself when the vehicle travelled back to a street it has been before. This bias is eliminated after the correction is applied.

Table 4.2: Odometry errors before and after the correction for the University of Toronto campus dataset

Sequence no.	Number of frames	Uncorrected odometry error (%)	Corrected odometry error(%)
0	5000	1.6748	**1.2696**
1	5000	1.9166	**1.3305**
2	6000	1.6252	**1.2102**
3	6000	2.2462	**1.9529**
4	3000	2.1262	**1.9334**
5	6600	**1.5629**	1.7260
6	7650	1.7962	**1.6395**
Total		1.8146	**1.5598**

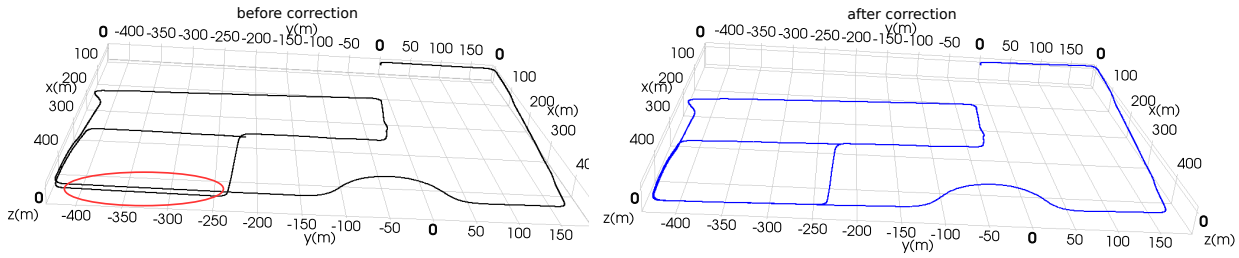

Figure 4.6: 3D plots of odometry estimates for sequence 2 of the University of Toronto campus datasets from the same perspective. Black is odometry before correction and blue is odometry after correction. Left: due to errors, odometry does not overlap when the vehicle travels back to a path it has been before (circled). Right: after applying the correction, the odometry estimate overlaps well when the vehicle travels back to the same path.

4.3 Convolutional Neural Networks

4.3.1 Overview

So far, when using GP regression for learning a bias, the input features to the GP model have been hand-picked. A natural extension to this work is therefore to apply convolutional neural networks (CNNs) for learning the bias correction, where the features are learned at each layer of the network, rather than hand-picked.

Recently, a number of works have been proposed on using deep learning for end-to-end visual odometry [47], visual-inertial odometry [9], and lidar odometry [29]. While deep learning does offer an alternative approach to the motion estimation problem, the results of these methods are typically much less accurate when compared against state-of-the-art classical estimators. As such, we do not seek to replace well-studied classical estimators with learning-based motion estimators.

The problem of quantifying bias from sensor measurements, however, is a task suitable for deep learning methods. There could be many unknown sources of bias in the estimator related to sensor measurements, and it is challenging to analytically quantify all of them, or to hand pick features to regress all sources of bias. We seek to utilize the best of both worlds, where we use classical state estimators for motion estimation, but we apply deep learning on top of the classical estimator to learn its biases.

Related to our approach is the work in [33], which learns a bias correction for visual odometry. In [33], the stereo images used for visual odometry are also used as inputs to a CNN, which outputs the predicted biases from the input images. For lidar odometry, the inputs to the estimator are 3D point-clouds. While deep learning methods using 3D points as inputs have been proposed [35], for simplicity we create 2D range images from the 3D point-clouds, and apply standard 2D image convolutions.

Given a 3D point-cloud with range, azimuth, and elevation, we discretize the point-cloud into a 360×64 grid to form a 2D range image. The horizontal axis of the grid is the azimuth angle from $-180°$ to $180°$, the vertical axis is the elevation spaced based on the lidar's vertical field of view. The pixel value in each grid (pixel) is the average range of all points in that grid. If there are no points in a grid, the pixel value is set to be zero. The horizontal grid spacing (image resolution) is chosen to be 360 since the point-cloud is measured with a full $360°$ revolution, while the vertical grid spacing (resolution) is chosen to be 64 since the Velodyne HDL-64E has 64 lasers with different elevation.

An example of generating a 2D range image from a 3D point-cloud is shown in Figure 4.7. Figure 4.7 shows the result conversion using dense point-clouds for better visualization. However, since our odometry algorithm runs on downsampled point-clouds, in

practice we create range images on the downsampled point-clouds, rather than the dense point-clouds.

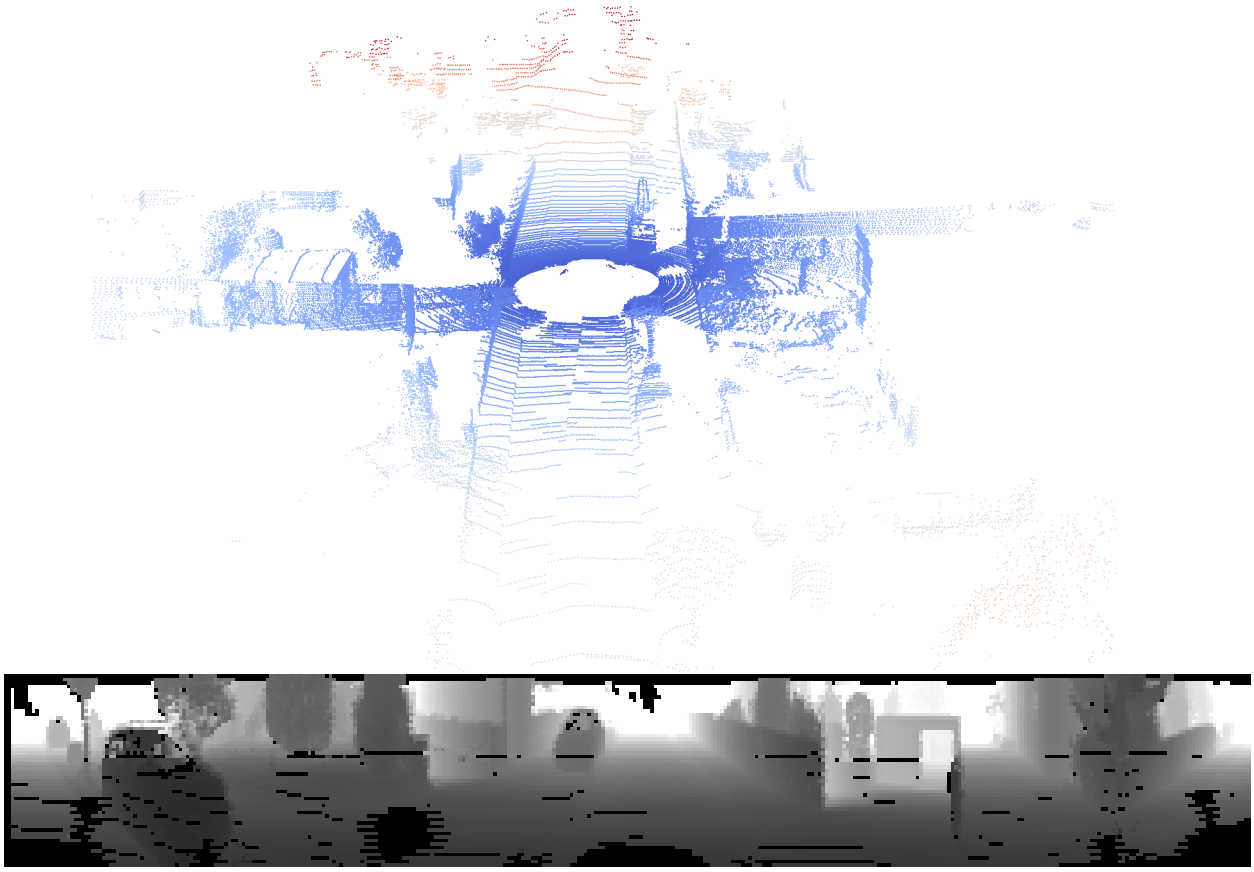

Figure 4.7: A 3D point-cloud with points colored by range (top), and the corresponding 2D range image (bottom)

4.3.2 Training and Testing

We follow the same notations as our GP approach, where we denote the predicted error as $\boldsymbol{\xi}^*_{\text{err}}$ or $\mathbf{T}^*_{\text{err}}$, and the ground truth error as $\boldsymbol{\xi}_{\text{err}}$ or $\mathbf{T}_{\text{err}}$. In our GP approach, we learned the error for one degree of freedom at a time, where the output of the model is an element of $\boldsymbol{\xi}^*_{\text{err}}$. Since CNNs are capable of producing vector outputs, we aim to predict the entire vector-space error $\boldsymbol{\xi}^*_{\text{err}} \in \mathbb{R}^6$.

For CNN, we need to define a loss function that the network seeks to minimize. We

define the loss using the same approach as in [33]:

$$\mathcal{L} = \frac{1}{2}\mathbf{g}(\boldsymbol{\xi}^*_{\text{err}})^T \boldsymbol{\Sigma}^{-1} \mathbf{g}(\boldsymbol{\xi}^*_{\text{err}}), \tag{4.16}$$

$$\mathbf{g}(\boldsymbol{\xi}^*_{\text{err}}) = \ln\Big(\exp\Big(\boldsymbol{\xi}^{*\wedge}_{\text{err}}\Big)\mathbf{T}^{-1}_{\text{err}}\Big)^{\vee}, \tag{4.17}$$

where $\boldsymbol{\Sigma}$ is an empirical covariance computed over the training set,

$$\boldsymbol{\Sigma} = \frac{1}{N-1}\sum_{i=1}^{N}(\boldsymbol{\xi}_{\text{err}} - \bar{\boldsymbol{\xi}}_{\text{err}})(\boldsymbol{\xi}_{\text{err}} - \bar{\boldsymbol{\xi}}_{\text{err}})^T, \quad \bar{\boldsymbol{\xi}}_{\text{err}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\xi}_{\text{err}}. \tag{4.18}$$

The errors in the translational degrees of freedom in $\boldsymbol{\xi}_{\text{err}}$ are usually an order of magnitude larger than the rotational degrees of freedom. Computing an empirical covariance balances the translational and rotational components in the loss term.

We evaluate our method on the KITTI training set. One sequence from the 11 sequences is the test sequence that we wish to correct the odometry for, while another sequence is selected as validation sequence, used for terminating the training when the network exhibits over-fitting. We train the model using the other 9 sequences over 30 epochs. At the end of each epoch, the trained model makes prediction on the validation sequence and computes a validation loss using Equation (4.16). The training terminates either upon reaching 30 epochs, or when the validation loss increases after an epoch, which signals the beginning of over-fitting. Finally, the trained model makes prediction on the test sequence, where the predicted errors are applied as corrections to the odometry of the test sequence.

To apply the predicted errors as pose corrections, we follow the methodology as in [33], where the pose corrections are constructed as cost terms, and applied to the estimated trajectory through a pose graph relaxation optimization. This has the advantage over our correction method for using GP models in Section 4.2.2, in that we no longer need to explicitly assume the error is accumulated uniformly within a short time interval.

In our experiments, we arbitrarily selected sequence 5 as the validation sequence. When sequence 5 is used as the test sequence, we arbitrarily selected sequence 0 as the validation sequence. The final results on the corrected odometry for each sequence of the KITTI training set are shown in Section 4.3.4.

4.3.3 Model Architecture

A number of network architectures were explored and the overall performance on the KITTI training set is evaluated for each architecture. Our experiments show that the DenseNet architecture [20] greatly outperforms the other architectures tested for our application. Specifically, we used the DenseNet-121 architecture as described in detail in [20]. In terms of software implementation, the model was trained on PyTorch [31], which offers built-in implementations for the DenseNet architectures.

An advantage of CNN over GP regression is that the training time does not necessarily increase super-linearly with input dimensions. As a result of this limitation in the GP case, each input was derived from a single point-cloud, as there was no obvious way to combine inputs from multiple point-clouds without increasing the input dimension. In the CNN approach, we can stack multiple range images together along a dimension and combine them into a single tensor input. This allows the model to learn the error accumulated over a window of frames, while taking into account of any temporal correlation between adjacent frames.

We have chosen a window size of 8 frames for our approach. It should be noted that the window size here is merely a parameter for learning the error, and needs not to be the same as the window size used by our sliding-window odometry algorithm. Assuming the estimated trajectory from odometry has knots spaced at timesteps $\{t_0, t_1, t_2, \ldots, t_N\}$, where for each knot time t_k there is a corresponding point-cloud $\mathbf{S}_k$ with end time at t_k. To predict for the error accumulated from t_k to t_{k+7}, we have a total of 8 point-clouds with end times at $t_k, t_{k+1}, \ldots, t_{k+7}$. After the range images are created for each point-cloud, they are stacked across the third dimension to produce a tensor of dimension $360 \times 64 \times 8$, as illustrated by Figure 4.8. This forms the input to the model.

Figure 4.8: A $360 \times 64 \times 8$ tensor range image used as input to the CNN model.

Table 4.3: Odometry errors before and after CNN correction for sequences 0 to 10 of the KITTI dataset.

Sequence no.	Number of frames	Uncorrected odometry error (%)	Corrected odometry error(%)
0	4541	1.5465	**1.4653**
1	1101	2.2232	**1.5235**
2	4661	0.9862	**0.8222**
3	801	0.7296	**0.4520**
4	271	0.6206	**0.3530**
5	2761	0.6064	**0.5434**
6	1101	0.5013	**0.3257**
7	1101	0.6795	**0.6015**
8	4071	1.0585	**1.0560**
9	1591	**0.9478**	0.9591
10	1201	1.7572	**1.1609**
Total		1.1288	**1.0050**

4.3.4 Results

The results of our CNN approach is evaluated on the KITTI training set, shown in Table 4.3. In general, the results are slightly better than the GP approach. We did not validate the CNN approach on other datasets due to time constraints.

4.4 Summary

In this chapter we explored methods to predict for odometry error given information on the point-clouds measured by the sensor. The key underlying assumption behind our methods is that the bias in motion estimation is correlated with the geometry of point-clouds. Specifically, the learning was done using a GP regression approach and a deep learning approach.

When using GP regression, our model takes in low-dimensional features derived from the geometry of point-clouds as inputs, and outputs a specific degree of freedom of the error. For the deep learning approach, we use a CNN which takes in a tensor of stacked range images as input, and outputs the 6-DOF error vector. Both methods successfully improved the odometry on the KITTI training set. The GP approach was also tested on the KITTI test set, and a dataset collected around the University of Toronto campus which has motion-distorted point-clouds.

The biggest limitation to our approach of learning a bias correction is that the training data has to sufficiently represent the test data. We found, for example, that a model trained using the KITTI training set did not improve the odometry for the University of

Toronto campus dataset. In the next chapter we will investigate more generic methods for reducing bias which do not require prior training data.

Chapter 5

Theoretical Analysis of Motion Estimation in SE(3)

5.1 Overview

So far we have demonstrated methods for learning a bias correction for our estimator, using either Gaussian process regression or convolutional neural networks. While these methods can bring considerable improvements to the estimation accuracy when there are sufficient training data, they reveal little on the root cause of biases in an estimator. Although the tuned hyperparameters of a GP model give information on the correlation between a specific input feature and the bias, we still lack a fundamental understanding of which aspect of the state estimator is the root cause of bias. Another limitation of our learning approach is that the training data needs to sufficiently represent the test data, which is not always the case.

In this chapter we instead focus on the theoretical aspects of continuous-time motion estimation, and think about how we can redesign the estimator for a potential improvement in accuracy and reduction in bias. In Section 5.2 we show algebraically that when using a white-noise-on-acceleration motion prior in the presence of acceleration, an interaction between the prior terms and the point-to-point measurement terms induces a bias to the estimation. With this in mind, we explore the idea of a white-noise-on-jerk motion prior, which is more suitable for motion estimation in scenarios that we know will have accelerations, such as in urban driving.

5.2 Interaction Between Prior and Measurements

Here we show that when the motion prior cannot sufficiently describe the underlying trajectory, the estimator will produce a bias as a result of the interaction between the measurement term and the prior term in the overall objective function.

Consider a very simple estimation problem with only one pose and one velocity as state variables. The robot, initially stationary, travels from $\boldsymbol{\mathcal{F}}_{\vec{0}}$ at timestep t_0 to $\boldsymbol{\mathcal{F}}_{\vec{1}}$ at timestep t_1 under constant acceleration. Suppose the robot takes a single point measurement $\mathbf{q}$ at t_0, and measures $\mathbf{p}$ at t_1, where $\mathbf{p}$ is a point match of $\mathbf{q}$. Our goal is to estimate $\mathbf{T}_1$ and $\boldsymbol{\varpi}_1$ using the point pair as well as a WNOA motion prior. Our state is

$$\mathbf{z} = \{\mathbf{T}_1, \boldsymbol{\varpi}_1\} \tag{5.1}$$

To keep the problem as simple as possible, let there be motion only in the direction of ρ_1, so the robot is travelling forward. We also assume $t_1 - t_0 = 1$ for simplicity. Suppose the robot is initially stationary at t_0, but has constant acceleration in the DOF of ρ_1 from t_0 to t_1. Using standard kinematics, we can define the following ground truth quantities:

$$\begin{aligned}
\boldsymbol{\xi}_{\text{gt},1} &= \begin{bmatrix} \frac{1}{2}a(t_1 - t_0)^2 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T, \\
\mathbf{T}_{\text{gt},1} &= \exp\left(\boldsymbol{\xi}_{\text{gt},1}^{\wedge}\right), \\
\boldsymbol{\varpi}_{\text{gt},1} &= \begin{bmatrix} a(t_1 - t_0) & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T,
\end{aligned} \tag{5.2}$$

where a is the acceleration in ρ_1. As shown in (5.2), the ground truth motion only occurs in the ρ_1 direction, but we wish to solve the problem using $SE(3)$ estimation in 6-DOF, without explicitly constraining the solution to any particular degree of freedom. We run Gauss-Newton for one iteration to solve for the perturbations. To do so, we can build the measurement error term as

$$\mathbf{g} = \mathbf{D}(\mathbf{p} - \mathbf{T}_{\text{op},1}\mathbf{q}), \tag{5.3}$$

where $\mathbf{T}_{\text{op},1}$ is the current operating point. Since we are only running Gauss-Newton for one iteration, the operating points are essentially the initial conditions. The measurement

Jacobian can be evaluated as

$$\begin{aligned}\frac{\partial \mathbf{g}}{\partial \delta\boldsymbol{\xi}_1}\Big|_{\mathbf{z}_{\text{op}}} &= -\mathbf{D}(\mathbf{T}_{\text{op},1}\mathbf{q})^{\odot} \\ &= \begin{bmatrix} -1 & 0 & 0 & 0 & -z & y \\ 0 & -1 & 0 & z & 0 & x \\ 0 & 0 & -1 & -y & x & 0 \end{bmatrix},\end{aligned} \tag{5.4}$$

where $\mathbf{T}_{\text{op},1}\mathbf{q} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$. There is also a prior term between t_0 and t_1, which can be evaluated as

$$\mathbf{e} = \begin{bmatrix} \ln\left(\mathbf{T}_{\text{op},1}^{-1}\right)^{\vee} \\ \boldsymbol{\mathcal{J}}_{1,0}^{-1}\boldsymbol{\varpi}_{\text{op},1} \end{bmatrix}, \tag{5.5}$$

where $\boldsymbol{\varpi}_0 = \mathbf{0}$ since we assumed the robot is initially stationary. The prior error Jacobian can be evaluated as

$$\mathbf{E} = \begin{bmatrix} -\boldsymbol{\mathcal{J}}_{1,0}^{-1}\boldsymbol{\mathcal{T}}_{1,0} & -\mathbf{1} & \boldsymbol{\mathcal{J}}_{1,0}^{-1} & \mathbf{0} \\ -\frac{1}{2}\boldsymbol{\mathcal{J}}_{1,0}^{-1}\boldsymbol{\mathcal{T}}_{1,0} & -\mathbf{1} & \frac{1}{2}\boldsymbol{\varpi}_{\text{op},1}^{\curlywedge}\boldsymbol{\mathcal{J}}_{1,0}^{-1} & \boldsymbol{\mathcal{J}}_{1,0}^{-1} \end{bmatrix}. \tag{5.6}$$

Given $\mathbf{g}, \mathbf{G}, \mathbf{e}$, and $\mathbf{E}$, we can build the linear system $\mathbf{A}\boldsymbol{\epsilon} = \mathbf{b}$, and solve for the optimal perturbation $\boldsymbol{\epsilon}^*$ as in Equation (2.11). When doing so, we make the further simplification that $\mathbf{Q}^{-1} = \mathbf{1}$ and $\mathbf{R}^{-1} = \mathbf{1}$. For the sake of argument, we do not use a zero measurement covariance, even though we assumed the measurements are noise-free.

If the state variables are initialized at the ground truth poses and velocities, and the measurement is noise-free, we have

$$\begin{aligned}\mathbf{T}_{\text{op},1} &= \mathbf{T}_{\text{gt},1}, \\ \mathbf{g} &= \mathbf{0}_{3\times 1}.\end{aligned} \tag{5.7}$$

Note that since there is constant acceleration between t_0 and t_1, the motion in this simple problem cannot be sufficiently modelled by a zero-acceleration-mean assumption. As a result, the prior error term $\mathbf{e}_{\text{op}}$ does not evaluate to $\mathbf{0}_{12\times 1}$ even when all state variables are set to be the ground truth.

Solving for $\boldsymbol{\epsilon}^*$ analytically in terms of scalar variables $\{x, y, z, a\}$, and extracting the

perturbation to each individual state variable, we have

$$\delta\boldsymbol{\xi}_1 = \frac{1}{m}\begin{bmatrix} -\frac{1}{4}a\Big((a^2-4x)^2+32(y^2+z^2+1)\Big) \\ ay(a+4x) \\ az(a+4x) \\ 0 \\ 8az \\ -8ay \end{bmatrix}, \tag{5.8}$$

where $m = (a^2-4x)^2 + 16(y^2+z^2+2)$.

Right away, we can make the observation that our simple problem results in perturbations to degrees of freedom where there is no motion, namely ρ_2, ρ_3, ϕ_2, and ϕ_3. Moreover, the perturbation to these degrees of freedom depend on $\begin{bmatrix} x & y & z \end{bmatrix}^T$, which is the Cartesian coordinate of the transformed point, $\mathbf{T}_{\text{op},1}\mathbf{q}$.

Now, consider a different case, where instead of having a point measurement, the robot is somehow able to measure its pose directly. For this case, the measurement error equation is

$$\mathbf{g} = \ln\big(\mathbf{T}_{\text{op},1}\mathbf{T}^{-1}_{\text{meas},1}\big)^{\vee}, \tag{5.9}$$

where $\mathbf{T}_{\text{meas},1}$ is the pose measurement. The measurement Jacobian can then be calculated as

$$\frac{\partial\,\mathbf{g}}{\partial\,\delta\boldsymbol{\xi}_1}\Big|_{\mathbf{z}_{\text{op}}} = \boldsymbol{\mathcal{J}}(\mathbf{g})^{-1}. \tag{5.10}$$

Again, if we initialize at ground truth, and the measurement is noise-free, we have

$$\begin{aligned} \mathbf{g} &= \mathbf{0}_{3\times 1}, \\ \frac{\partial\,\mathbf{g}}{\partial\,\delta\boldsymbol{\xi}_1}\Big|_{\mathbf{z}_{\text{op}}} &= \mathbf{1}_{6\times 6}. \end{aligned} \tag{5.11}$$

We make the same assumptions as before, and use the same prior error term and Jacobian for a WNOA prior as in Equations (5.5) and (5.6). Building the linear system $\mathbf{A}\boldsymbol{\epsilon} = \mathbf{b}$ as usual and solve for $\boldsymbol{\epsilon}^*$ analytically, we now have

$$\delta\boldsymbol{\xi}_1 = \begin{bmatrix} -\frac{3}{8}a^2 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T. \tag{5.12}$$

Equation (5.12) shows that under a different measurement model, we no longer have perturbations in directions other than ρ_1, even if there is acceleration in the ground truth motion between t_0 and t_1.

5.2.1 Observation

When there is acceleration in the trajectory segment considered, then the underlying trajectory cannot be sufficiently represented by a WNOA motion model. As a result, the measurement error terms and the prior error terms cannot both converge to zero (even when assuming the measurements are noise-free). Intuitively, assuming we have sufficient inlier measurements to estimate the trajectory with high accuracy, the prior terms in the objective function ends up pulling the estimated trajectory away from the ground truth. Most importantly, the result in (5.8) shows that, the optimal solution $\boldsymbol{\epsilon}^*$ to $\mathbf{A}\boldsymbol{\epsilon} = \mathbf{b}$ involves distributing the error in one degree of freedom to other degrees of freedom, effectively creating a bias to those degrees of freedom.

The perturbation to the other degrees of freedom is a function of the Cartesian coordinates of the point measurements. Given a ground truth trajectory constrained to the plane $z = 0$, the presence of point measurements out of the plane $z = 0$ may 'push' the estimated trajectory out of the plane. This effect is a result of the interaction between the point measurement terms and the prior terms in the Gauss-Newton update scheme. Specifically, the interaction comes from the presence of x, y, and z in the measurement Jacobian as in (5.4). If a different measurement model is used where the measurement Jacobian is not a function of the point coordinates, such as in (5.10), then this effect of interaction disappears, as shown in (5.12).

Results in (5.8) are derived assuming there is only one point measurement with Cartesian coordinate $\begin{bmatrix} x & y & z \end{bmatrix}^T$. A similar argument can be made for a point-cloud with centroid at $\begin{bmatrix} x & y & z \end{bmatrix}^T$. Assuming all point-pairs have equal weight in the objective function, if the centroid of the point-cloud is not at the origin, then motion estimation will result in a bias to certain degrees of freedom when the prior cannot sufficiently represent the underlying trajectory.

Consider a typical real-world dataset collected with a Velodyne lidar mounted on the top of a vehicle, such as in Figure 2.7. The measured points will have x and y coordinates more or less symmetrical about zero, since we can have point measurements in front and behind the vehicle, as well as to the left and to the right of the vehicle. However, as a result of the mounting height of the sensor, as well as the elevation angles of each laser in a Velodyne lidar, it is likely that the z centroid of a measured point-cloud will be negative. This will result in biases to ρ_3 and ϕ_2 when the vehicle experiences acceleration in ρ_1, as shown previously.

5.2.2 Verification Using Simple Problems in Simulation

Translational Acceleration

Here we verify results shown in Section 5.2 experimentally using simple problems in simulation. Consider a short trajectory segment with knot times $\{t_0, t_1, \ldots, t_N\}$. The robot is stationary at t_0, and has constant body-centric acceleration $\dot{\boldsymbol{\varpi}}$ in ρ_1 from t_0 to t_N. We can define the following ground truth state quantities:

$$\begin{aligned} \boldsymbol{\varpi}_{\text{gt},0} &= \mathbf{0}_{6\times 1}, \\ \dot{\boldsymbol{\varpi}} &= \begin{bmatrix} a_{\rho_1} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T, \\ \boldsymbol{\varpi}_{\text{gt},k} &= \Delta t_{k:0} \dot{\boldsymbol{\varpi}}, \\ \mathbf{T}_{\text{gt},k} &= \exp\left(\frac{1}{2}\Delta t_{k:0}^2 \dot{\boldsymbol{\varpi}}^{\wedge}\right). \end{aligned} \tag{5.13}$$

The robot takes point measurements at each discrete timestep t_k, where points measured at $t_1, t_2, \ldots, t_N$ are matched to points measured at t_0. We assume point matching is perfect and the measurements are noise-free. We aim to estimate the trajectory using sliding window optimization, where the objective function consists of measurement terms and WNOA prior terms, as usual.

For our simple problem, we have chosen $N = 10$, $\Delta t_{k:k-1} = 0.1\text{s}$, and $a_{\rho_1} = 3\text{m/s}^2$. The Cartesian coordinates of the measured points are generated from a uniform random distribution. To demonstrate the effects of the interaction between the measurement terms, the points are generated such that their z coordinates are not symmetrical about zero. The x, y, z coordinates of each point are generated such that they have the following range:

$$\begin{aligned} x &\sim (-20\text{m}, 20\text{m}), \\ y &\sim (-20\text{m}, 20\text{m}), \\ z &\sim (-20\text{m}, 0\text{m}). \end{aligned} \tag{5.14}$$

Finally, we can define an error at the end of the trajectory segment:

$$\begin{aligned} \mathbf{T}_{\text{err},N} &= \mathbf{T}_{\text{gt},N} \hat{\mathbf{T}}_N^{-1}, \\ \boldsymbol{\xi}_{\text{err},N} &= \ln(\mathbf{T}_{\text{err},N})^{\vee}, \end{aligned} \tag{5.15}$$

where $\hat{\mathbf{T}}_N$ is the posterior estimate of $\mathbf{T}_N$. Using the same notation as in Equations (4.12)

and (4.13), we can look at the error in each DOF of $\boldsymbol{\xi}_{\text{err},N}$. Define

$$\boldsymbol{\xi}_{\text{err},N} = \begin{bmatrix} e_{\rho_1} & e_{\rho_2} & e_{\rho_3} & e_{\phi_1} & e_{\phi_2} & e_{\phi_3} \end{bmatrix}^T. \tag{5.16}$$

While the ground truth motion takes place on the plane $z = 0$, the estimated trajectory will not be constrained to the plane since we have point measurements out of the plane, as explained in Section 5.2. $10,000$ runs of the simple simulated problems are carried out, and a histogram distribution for each DOF of $\boldsymbol{\xi}_{\text{err},N}$ is shown in Figure 5.1. Other than e_{ρ_1}, the error distribution is not symmetrical about zero for e_{ρ_3} and e_{ϕ_2}, suggesting we have a positive bias in these degrees of freedom in the estimation. The results here are in accordance with results derived in Equation (5.8), which shows the perturbation to ρ_3 and ϕ_2 are functions of the z coordinate of point measurements, when there is acceleration in ρ_1.

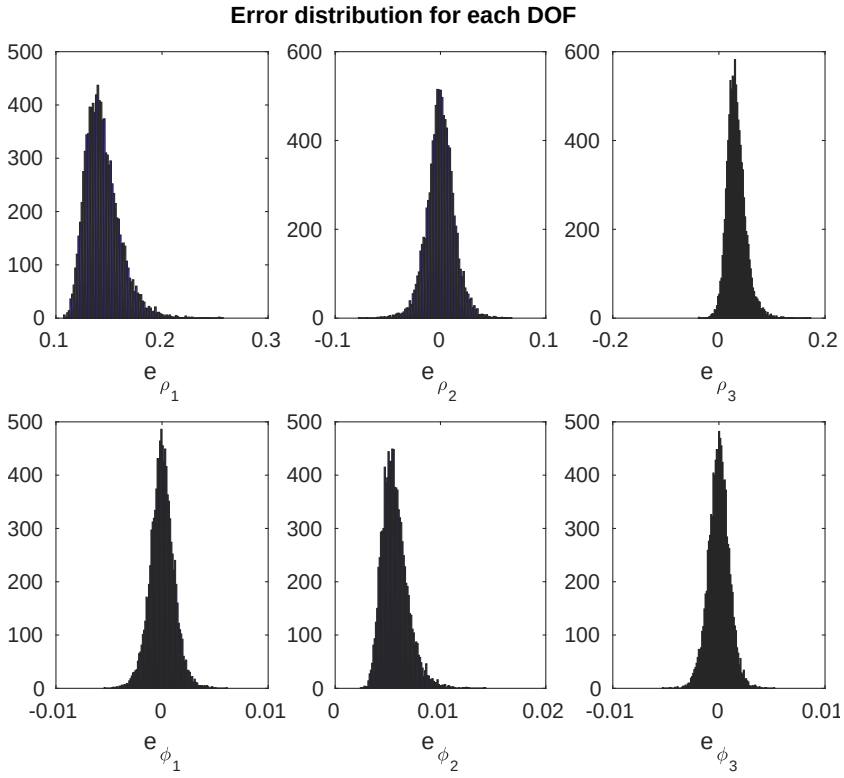

Figure 5.1: Error distribution for each DOF of $\boldsymbol{\xi}_{\text{err},N}$ with 10,000 simulated runs. Points have z coordinates between -20m and 0.

We can repeat this simple experiment using points with $z \sim (0, 20\text{m})$, while keeping the range for x and y coordinates the same. The distribution of $\boldsymbol{\xi}_{\text{err},N}$ over $10,000$ runs using points with $z \sim (0, 20\text{m})$ is shown in Figure 5.2. It can be seen that while the distribution for e_{ρ_1} stays the same, the distribution for e_{ρ_3} and e_{ϕ_2} are essentially mirrored about zero comparing to Figure 5.1, where we now have a negative bias in these

degrees of freedom.

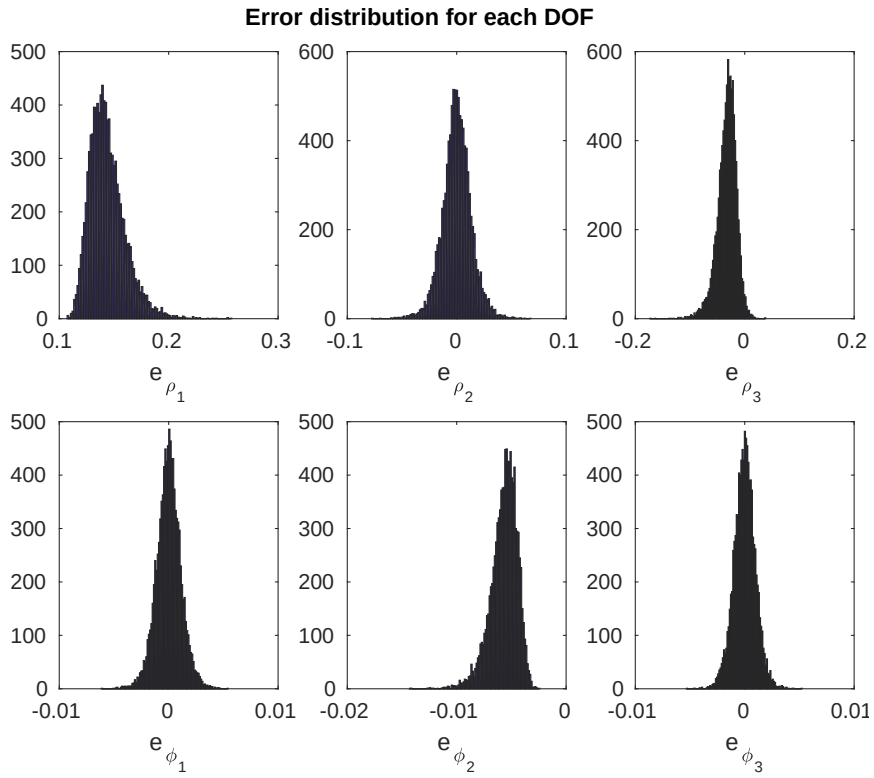

Figure 5.2: Error distribution for each DOF of $\boldsymbol{\xi}_{\text{err},N}$ with 10,000 simulated runs. Points have z coordinates between 0 and 20m.

A more intuitive visualization would be to compare the final estimated z position of the robot at t_N as seen from $\boldsymbol{\mathcal{F}}_{\vec{0}}$, for point measurements with different z coordinates. This is shown in Figure 5.3. It can be seen in Figure 5.3 that for this simple problem, although the underlying trajectory takes place on the plane $z = 0$, we have a positive bias in the estimated z position of the robot if we use point measurements above the plane, and a negative bias in the estimated z position if we use point measurements below the plane.

Rotational Acceleration

A robot in motion may also encounter rotational, or angular body-centric acceleration. Here we repeat the experiment for the case where the robot accelerates into a turn about its $z-$ axis. The robot is moving forward at t_0 with a body-centric velocity $\boldsymbol{\varpi}_0$ in ρ_1, and has constant body-centric acceleration $\dot{\boldsymbol{\varpi}}$ in ϕ_3 from t_0 to t_N. We can define the

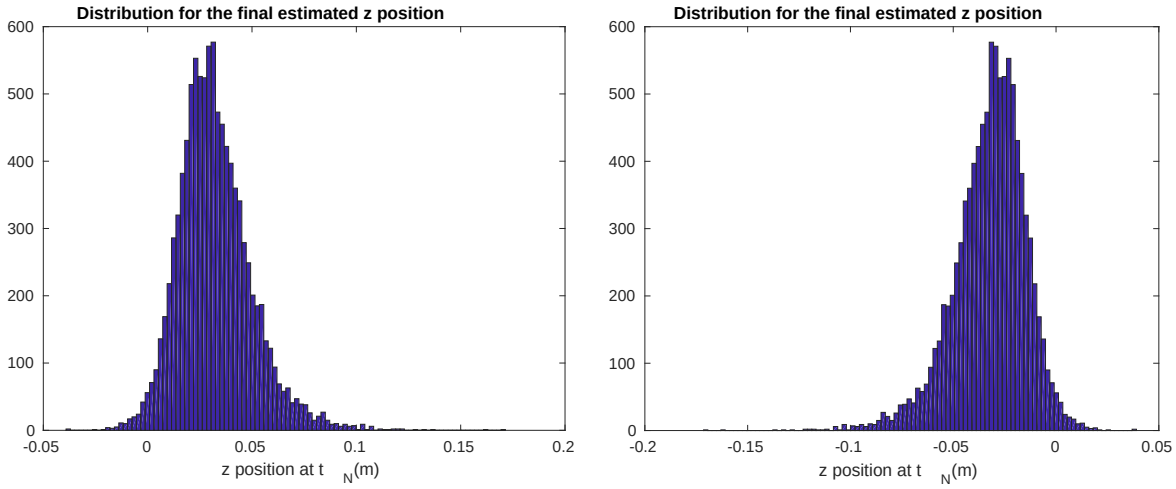

Figure 5.3: Estimated z position of the robot at t_N when the motion has acceleration in ρ_1, using point measurements with z coordinates $\sim (-20\text{m}, 0)$ (left) and point measurements with z coordinates $\sim (0, 20\text{m})$ (right)

following ground truth state quantities:

$$\begin{aligned} \boldsymbol{\varpi}_{\text{gt},0} &= \begin{bmatrix} v_{\rho_1} & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T, \\ \dot{\boldsymbol{\varpi}} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & a_{\phi_3} \end{bmatrix}^T, \\ \boldsymbol{\varpi}_{\text{gt},k} &= \Delta t_{k:0} \dot{\boldsymbol{\varpi}}, \\ \mathbf{T}_{\text{gt},k} &= \exp\left(\left(\Delta t_{k:0} \boldsymbol{\varpi}_{\text{gt},0} + \frac{1}{2} \Delta t_{k:0}^2 \dot{\boldsymbol{\varpi}}\right)^\wedge\right). \end{aligned} \tag{5.17}$$

We have chosen $v_{\rho_1} = 1\text{m/s}$, and $a_{\phi_3} = 0.3\text{rad/s}^2$. Similarly, 10,000 runs of the simple estimation problem are simulated each for points with z coordinates greater than zero and less than zero. The distribution for the final estimated z position of the robot is shown in Figure 5.4. It can be seen that the bias in the estimated z position when we have rotational acceleration is also dependent on the Cartesian coordinates of points.

5.2.3 Verification Using CARLA Dataset

To further verify our theory regarding the interaction between prior and measurement terms, we look at data generated from a more realistic simulation environment using the CARLA simulator [12]. Our implementation of the CARLA simulator produces motion-distorted point-clouds in urban driving scenarios with a resolution similar to that of a Velodyne HDL-64E sensor. Figure 5.5 shows an example of a point-cloud measured by the simulated Velodyne lidar.

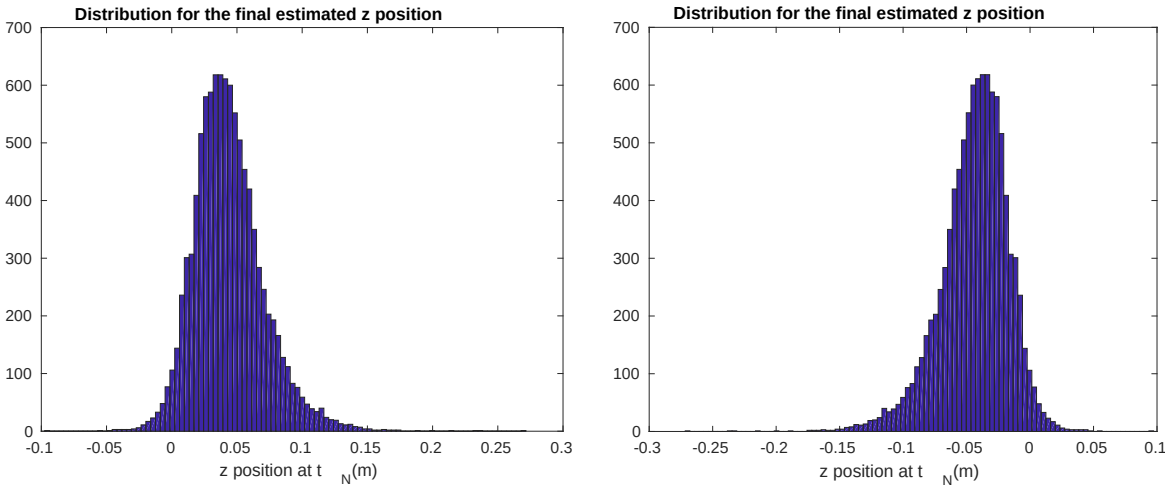

Figure 5.4: Estimated z position of the robot at t_N when the motion has acceleration in ϕ_3, for using point measurements with z coordinates $\sim (-20\text{m}, 0)$ (left) and point measurements with z coordinates $\sim (0, 20\text{m})$ (right)

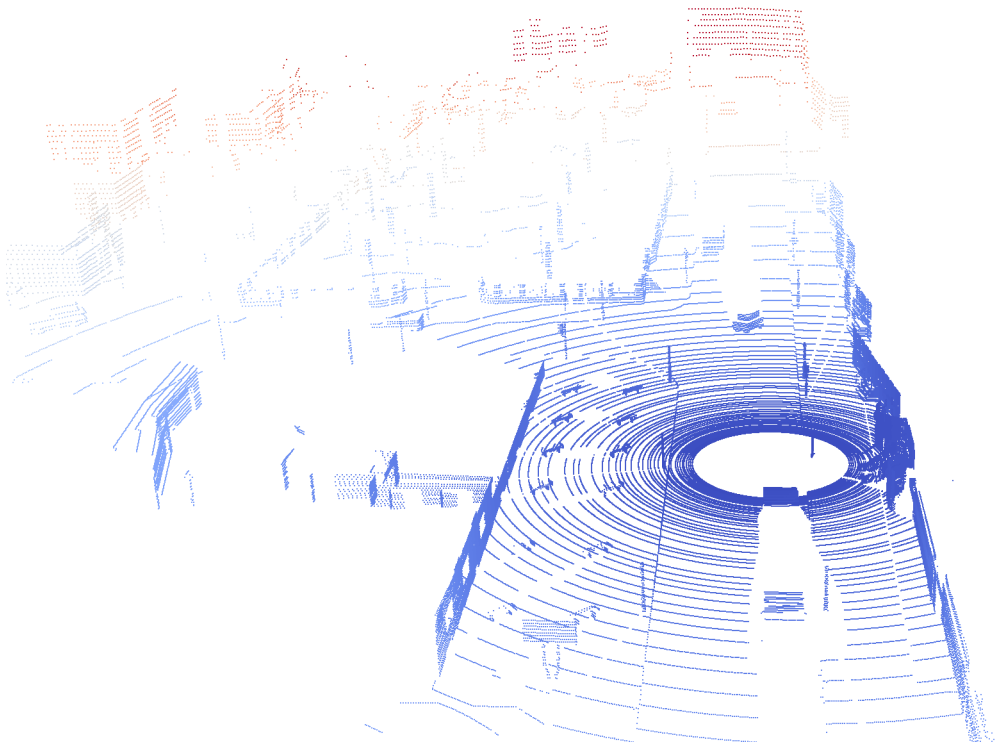

Figure 5.5: A 3D point-cloud generated using the CARLA simulator colored by range. Objects shown in the scene include the exterior of walls and buildings, cars, billboards, and more.

A particular advantage of using simulated data is that we have access to sensor data that are noise-free and perfectly calibrated. This helps us isolate certain potential sources of bias. Without the presence of sensor noise and calibration error, any biases that still exist are primarily caused by inherent limitations of the state estimator. We argue that the inherent estimator limitations are largely contributed by the fact that our motion prior cannot sufficiently describe the underlying trajectory in certain cases. This in turn causes an interaction between the prior terms and the measurement terms that induces biases to the estimated motion.

A dataset with approximately 6 minutes of driving was collected. The simulated world has a perfectly flat ground plane, thus the ground truth z position of the vehicle stays at zero throughout the entire dataset. Similar to driving a car in real world, the simulated motion has translational acceleration when the car speeds up or slows down, and angular acceleration when the car enters or exits from a turn. A top-down view of the estimated trajectory and the ground truth trajectory of the vehicle is shown in Figure 5.6. A 3D view of the estimated trajectory compared to the ground truth is shown in Figure 5.7. The bias in roll and pitch in the estimated trajectory can be seen in Figure 5.7, as the estimated trajectory clearly drifts out of the plane $z = 0$.

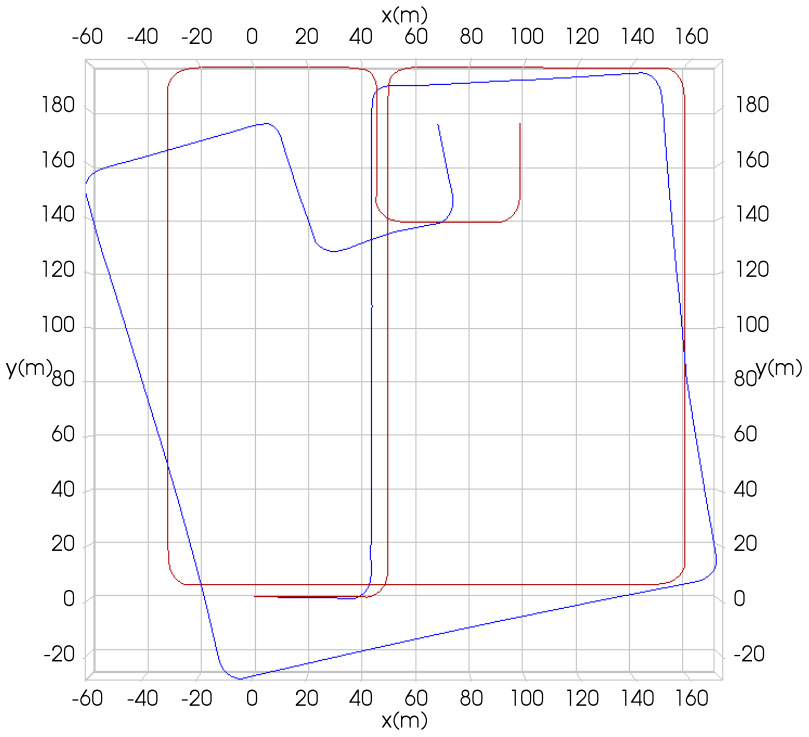

Figure 5.6: Top-down view of the estimated trajectory (blue) vs. ground truth trajectory (red) for a dataset generated using CARLA.

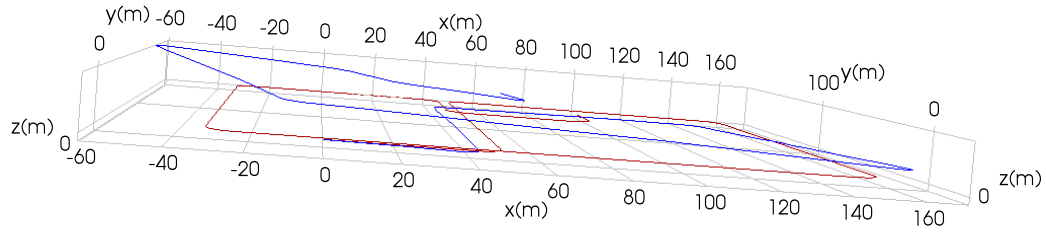

Figure 5.7: 3D view of the estimated trajectory (blue) vs. ground truth trajectory (red) for a dataset generated using CARLA.

To demonstrate the dependence of bias on the Cartesian coordinates of point measurements, we have artificially added a $+10$m offset to the z coordinate of all points measured, and repeated the estimation keeping everything else the same. We also experimented with adding a -10m offset to the z coordinate of points and repeated the estimation again. The effect of adding a constant offset to measurements is equivalent to shifting the estimation frame by an opposite offset. The results are shown in Figure 5.8.

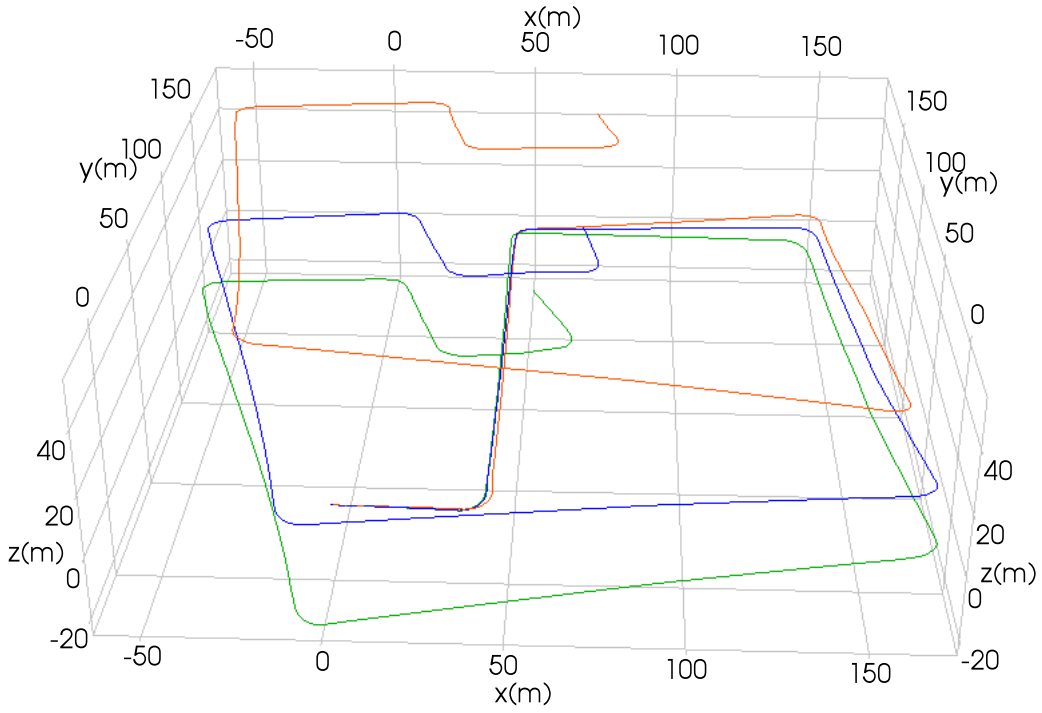

Figure 5.8: 3D view of the estimated trajectory using original point coordinates (blue), estimated trajectory when the z coordinates of points are offset by $+10$m (green), and estimated trajectory when the z coordinates of points are offset by -10m (orange).

As shown in Figure 5.8, the behaviour of estimator bias changes drastically when the

Cartesian coordinates of point measurements are shifted by a constant offset. When a positive offset is added to the z coordinates of points, biases in roll and pitch result in the estimated positions of the vehicle having a consistent downward drift. On the other hand, when a negative offset is added, the biases result in the estimated positions having a consistent upward drift.

Again, since the point-clouds are motion-distorted, a motion prior is needed to solve the lidar-only trajectory estimation problem. However, it can be shown easily that if the measurement terms alone are sufficient to estimate the motion, such as a typical visual SLAM problem using a global shutter camera, a shift in the estimation frame will not cause the estimated motion to drastically drift towards a particular direction. When WNOA prior terms are used in the optimization problem, and the underlying trajectory cannot be sufficiently treated as zero acceleration, the interaction between prior and measurement terms create biases to the estimated motion that depend on the coordinates of point measurements.

5.3 Derivation of a White-Noise-On-Jerk Prior

5.3.1 Motivation

In section 1.1 we explained why it is essential to introduce a motion prior in lidar-only estimation. So far we have used a white-noise-on-acceleration motion prior, namely one where the prior mean encourages constant velocity. However, for a large number of outdoor robotic applications, such as estimating the motion of a car in urban driving scenarios, we know the motion will contain acceleration. This is against the fundamental assumption of a WNOA motion prior. Moreover, we have shown in section 5.2 that such violation causes an interaction between the prior terms and the measurement terms, which in turn creates an unwanted bias.

In this section we redesign the motion prior altogether. We derive a white-noise-on-jerk motion prior, where the prior mean encourages constant acceleration. Our hope is that a WNOJ prior is more suitable for describing motion which we know contains non-zero acceleration.

5.3.2 A Class of Exactly Sparse GP priors

Our goal is to employ a class of GP priors that leads to an efficient formulation and a simple solution [5] [6]. This class of GP priors is based on linear time-invariant (LTI)

stochastic differential equations (SDEs) of the form

$$\begin{aligned} \dot{\boldsymbol{\gamma}}(t) &= \boldsymbol{A}\boldsymbol{\gamma}(t) + \boldsymbol{B}\mathbf{u}(t) + \boldsymbol{L}\mathbf{w}(t), \\ \mathbf{w}(t) &\sim \mathcal{GP}(0, \mathbf{Q}_c\delta(t-t')), \end{aligned} \tag{5.18}$$

where $\boldsymbol{\gamma}(t)$ is the state, $\mathbf{u}(t)$ is a known exogenous input, and $\mathbf{w}(t)$ is a zero-mean, white-noise GP with power spectral density matrix, $\mathbf{Q}_c$. If $\mathbf{u}(t) = \mathbf{0}$, then for the mean function we have the simple solution

$$\check{\boldsymbol{\gamma}}(\tau) = \boldsymbol{\Phi}(\tau, t_k)\check{\boldsymbol{\gamma}}(t_k), \tag{5.19}$$

where $\check{\boldsymbol{\gamma}}$ is the prior mean, and $\boldsymbol{\Phi}(\tau, t_k)$ is the *state transition function*. We can write the prior covariance as

$$\check{\mathbf{P}} = \mathbf{F}\mathbf{Q}\mathbf{F}^T, \tag{5.20}$$

where

$$\mathbf{F} = \begin{bmatrix} \mathbf{1} & & & & & \\ \boldsymbol{\Phi}(t_1,t_0) & \mathbf{1} & & & & \\ \boldsymbol{\Phi}(t_2,t_0) & \boldsymbol{\Phi}(t_2,t_1) & \mathbf{1} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ \boldsymbol{\Phi}(t_{K-1},t_0) & \boldsymbol{\Phi}(t_{K-1},t_1) & \boldsymbol{\Phi}(t_{K-1},t_2) & \dots & \mathbf{1} & \\ \boldsymbol{\Phi}(t_K,t_0) & \boldsymbol{\Phi}(t_K,t_1) & \boldsymbol{\Phi}(t_K,t_2) & \dots & \boldsymbol{\Phi}(t_K,t_{K-1}) & \mathbf{1} \end{bmatrix}, \tag{5.21}$$

and

$$\mathbf{Q} = \operatorname{diag}(\check{\mathbf{P}}_0, \mathbf{Q}_1, \mathbf{Q}_2, \dots, \mathbf{Q}_k), \tag{5.22}$$

with

$$\mathbf{Q}_k = \int_0^{\Delta t_{k:k-1}} \exp(\boldsymbol{A}(\Delta t_{k:k-1} - s))\boldsymbol{L}\mathbf{Q}_c\boldsymbol{L}^T \exp\big(\boldsymbol{A}(\Delta t_{k:k-1} - s)^T\big)\, ds. \tag{5.23}$$

WNOA GP Prior for SE(3)

In $SE(3)$, a physically-motivated GP prior is the following SDE:

$$\begin{aligned} \dot{\mathbf{T}}(t) &= \boldsymbol{\varpi}(t)^\wedge \mathbf{T}(t), \\ \dot{\boldsymbol{\varpi}}(t) &= \mathbf{w}'(t), \quad \mathbf{w}(t) \sim \mathcal{GP}(0, \mathbf{Q}_c\delta(t-t')), \end{aligned} \tag{5.24}$$

As shown before, the state is

$$\mathbf{z}(t) = \{\mathbf{T}(t), \boldsymbol{\varpi}(t)\}. \tag{5.25}$$

However, it can be seen that the SDE in (5.24) is nonlinear, and therefore cannot be cast into the form of (5.18) and solved efficiently [3]. Instead, [3] defines a *local* pose variable:

$$\boldsymbol{\xi}_i(t) = \ln\big(\mathbf{T}(t)\mathbf{T}_i^{-1}\big)^{\vee}, \quad t_i \leq t \leq t_{i+1} \tag{5.26}$$

which is a function of the *global* pose variables. Using *local* variables, [3] defines a sequence of *local* priors that can be cast into a LTI SDE of the form in (5.18), with

$$\boldsymbol{\gamma}_i(t) = \begin{bmatrix} \boldsymbol{\xi}_i(t) \\ \dot{\boldsymbol{\xi}}_i(t) \end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix} \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \boldsymbol{L} = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}, \tag{5.27}$$

where $\boldsymbol{\gamma}_i(t)$ is defined as the *local* state. Under this formulation, we have white-noise on the second derivative of $\boldsymbol{\xi}_i(t)$, i.e. $\ddot{\boldsymbol{\xi}}_i(t) = \mathbf{w}(t)$. Furthermore, we have the following [3]:

$$\dot{\boldsymbol{\xi}}_i(t) = \boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_i(t))^{-1}\boldsymbol{\varpi}(t). \tag{5.28}$$

The state transition function can be computed as in [6],

$$\boldsymbol{\Phi}(t, t_i) = \exp(\boldsymbol{A}\Delta t_i) = \begin{bmatrix} \mathbf{1} & \Delta t_i \mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}, \tag{5.29}$$

and the inverse covariance matrix is [7] [3]

$$\mathbf{Q}_i(t)^{-1} = \begin{bmatrix} 12\Delta t_i^{-3}\mathbf{Q}_c^{-1} & -6\Delta t_i^{-2}\mathbf{Q}_c^{-1} \\ -6\Delta t_i^{-2}\mathbf{Q}_c^{-1} & 4\Delta t_i^{-1}\mathbf{Q}_c^{-1} \end{bmatrix}, \tag{5.30}$$

where $\Delta t_i = t - t_i$. As shown in Equation (2.3), the objective function consists of measurement and prior error terms. In terms of *local* pose variables, the prior error can be computed as

$$\mathbf{e}_i = \boldsymbol{\gamma}_i(t_{i+1}) - \hat{\boldsymbol{\gamma}}_i(t_{i+1}) - \boldsymbol{\Phi}(t_{i+1}, t_i)(\boldsymbol{\gamma}_i(t_i) - \hat{\boldsymbol{\gamma}}_i(t_i)), \tag{5.31}$$

where the *local* state variables are defined as [3]

$$\boldsymbol{\gamma}_i(t_i) = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\varpi}_i \end{bmatrix}, \quad \boldsymbol{\gamma}_i(t_{i+1}) = \begin{bmatrix} \ln(\mathbf{T}_{i+1,i})^{\vee} \\ \boldsymbol{\mathcal{J}}_{i+1,i}^{-1}\boldsymbol{\varpi}_{i+1} \end{bmatrix}. \tag{5.32}$$

Using the relationship between *local* and *global* state variables, the prior error term for the WNOA prior can be expressed as in Equation (2.14) [3].

In terms of *local* state variables, we can interpolate for the state as [3]

$$\boldsymbol{\gamma}_i(\tau) = \boldsymbol{\Lambda}(t)\boldsymbol{\gamma}_i(t_i) + \boldsymbol{\Omega}(t)\boldsymbol{\gamma}_i(t_{i+1}), \quad t_i \leq \tau \leq t_{i+1}, \tag{5.33}$$

where $\boldsymbol{\Lambda}(\tau) \in \mathbb{R}^{12\times 12}$ and $\boldsymbol{\Omega}(\tau) \in \mathbb{R}^{12\times 12}$ are [5]

$$\begin{aligned} \boldsymbol{\Lambda}(\tau) &= \boldsymbol{\Phi}(\tau, t_i) - \boldsymbol{\Omega}(\tau)\boldsymbol{\Phi}(t_{i+1}, t_i), \\ \boldsymbol{\Omega}(\tau) &= \mathbf{Q}_i(\tau)\boldsymbol{\Phi}(t_{i+1}, t)^T\mathbf{Q}_i(\tau)^{-1}. \end{aligned} \tag{5.34}$$

Again, using our knowledge on the relationship between *local* and *global* state variables as in (5.32), we can re-formulate (5.33) using *global* state variables. While interpolating for the body-centric velocity at an arbitrary time might be of interest to certain applications, for lidar-only odometry we are mainly interested in pose interpolation:

$$\mathbf{T}_\tau = \exp\left((\boldsymbol{\Lambda}_{12}(\tau)\boldsymbol{\varpi}_i + \boldsymbol{\Omega}_{11}(\tau)\ln(\mathbf{T}_{i+1,i})^\vee + \boldsymbol{\Omega}_{12}(\tau)\boldsymbol{\mathcal{J}}_{i+1,i}^{-1}\boldsymbol{\varpi}_{i+1})^\wedge\right)\mathbf{T}_i, \tag{5.35}$$

where we have

$$\boldsymbol{\Omega}(\tau) = \begin{bmatrix} \boldsymbol{\Omega}_{11}(\tau) & \boldsymbol{\Omega}_{12}(\tau) \\ \boldsymbol{\Omega}_{21}(\tau) & \boldsymbol{\Omega}_{22}(\tau) \end{bmatrix}, \quad \boldsymbol{\Lambda}(\tau) = \begin{bmatrix} \boldsymbol{\Lambda}_{11}(\tau) & \boldsymbol{\Lambda}_{12}(\tau) \\ \boldsymbol{\Lambda}_{21}(\tau) & \boldsymbol{\Lambda}_{22}(\tau) \end{bmatrix}. \tag{5.36}$$

5.3.3 Transition Function and Covariance Matrix

Instead of modelling the acceleration as a zero-mean, white-noise Gaussian process as in the case of a WNOA prior [3], we now explicitly estimate the following state:

$$\mathbf{x}(t) = \{\mathbf{T}(t), \boldsymbol{\varpi}(t), \dot{\boldsymbol{\varpi}}(t)\}, \tag{5.37}$$

where $\dot{\boldsymbol{\varpi}}(t) \in \mathbb{R}^6$ is the body-centric acceleration.

Extending the idea of *local* pose variables as presented in Section 5.3.2, we can define a sequence of *local* white-noise-on-jerk priors as a LTI SDE in the form of (5.18):

$$\boldsymbol{\gamma}_i(t) := \begin{bmatrix} \boldsymbol{\xi}_i(t) \\ \dot{\boldsymbol{\xi}}_i(t) \\ \ddot{\boldsymbol{\xi}}_i(t) \end{bmatrix} \quad \boldsymbol{A} = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \boldsymbol{L} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix}. \tag{5.38}$$

We now have white-noise on the third derivative (jerk) of $\boldsymbol{\xi}_i(t)$, $\dddot{\boldsymbol{\xi}}_i(t) = \mathbf{w}(t)$, where $\mathbf{w}(t) \sim \mathcal{GP}(0, \mathbf{Q}_c\delta(t-t'))$.

For the WNOJ prior, the state transition function is now

$$\boldsymbol{\Phi}(t, t_i) = \exp(\boldsymbol{A}\Delta t_i) = \begin{bmatrix} \mathbf{1} & \Delta t_i \mathbf{1} & \frac{1}{2}\Delta t_i^2 \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \Delta t_i \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}. \tag{5.39}$$

For the covariance matrix at time t_k, we have

$$\begin{aligned}
\mathbf{Q}_k &= \int_0^{\Delta t_{k:k-1}} \exp(\boldsymbol{A}(\Delta t_{k:k-1} - s)) \boldsymbol{L}\mathbf{Q}_c\boldsymbol{L}^T \exp\big(\boldsymbol{A}(\Delta t_{k:k-1} - s)^T\big)\, ds \\
&= \int_0^{\Delta t_{k:k-1}} \begin{bmatrix} \mathbf{1} & (\Delta t_{k:k-1} - s)\mathbf{1} & \frac{1}{2}(\Delta t_{k:k-1} - s)^2 \mathbf{1} \\ \mathbf{0} & \mathbf{1} & (\Delta t_{k:k-1} - s)\mathbf{1} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \mathbf{Q}_c \\
&\quad \times \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ (\Delta t_{k:k-1} - s)\mathbf{1} & \mathbf{1} & \mathbf{0} \\ \frac{1}{2}(\Delta t_{k:k-1} - s)^2 \mathbf{1} & (\Delta t_{k:k-1} - s)\mathbf{1} & \mathbf{1} \end{bmatrix} ds \\
&= \int_0^{\Delta t_{k:k-1}} \begin{bmatrix} \frac{1}{4}(\Delta t_{k:k-1} - s)^4 \mathbf{Q}_c & \frac{1}{2}(\Delta t_{k:k-1} - s)^3 \mathbf{Q}_c & \frac{1}{2}(\Delta t_{k:k-1} - s)^2 \mathbf{Q}_c \\ \frac{1}{2}(\Delta t_{k:k-1} - s)^3 \mathbf{Q}_c & (\Delta t_{k:k-1} - s)^2 \mathbf{Q}_c & (\Delta t_{k:k-1} - s)\mathbf{Q}_c \\ \frac{1}{2}(\Delta t_{k:k-1} - s)^2 \mathbf{Q}_c & (\Delta t_{k:k-1} - s)\mathbf{Q}_c & \mathbf{Q}_c \end{bmatrix} ds \\
&= \begin{bmatrix} \frac{1}{20}\Delta t_{k:k-1}^5 \mathbf{Q}_c & \frac{1}{8}\Delta t_{k:k-1}^4 \mathbf{Q}_c & \frac{1}{6}\Delta t_{k:k-1}^3 \mathbf{Q}_c \\ \frac{1}{8}\Delta t_{k:k-1}^4 \mathbf{Q}_c & \frac{1}{3}\Delta t_{k:k-1}^3 \mathbf{Q}_c & \frac{1}{2}\Delta t_{k:k-1}^2 \mathbf{Q}_c \\ \frac{1}{6}\Delta t_{k:k-1}^3 \mathbf{Q}_c & \frac{1}{2}\Delta t_{k:k-1}^2 \mathbf{Q}_c & \Delta t_{k:k-1} \mathbf{Q}_c \end{bmatrix}.
\end{aligned} \tag{5.40}$$

The inverse covariance matrix is

$$\mathbf{Q}_k^{-1} = \begin{bmatrix} 720\Delta t_{k:k-1}^{-5} \mathbf{Q}_c^{-1} & -360\Delta t_{k:k-1}^{-4} \mathbf{Q}_c^{-1} & 60\Delta t_{k:k-1}^{-3} \mathbf{Q}_c^{-1} \\ -360\Delta t_{k:k-1}^{-4} \mathbf{Q}_c^{-1} & 192\Delta t_{k:k-1}^{-3} \mathbf{Q}_c^{-1} & -36\Delta t_{k:k-1}^{-2} \mathbf{Q}_c^{-1} \\ 60\Delta t_{k:k-1}^{-3} \mathbf{Q}_c^{-1} & -36\Delta t_{k:k-1}^{-2} \mathbf{Q}_c^{-1} & 9\Delta t_{k:k-1}^{-1} \mathbf{Q}_c^{-1} \end{bmatrix}. \tag{5.41}$$

Figure 5.9 shows trajectories sampled from a white-noise-on-jerk prior distribution where the prior mean is constant-acceleration, compared with trajectories sampled from a white-noise-on-acceleration prior distribution where the prior mean is constant-velocity. We argue that the WNOJ prior is more suitable for representing motion with non-zero acceleration trajectory sections, such as in urban driving.

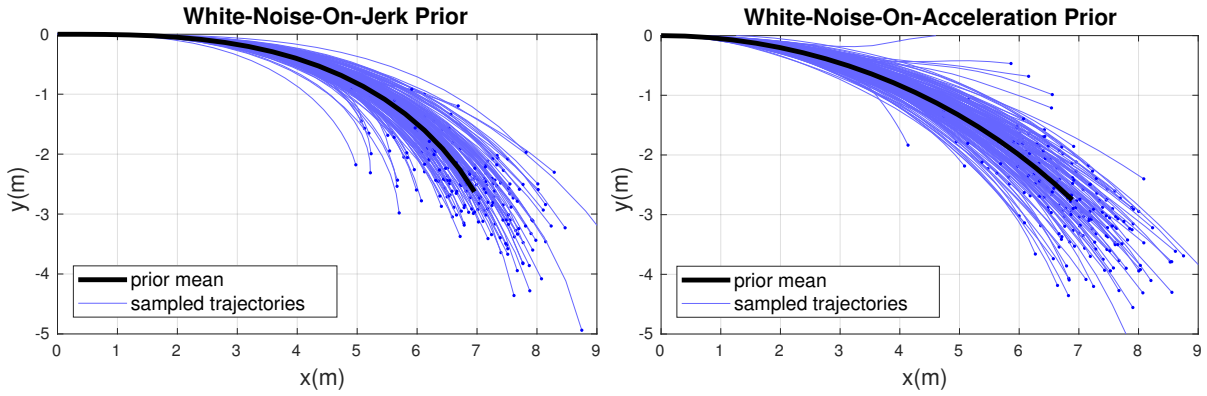

Figure 5.9: Existing formulations of STEAM use a white-noise-on-acceleration motion prior (right), which have trouble representing trajectories with non-zero acceleration, such as in the motion of a vehicle in urban driving. We propose a white-noise-on-jerk motion prior (left), which is more suitable for representing these types of trajectories.

5.3.4 Motion Prior Error Term

In *local* pose variables, the prior error term is the same as in (5.31). We wish to then express the prior error in terms of $\mathbf{T}$, $\boldsymbol{\varpi}$, and $\dot{\boldsymbol{\varpi}}$. The relationship between $\boldsymbol{\xi}_i(t)$ and $\dot{\boldsymbol{\xi}}_i(t)$ and *global* state variables are shown in Equations (5.26) and (5.28). To express $\ddot{\boldsymbol{\xi}}_i(t)$ in terms of *global* state variables, we have

$$\begin{aligned}\ddot{\boldsymbol{\xi}}_k(t) &= \frac{d}{dt}(\dot{\boldsymbol{\xi}}_k(t)) = \frac{d}{dt}(\boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_k(t))^{-1}\boldsymbol{\varpi}(t)) \\ &= \frac{d}{dt}(\boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_k(t))^{-1})\boldsymbol{\varpi}(t) + \boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_k(t))^{-1}\dot{\boldsymbol{\varpi}}(t).\end{aligned} \tag{5.42}$$

We can write $\boldsymbol{\mathcal{J}}^{-1}$ as a power-series expansion:

$$\boldsymbol{\mathcal{J}}^{-1} = \sum_{n=0}^{\infty} \frac{B_n}{n!}(\boldsymbol{\xi}^{\curlywedge})^n = B_0 + \frac{B_1}{1!}\boldsymbol{\xi}^{\curlywedge} + \frac{B_2}{2!}(\boldsymbol{\xi}^{\curlywedge})^2 + \frac{B_3}{3!}(\boldsymbol{\xi}^{\curlywedge})^3 + \cdots, \tag{5.43}$$

where the coefficients, B_n, are the *Bernoulli numbers*. The operator $(\cdot)^{\curlywedge}$ is defined as [6]

$$\mathbf{x}^{\curlywedge} = \begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix}^{\curlywedge} = \begin{bmatrix}\mathbf{v}^{\wedge} & \mathbf{u}^{\wedge}\\ \mathbf{0} & \mathbf{v}^{\wedge}\end{bmatrix}. \tag{5.44}$$

It can be shown easily that $\frac{d}{dt}(\mathbf{u}^{\wedge}) = \dot{\mathbf{u}}^{\wedge}$ for $\mathbf{u} \in \mathbb{R}^3$, and $\frac{d}{dt}(\mathbf{x}^{\curlywedge}) = \dot{\mathbf{x}}^{\curlywedge}$ for $\mathbf{x} \in \mathbb{R}^6$. Therefore, the time derivative is

$$\frac{d}{dt}(\boldsymbol{\mathcal{J}}^{-1}) = \frac{B_1}{1!}\dot{\boldsymbol{\xi}}^{\curlywedge} + \frac{B_2}{2!}(\dot{\boldsymbol{\xi}}^{\curlywedge}\boldsymbol{\xi}^{\curlywedge} + \boldsymbol{\xi}^{\curlywedge}\dot{\boldsymbol{\xi}}^{\curlywedge}) + \frac{B_3}{3!}\left(\dot{\boldsymbol{\xi}}^{\curlywedge}(\boldsymbol{\xi}^{\curlywedge})^2 + \boldsymbol{\xi}^{\curlywedge}\dot{\boldsymbol{\xi}}^{\curlywedge}\boldsymbol{\xi}^{\curlywedge} + (\boldsymbol{\xi}^{\curlywedge})^2\dot{\boldsymbol{\xi}}^{\curlywedge}\right) + \cdots. \tag{5.45}$$

As it turns out, we cannot express $\frac{d}{dt}(\boldsymbol{\mathcal{J}}^{-1})$ nicely in terms of $\boldsymbol{\mathcal{J}}^{-1}$ or $\boldsymbol{\mathcal{J}}$, which are familiar terms with which to work. We thus resort to making the approximation that $\boldsymbol{\mathcal{J}}^{-1} \approx \mathbf{1} - \frac{1}{2}\boldsymbol{\xi}^{\curlywedge}$. Under this approximation, we have

$$\frac{d}{dt}(\boldsymbol{\mathcal{J}}^{-1}) \approx -\frac{1}{2}\dot{\boldsymbol{\xi}}^{\curlywedge}, \tag{5.46}$$

and

$$\begin{aligned}
\ddot{\boldsymbol{\xi}}_k(t) &= \frac{d}{dt}(\boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_k(t))^{-1})\boldsymbol{\varpi}(t) + \boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_k(t))^{-1}\dot{\boldsymbol{\varpi}}(t) \\
&\approx -\frac{1}{2}\dot{\boldsymbol{\xi}}_k(t)^{\curlywedge}\boldsymbol{\varpi}(t) + \boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_k(t))^{-1}\dot{\boldsymbol{\varpi}}(t) \\
&= -\frac{1}{2}(\boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_k(t))^{-1}\boldsymbol{\varpi}(t))^{\curlywedge}\boldsymbol{\varpi}(t) + \boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_k(t))^{-1}\dot{\boldsymbol{\varpi}}(t).
\end{aligned} \tag{5.47}$$

In terms of $\mathbf{T}, \boldsymbol{\varpi}$, and $\dot{\boldsymbol{\varpi}}$, our local state variable is

$$\boldsymbol{\gamma}_k(t) = \begin{bmatrix} \boldsymbol{\xi}_k(t) \\ \dot{\boldsymbol{\xi}}_k(t) \\ \ddot{\boldsymbol{\xi}}_k(t) \end{bmatrix} = \begin{bmatrix} \ln\big(\mathbf{T}(t)\mathbf{T}_k^{-1}\big)^{\vee} \\ \boldsymbol{\mathcal{J}}_{t,k}\boldsymbol{\varpi}(t) \\ -\frac{1}{2}(\boldsymbol{\mathcal{J}}_{t,k}^{-1}\boldsymbol{\varpi}(t))^{\curlywedge}\boldsymbol{\varpi}(t) + \boldsymbol{\mathcal{J}}_{t,k}^{-1}\dot{\boldsymbol{\varpi}}(t) \end{bmatrix}, \tag{5.48}$$

moreover, by definition, we have

$$\boldsymbol{\gamma}_k(t_{k+1}) = \begin{bmatrix} \ln(\mathbf{T}_{k+1,k})^{\vee} \\ \boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\boldsymbol{\varpi}_{k+1} \\ -\frac{1}{2}(\boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\boldsymbol{\varpi}_{k+1})^{\curlywedge}\boldsymbol{\varpi}_{k+1} + \boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\dot{\boldsymbol{\varpi}}_{k+1} \end{bmatrix}, \quad \boldsymbol{\gamma}_k(t_k) = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\varpi}_k \\ \dot{\boldsymbol{\varpi}}_k \end{bmatrix}, \tag{5.49}$$

where we made use of the identity $\mathbf{x}^{\curlywedge}\mathbf{x} = \mathbf{0}$. Using (5.31) and the relationship between *local* and *global* state variables, the prior error term is

$$\mathbf{e}_k = \begin{bmatrix} \ln(\mathbf{T}_{k+1,k})^{\vee} - (t_{k+1} - t_k)\boldsymbol{\varpi}_k - \frac{1}{2}(t_{k+1} - t_k)^2\dot{\boldsymbol{\varpi}}_k \\ \boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\boldsymbol{\varpi}_{k+1} - \boldsymbol{\varpi}_k - (t_{k+1} - t_k)\dot{\boldsymbol{\varpi}}_k \\ -\frac{1}{2}(\boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\boldsymbol{\varpi}_{k+1})^{\curlywedge}\boldsymbol{\varpi}_{k+1} + \boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\dot{\boldsymbol{\varpi}}_{k+1} - \dot{\boldsymbol{\varpi}}_k \end{bmatrix}. \tag{5.50}$$

Suppose we assume the trajectory has zero acceleration (which is assumed by a prior mean that is constant-velocity), $\dot{\boldsymbol{\varpi}}_i = \dot{\boldsymbol{\varpi}}_{i+1} = \mathbf{0}$, and also make the assumption that $\boldsymbol{\mathcal{J}}_{i+1,i}^{-1}\boldsymbol{\varpi}_{i+1} \approx \boldsymbol{\varpi}_{i+1}$. In this case, the last component in (5.50) becomes zero, and the first two components become identical to the WNOA prior as in (2.14); we have essentially recovered the prior error equation for the WNOA prior.

5.3.5 Motion Prior Jacobian

Let $\mathbf{e}_k \in \mathbb{R}^{18} = \begin{bmatrix} \mathbf{e}_{k_1}^T & \mathbf{e}_{k_2}^T & \mathbf{e}_{k_3}^T \end{bmatrix}^T$. Using results from [4], it is straightforward to compute the Jacobian for $\mathbf{e}_{k_1}$ and $\mathbf{e}_{k_2}$, it then remains to compute the Jacobian for $\mathbf{e}_{k_3}$. Define a dummy variable $\mathbf{e}' = -\frac{1}{2}(\boldsymbol{\mathcal{J}}_{b,a}^{-1}\boldsymbol{\varpi}_b)^\wedge \boldsymbol{\varpi}_b$. Linearizing about an operating point, we have

$$\mathbf{e}' \approx -\frac{1}{2}\left(\boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_{\text{op},b,a} + \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\xi}_{b,a})^{-1}(\boldsymbol{\varpi}_{\text{op},b} + \delta\boldsymbol{\varpi}_b)\right)^\wedge (\boldsymbol{\varpi}_{\text{op},b} + \delta\boldsymbol{\varpi}_b), \tag{5.51}$$

here we made the approximation that $\ln(\mathbf{T}_{b,a})^\vee \approx \ln(\mathbf{T}_{\text{op},b,a})^\vee + \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\xi}_{\text{op},b,a}$, as usual. Results in [4] show that

$$\boldsymbol{\mathcal{J}}(\boldsymbol{\xi}_{\text{op},b,a} + \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\xi}_{b,a})^{-1}(\boldsymbol{\varpi}_{\text{op},b} + \delta\boldsymbol{\varpi}_b) \approx \boldsymbol{\mathcal{J}}_{b,a}^{-1}\boldsymbol{\varpi}_{\text{op},b} + \frac{1}{2}\boldsymbol{\varpi}_{\text{op},b}^\wedge \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\xi}_{b,a} + \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\varpi}_b. \tag{5.52}$$

Therefore, we have

$$\begin{aligned}
\mathbf{e}' &\approx -\frac{1}{2}(\boldsymbol{\mathcal{J}}_{b,a}^{-1}\boldsymbol{\varpi}_{\text{op},b} + \frac{1}{2}\boldsymbol{\varpi}_{\text{op},b}^\wedge \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\xi}_{b,a} + \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\varpi}_b)^\wedge (\boldsymbol{\varpi}_{\text{op},b} + \delta\boldsymbol{\varpi}_b) \\
&= -\frac{1}{2}\left((\boldsymbol{\mathcal{J}}_{b,a}\boldsymbol{\varpi}_{\text{op},b})^\wedge + \frac{1}{2}(\boldsymbol{\varpi}_{\text{op},b}^\wedge \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\xi}_{b,a})^\wedge + (\boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\varpi}_b)^\wedge\right)(\boldsymbol{\varpi}_{\text{op},b} + \delta\boldsymbol{\varpi}_b) \\
&= -\frac{1}{2}\left((\boldsymbol{\mathcal{J}}_{b,a}\boldsymbol{\varpi}_{\text{op},b})^\wedge + \frac{1}{2}\boldsymbol{\varpi}_{\text{op},b}^\wedge(\boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\xi}_{b,a})^\wedge - \frac{1}{2}(\boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\xi}_{b,a})^\wedge \boldsymbol{\varpi}_{\text{op},b}^\wedge + (\boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\varpi}_b)^\wedge\right) \\
&\quad (\boldsymbol{\varpi}_{\text{op},b} + \delta\boldsymbol{\varpi}_b) \\
&\approx -\frac{1}{2}\left((\boldsymbol{\mathcal{J}}_{b,a}^{-1}\boldsymbol{\varpi}_{\text{op},b})^\wedge \boldsymbol{\varpi}_{\text{op},b} + (\boldsymbol{\mathcal{J}}_{b,a}^{-1}\boldsymbol{\varpi}_{\text{op},b})^\wedge \delta\boldsymbol{\varpi}_b - \boldsymbol{\varpi}_{\text{op},b}^\wedge \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\varpi}_b - \frac{1}{2}\boldsymbol{\varpi}_{\text{op},b}^\wedge \boldsymbol{\varpi}_{\text{op},b}^\wedge \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\xi}_{b,a}\right),
\end{aligned} \tag{5.53}$$

where we rearranged using $SE(3)$ identities and dropped higher-order terms. Furthermore, by examining (5.52), it is straightforward to see that

$$\boldsymbol{\mathcal{J}}_{b,a}^{-1}\dot{\boldsymbol{\varpi}}_b \approx \boldsymbol{\mathcal{J}}_{b,a}^{-1}\dot{\boldsymbol{\varpi}}_{\text{op},b} + \frac{1}{2}\dot{\boldsymbol{\varpi}}_{\text{op},b}^\wedge \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\boldsymbol{\xi}_{b,a} + \boldsymbol{\mathcal{J}}_{b,a}^{-1}\delta\dot{\boldsymbol{\varpi}}_b. \tag{5.54}$$

Therefore, we have the following derivatives:

$$\frac{\partial\, \mathbf{e}_{k_3}}{\partial\, \delta\boldsymbol{\xi}_{k+1,k}} = \frac{1}{4}\boldsymbol{\varpi}_{\text{op},k+1}^\wedge \boldsymbol{\varpi}_{\text{op},k+1}^\wedge \boldsymbol{\mathcal{J}}_{k+1,k}^{-1} + \frac{1}{2}\dot{\boldsymbol{\varpi}}_{\text{op},k+1}^\wedge \boldsymbol{\mathcal{J}}_{k+1,k}^{-1}, \tag{5.55}$$

$$\frac{\partial\, \mathbf{e}_{k_3}}{\partial\, \delta\boldsymbol{\varpi}_{k+1}} = -\frac{1}{2}(\boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\boldsymbol{\varpi}_{\text{op},k+1})^\wedge + \frac{1}{2}\boldsymbol{\varpi}_{\text{op},k+1}^\wedge \boldsymbol{\mathcal{J}}_{k+1,k}^{-1}, \tag{5.56}$$

$$\frac{\partial\, \mathbf{e}_{k_3}}{\partial\, \delta\dot{\boldsymbol{\varpi}}_k} = -\mathbf{1}, \tag{5.57}$$

$$\frac{\partial\, \mathbf{e}_{k_3}}{\partial\, \delta\dot{\boldsymbol{\varpi}}_{k+1}} = \boldsymbol{\mathcal{J}}^{-1}_{k+1,k}. \tag{5.58}$$

Finally, we make note of the approximation $\delta\boldsymbol{\xi}_{b,a} \approx \delta\boldsymbol{\xi}_b - \boldsymbol{\mathcal{T}}_{b,a}\delta\boldsymbol{\xi}_a$. Therefore we have $\mathbf{E}_k = \begin{bmatrix}\mathbf{E}_{k_1}^T & \mathbf{E}_{k_2}^T & \mathbf{E}_{k_3}^T\end{bmatrix}^T$, with

$$\begin{aligned}\mathbf{E}_{k_1} &= \begin{bmatrix}\frac{\partial\, \mathbf{e}_{k_1}}{\partial\, \delta\boldsymbol{\xi}_k} & \frac{\partial\, \mathbf{e}_{k_1}}{\partial\, \delta\boldsymbol{\varpi}_k} & \frac{\partial\, \mathbf{e}_{k_1}}{\partial\, \delta\dot{\boldsymbol{\varpi}}_k} & \frac{\partial\, \mathbf{e}_{k_1}}{\partial\, \delta\boldsymbol{\xi}_{k+1}} & \frac{\partial\, \mathbf{e}_{k_1}}{\partial\, \delta\boldsymbol{\varpi}_{k+1}} & \frac{\partial\, \mathbf{e}_{k_1}}{\partial\, \delta\dot{\boldsymbol{\varpi}}_{k+1}}\end{bmatrix} \\ &= \begin{bmatrix}-\boldsymbol{\mathcal{J}}^{-1}_{k+1,k}\boldsymbol{\mathcal{T}}_{k+1,k} & -(t_{k+1}-t_k)\mathbf{1} & -\frac{1}{2}(t_{k+1}-t_k)^2\mathbf{1} & \boldsymbol{\mathcal{J}}^{-1}_{k+1,k} & \mathbf{0} & \mathbf{0}\end{bmatrix},\end{aligned} \tag{5.59}$$

$$\mathbf{E}_{k_2} = \begin{bmatrix}-\frac{1}{2}\boldsymbol{\varpi}^{\curlywedge}_{\mathrm{op},k+1}\boldsymbol{\mathcal{J}}^{-1}_{k+1,k}\boldsymbol{\mathcal{T}}_{k+1,k} & -\mathbf{1} & -(t_{k-1}-t_k)\mathbf{1} & \frac{1}{2}\boldsymbol{\varpi}^{\curlywedge}_{\mathrm{op},k+1}\boldsymbol{\mathcal{J}}^{-1}_{k+1,k} & \boldsymbol{\mathcal{J}}^{-1}_{k+1,k} & \mathbf{0}\end{bmatrix}, \tag{5.60}$$

$$\mathbf{E}_{k_3} = \begin{bmatrix}\left(-\frac{1}{4}\boldsymbol{\varpi}^{\curlywedge}_{\mathrm{op},k+1}\boldsymbol{\varpi}^{\curlywedge}_{\mathrm{op},k+1}\boldsymbol{\mathcal{J}}^{-1}_{k+1,k}\boldsymbol{\mathcal{T}}_{k+1,k} - \frac{1}{2}\dot{\boldsymbol{\varpi}}^{\curlywedge}_{\mathrm{op},k+1}\boldsymbol{\mathcal{J}}^{-1}_{k+1,k}\boldsymbol{\mathcal{T}}_{k+1,k}\right)^T \\ \mathbf{0} \\ -\mathbf{1} \\ \left(\frac{1}{4}\boldsymbol{\varpi}^{\curlywedge}_{\mathrm{op},k+1}\boldsymbol{\varpi}^{\curlywedge}_{\mathrm{op},k+1}\boldsymbol{\mathcal{J}}^{-1}_{k+1,k} + \frac{1}{2}\dot{\boldsymbol{\varpi}}^{\curlywedge}_{\mathrm{op},k+1}\boldsymbol{\mathcal{J}}^{-1}_{k+1,k}\right)^T \\ \left(-\frac{1}{2}(\boldsymbol{\mathcal{J}}^{-1}_{k+1,k}\boldsymbol{\varpi}_{\mathrm{op},k+1})^{\curlywedge} + \frac{1}{2}\boldsymbol{\varpi}^{\curlywedge}_{\mathrm{op},k+1}\boldsymbol{\mathcal{J}}^{-1}_{k+1,k}\right)^T \\ \left(\boldsymbol{\mathcal{J}}^{-1}_{k+1,k}\right)^T\end{bmatrix}^T. \tag{5.61}$$

5.3.6 Querying the Trajectory Mean

We start from the same interpolation equation using *local* state variables (5.33). For the WNOJ prior, the interpolation coefficients $\boldsymbol{\Lambda}(\tau)$ and $\boldsymbol{\Omega}(\tau)$ can be computed from (5.34), using $\boldsymbol{\Phi} \in \mathbb{R}^{18\times 18}$ and $\mathbf{Q}_i \in \mathbb{R}^{18\times 18}$ from (5.39) and (5.40).

Substituting with *global* state variables for the WNOJ prior using (5.49), the pose interpolation equation is

$$\begin{aligned}\mathbf{T}_\tau = \exp\Big(&(\boldsymbol{\Lambda}_{12}(\tau)\boldsymbol{\varpi}_i + \boldsymbol{\Lambda}_{13}(\tau)\dot{\boldsymbol{\varpi}}_i + \boldsymbol{\Omega}_{11}(\tau)\ln(\mathbf{T}_{i+1,i})^{\vee} + \boldsymbol{\Omega}_{12}(\tau)\boldsymbol{\mathcal{J}}^{-1}_{i+1,i}\boldsymbol{\varpi}_{i+1} \\ &+ \boldsymbol{\Omega}_{13}(\tau)(-\frac{1}{2}(\boldsymbol{\mathcal{J}}^{-1}_{i+1,i}\boldsymbol{\varpi}_{i+1})^{\curlywedge}\boldsymbol{\varpi}_{i+1} + \boldsymbol{\mathcal{J}}^{-1}_{i+1,i}\dot{\boldsymbol{\varpi}}_{i+1}))^{\wedge}\Big)\mathbf{T}_i,\end{aligned} \tag{5.62}$$

where $t_i < \tau < t_{i+1}$, and

$$\boldsymbol{\Omega}(\tau) = \begin{bmatrix}\boldsymbol{\Omega}_{11}(\tau) & \boldsymbol{\Omega}_{12}(\tau) & \boldsymbol{\Omega}_{13}(\tau) \\ \boldsymbol{\Omega}_{21}(\tau) & \boldsymbol{\Omega}_{22}(\tau) & \boldsymbol{\Omega}_{23}(\tau) \\ \boldsymbol{\Omega}_{31}(\tau) & \boldsymbol{\Omega}_{32}(\tau) & \boldsymbol{\Omega}_{33}(\tau)\end{bmatrix}, \quad \boldsymbol{\Lambda}(\tau) = \begin{bmatrix}\boldsymbol{\Lambda}_{11}(\tau) & \boldsymbol{\Lambda}_{12}(\tau) & \boldsymbol{\Lambda}_{13}(\tau) \\ \boldsymbol{\Lambda}_{21}(\tau) & \boldsymbol{\Lambda}_{22}(\tau) & \boldsymbol{\Lambda}_{23}(\tau) \\ \boldsymbol{\Lambda}_{31}(\tau) & \boldsymbol{\Lambda}_{32}(\tau) & \boldsymbol{\Lambda}_{33}(\tau)\end{bmatrix}. \tag{5.63}$$

Again, if we assume that $\dot{\boldsymbol{\varpi}}_i = \dot{\boldsymbol{\varpi}}_{i+1} = \mathbf{0}$, and $\boldsymbol{\mathcal{J}}^{-1}_{i+1,i}\boldsymbol{\varpi}_{i+1} \approx \boldsymbol{\varpi}_{i+1}$, the terms with coefficients $\boldsymbol{\Lambda}_{13}$ and $\boldsymbol{\Omega}_{13}$ become zeros. Similar to the case with the prior error term, we

can essentially recover the pose interpolation equation for the WNOA prior as in (5.35).

Defining $\Delta t = t_{k+1} - t_k$, $\Delta\tau = \tau - t_k$ and $\Delta\tau' = t_{k+1} - \tau$. Without needing to compute the full coefficient matrices $\boldsymbol{\Omega}(\tau)$ and $\boldsymbol{\Lambda}(\tau)$, the smaller matrices $\boldsymbol{\Omega}_{ij}(\tau)$ and $\boldsymbol{\Lambda}_{ij}(\tau)$ in (5.62) can be computed efficiently as

$$\boldsymbol{\Lambda}_{12}(\tau) = \frac{\Delta\tau\Delta\tau'^3}{\Delta t^4}(t_{k+1} - 4t_k + 3\tau)\mathbf{1}, \tag{5.64}$$

$$\boldsymbol{\Lambda}_{13}(\tau) = \frac{\Delta\tau^2\Delta\tau'^3}{2\Delta t^3}\mathbf{1}, \tag{5.65}$$

$$\boldsymbol{\Omega}_{11}(\tau) = \frac{\Delta\tau^3}{\Delta t^5}(t_k^2 - 5t_k t_{k+1} + 3t_k\tau + 10t_{k+1}^2 - 15t_{k+1}\tau + 6\tau^2)\mathbf{1}, \tag{5.66}$$

$$\boldsymbol{\Omega}_{12}(\tau) = \frac{\Delta\tau^3\Delta\tau'}{\Delta t^4}(t_k - 4t_{k+1} + 3\tau)\mathbf{1}, \tag{5.67}$$

$$\boldsymbol{\Omega}_{13}(\tau) = \frac{\Delta\tau^3\Delta\tau'^2}{2\Delta t^3}\mathbf{1}. \tag{5.68}$$

Finally, we need the Jacobian of the interpolated pose $\mathbf{T}_\tau$ with respect to the perturbation to each state variable, in order to update the state variables appropriately in Gauss-Newton. Since we have $\mathbf{T}_\tau = \mathbf{T}_{\tau,k}\mathbf{T}_k$, using the product rule and chain rule, we can write

$$\begin{aligned}\frac{\partial\,\boldsymbol{\xi}_\tau}{\partial\,\delta\boldsymbol{\xi}_k} &\approx \frac{\partial\,\boldsymbol{\xi}_{\tau,k}}{\partial\,\delta\boldsymbol{\xi}_k} + \boldsymbol{\mathcal{T}}_{\tau,k}\frac{\partial\,\boldsymbol{\xi}_k}{\partial\,\delta\boldsymbol{\xi}_k} \\ &= \frac{\partial\,\boldsymbol{\xi}_{\tau,k}}{\partial\,\delta\boldsymbol{\xi}_k} + \boldsymbol{\mathcal{T}}_{\tau,k},\end{aligned} \tag{5.69}$$

$$\begin{aligned}\frac{\partial\,\boldsymbol{\xi}_\tau}{\partial\,\delta\boldsymbol{\xi}_{k+1}} &\approx \frac{\partial\,\boldsymbol{\xi}_{\tau,k}}{\partial\,\delta\boldsymbol{\xi}_{k+1}} + \boldsymbol{\mathcal{T}}_{\tau,k}\frac{\partial\,\boldsymbol{\xi}_k}{\partial\,\delta\boldsymbol{\xi}_{k+1}} \\ &= \frac{\partial\,\boldsymbol{\xi}_{\tau,k}}{\partial\,\delta\boldsymbol{\xi}_{k+1}}.\end{aligned} \tag{5.70}$$

Making use of results from Section 5.3.5, the interpolation Jacobians can be calculated as

$$\frac{\partial\,\boldsymbol{\xi}_\tau}{\partial\,\delta\boldsymbol{\xi}_k} = -\frac{\partial\,\boldsymbol{\xi}_{\tau,k}}{\partial\,\delta\boldsymbol{\xi}_{k+1,k}}\boldsymbol{\mathcal{T}}_{k+1,k} + \boldsymbol{\mathcal{T}}_{\tau,k}, \tag{5.71}$$

$$\frac{\partial\,\boldsymbol{\xi}_\tau}{\partial\,\delta\boldsymbol{\varpi}_k} = \boldsymbol{\Lambda}_{12}(\tau)\boldsymbol{\mathcal{J}}_{\tau,k}, \tag{5.72}$$

$$\frac{\partial\,\boldsymbol{\xi}_\tau}{\partial\,\delta\dot{\boldsymbol{\varpi}}_k} = \boldsymbol{\Lambda}_{13}(\tau)\boldsymbol{\mathcal{J}}_{\tau,k}, \tag{5.73}$$

$$\frac{\partial\,\boldsymbol{\xi}_\tau}{\partial\,\delta\boldsymbol{\xi}_{k+1}} = \frac{\partial\,\boldsymbol{\xi}_{\tau,k}}{\partial\,\delta\boldsymbol{\xi}_{k+1,k}}, \tag{5.74}$$

$$\frac{\partial \boldsymbol{\xi}_\tau}{\partial \delta \boldsymbol{\varpi}_{k+1}} = \boldsymbol{\Omega}_{12}(\tau)\boldsymbol{\mathcal{J}}_{\tau,k}\boldsymbol{\mathcal{J}}_{k+1,k}^{-1} - \frac{1}{2}\boldsymbol{\Omega}_{13}(\tau)\boldsymbol{\mathcal{J}}_{\tau,k}\left((\boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\boldsymbol{\varpi}_{\mathrm{op},k+1})^\curlywedge - \boldsymbol{\varpi}_{\mathrm{op},k+1}^\curlywedge\boldsymbol{\mathcal{J}}_{k+1,k}^{-1}\right), \tag{5.75}$$

$$\frac{\partial \boldsymbol{\xi}_\tau}{\partial \delta \dot{\boldsymbol{\varpi}}_{k+1}} = \boldsymbol{\Omega}_{13}(\tau)\boldsymbol{\mathcal{J}}_{\tau,k}\boldsymbol{\mathcal{J}}_{k+1,k}^{-1}, \tag{5.76}$$

where the intermediate partial derivative $\frac{\partial \boldsymbol{\xi}_{\tau,k}}{\partial \delta \boldsymbol{\xi}_{k+1,k}}$ can be evaluated as

$$\begin{aligned}\frac{\partial \boldsymbol{\xi}_{\tau,k}}{\partial \delta \boldsymbol{\xi}_{k+1,k}} = {} & \boldsymbol{\Omega}_{11}(\tau)\boldsymbol{\mathcal{J}}_{\tau,k}\boldsymbol{\mathcal{J}}_{k+1,k}^{-1} + \frac{1}{2}\boldsymbol{\Omega}_{12}\boldsymbol{\mathcal{J}}_{\tau,k}\boldsymbol{\varpi}_{\mathrm{op},k+1}^\curlywedge\boldsymbol{\mathcal{J}}_{k+1,k}^{-1} \\ & + \frac{1}{4}\boldsymbol{\Omega}_{13}(\tau)\boldsymbol{\mathcal{J}}_{\tau,k}\boldsymbol{\varpi}_{\mathrm{op},k+1}^\curlywedge\boldsymbol{\varpi}_{\mathrm{op},k+1}^\curlywedge\boldsymbol{\mathcal{J}}_{k+1,k}^{-1} + \frac{1}{2}\boldsymbol{\Omega}_{13}(\tau)\boldsymbol{\mathcal{J}}_{\tau,k}\dot{\boldsymbol{\varpi}}_{\mathrm{op},k+1}^\curlywedge\boldsymbol{\mathcal{J}}_{k+1,k}^{-1}.\end{aligned} \tag{5.77}$$

Although tedious, Equations (5.71) to (5.77) are exact to first order. Where appropriate, further approximations may be made to speed up the computation.

5.4 Comparison Between WNOJ Prior and WNOA Prior

5.4.1 Simulation Set-up

To compare between the WNOJ prior and the WNOA prior, we created a simulation environment resembling a simplified problem of a vehicle driving in city blocks, while estimating its pose using point measurements.

The simulation world is a 1000m $\times$ 1000m space, made up of 10×10 blocks, where each block is a square with 100m on each side. At t_0, the vehicle spawns at a random intersection, facing a random direction among East, West, North, and South. The vehicle then travels a random path constrained within the 10 block-by-10 block world. Examples of two trajectories generated by the simulation is shown in Figure 5.10.

There are a number of stops randomly located at certain intersections, marked by the red diamonds in Figure 5.10. The trajectory of the vehicle is designed to behave with the following characteristics:

- The vehicle will either drive forward, turn left, or turn right at each intersection
- The vehicle has a higher probability of choosing to drive forward than taking a turn
- When initiating a turn, the vehicle will first slow down to come to a full stop. It will then accelerate into a turn, followed by decelerating out of the turn

- When the vehicle drives forward through an intersection with a stop, it will slow down to come to a full stop, and then speed up to its cruise speed
- When the vehicle drives forward through an intersection without a stop, it will drive through with a constant velocity
- Other than needing to speed up or slow down because of taking a turn, or the presence of a stop, the vehicle drives forward with a constant velocity

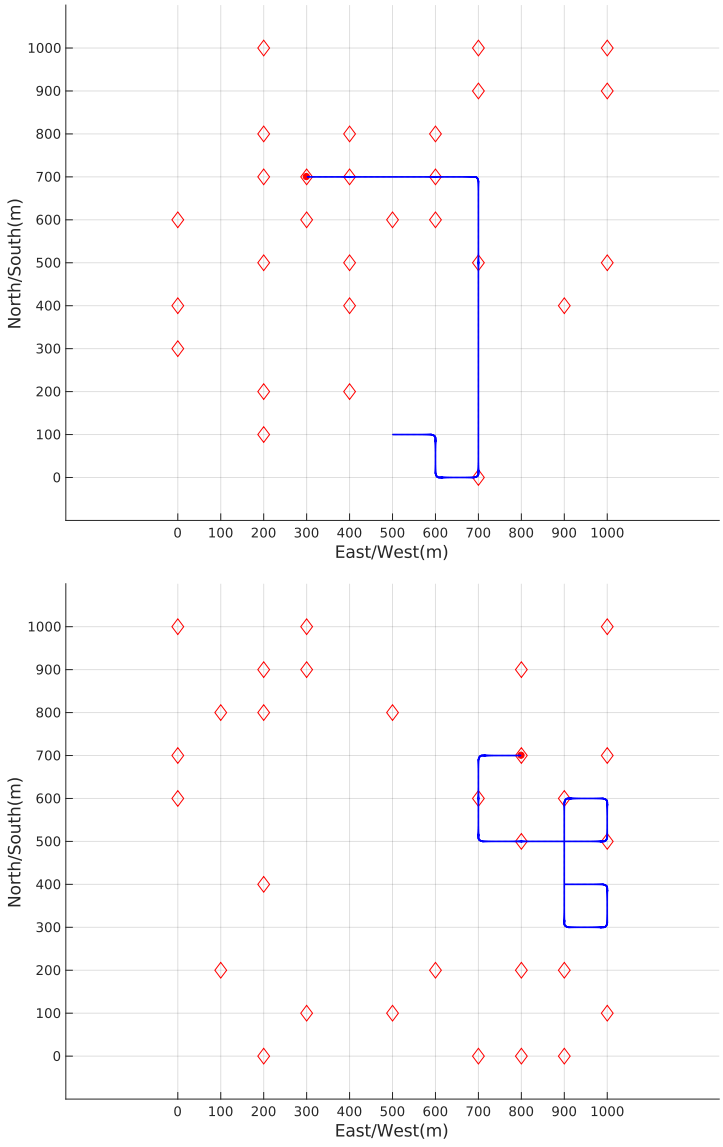

Figure 5.10: Simulated trajectories of driving in city blocks. The starting location is marked by a red dot.

The simulated trajectory is therefore either piece-wise constant-acceleration (slowing down or speeding up while driving forward, or when taking a turn), or constant-velocity

at cruise speed while driving forward. Moreover, the cruise speed is chosen to be 20m/s, while the turning radius is 10m.

There are rectangular structures within each block resembling buildings. The on-board sensor of the vehicle measures 3D points on the surfaces of the buildings, which can be used to estimate the trajectory of the vehicle. Examples of point measurements are shown in Figure 5.11. The small blue points are all points available on the surface of buildings in the 4 nearest blocks, while the red points are the points measured after taking into account of sensor range and occlusion. Each building has a certain height, while Figure 5.11 shows a top-down view, every blue or red point in Figure 5.11 is in fact several points with the same x and y coordinates but different z coordinates stacked together.

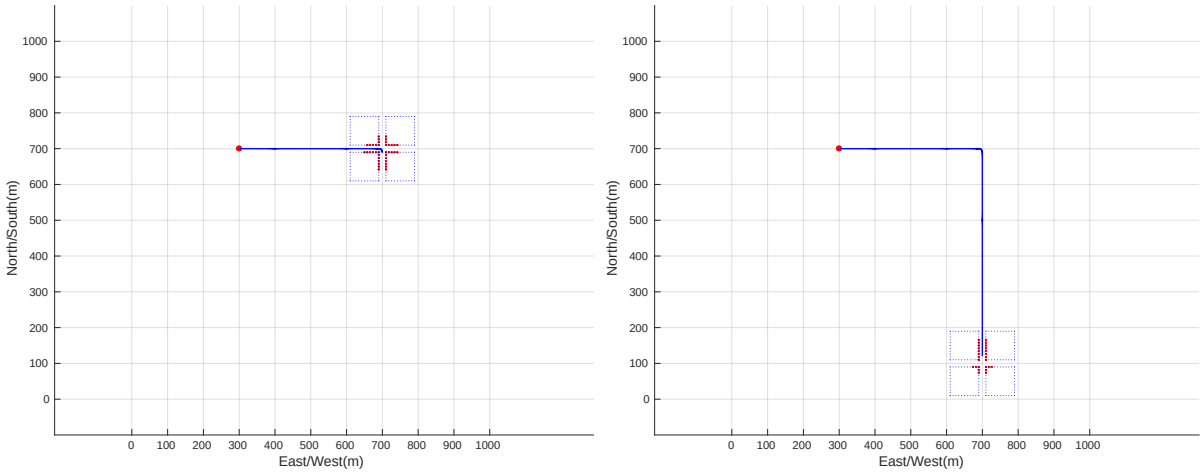

Figure 5.11: Points available on nearby buildings are shown as small blue points, while the points measured are shown as red points. Every point shown in plot are a number of points with the same x and y coordinates but different z coordinates stacked together.

The sensor measures points at intervals of 0.1s, to resemble the Velodyne sensor. The knots in the continuous-time trajectory are also spaced 0.1s apart. We have point measurements every 0.3s. Furthermore, we simulated both motion-distorted measurements, and non-distorted measurements where all points are collected at the same timestep.

The trajectories are solved using point measurements along with either a WNOA prior or a WNOJ prior. The results are evaluated against ground truth. Given sample trajectories from training data, we can solve for the best hyperparameter $\mathbf{Q}_c$ by maximizing the log-likelihood as shown in [4]. We generate a large number of trajectory sequences as the training set, which we use to tune $\mathbf{Q}_c$ for both the WNOA prior and the WNOJ prior. The tuned hyperparameter is then used in estimation for trajectory sequences in

the test set.

For simplicity, we simulate the sensor to directly measure the Cartesian coordinates of points, rather than return a range and azimuth as real-world laser sensors. We keep the point measurements noise-free, and we assume we have perfect correspondences, therefore we do not need to perform nearest-neighbor search to find point matches. Moreover, we use a sliding-window style estimation, where the window size is chosen to be two.

5.4.2 Results

As discussed previously, we evaluated using both motion-distorted measurements and non-distorted measurements. For each type of measurements we estimate the trajectory using both the WNOA motion prior and the WNOJ motion prior.

For distorted measurements, we need to interpolate the pose associated with each point measurement given its associated time. When using a WNOJ prior, Equation (5.62) is the pose interpolation equation used. When using a WNOA prior, we perform pose interpolation using (5.35).

For non-distorted measurements, we do not need pose interpolation since measurements are collected simultaneously at a knot time, but we still use prior error terms. In this case, the motion prior error terms are used to constrain the trajectory at timesteps without measurements, and to act as a trajectory smoother.

As a comparison baseline for distorted measurements, we also attempt to solve the trajectory using only measurement terms (ie. no prior). As such, we are essentially doing discrete-time estimation, where we make the ad-hoc assumption that all points in a scan are collected at the same timestep.

Ten test trajectory sequences are evaluated, where the total distance travelled in each sequence is more than 1.3km. A comparison table of the odometry errors for the different scenarios is shown in Table 5.1. In can be seen that in our simulated problem, the WNOJ prior always outperforms the WNOA prior, as it offers a more appropriate representation of the underlying trajectory. Furthermore, the error reduction from switching from WNOA prior to WNOJ prior is more significant when we have distorted measurements.

Examples of the estimated trajectory compared against ground truth for distorted measurements are shown in Figure 5.12. It can be seen from Figure 5.12 that the estimated trajectory using a WNOJ prior is visibly more aligned to the ground truth.

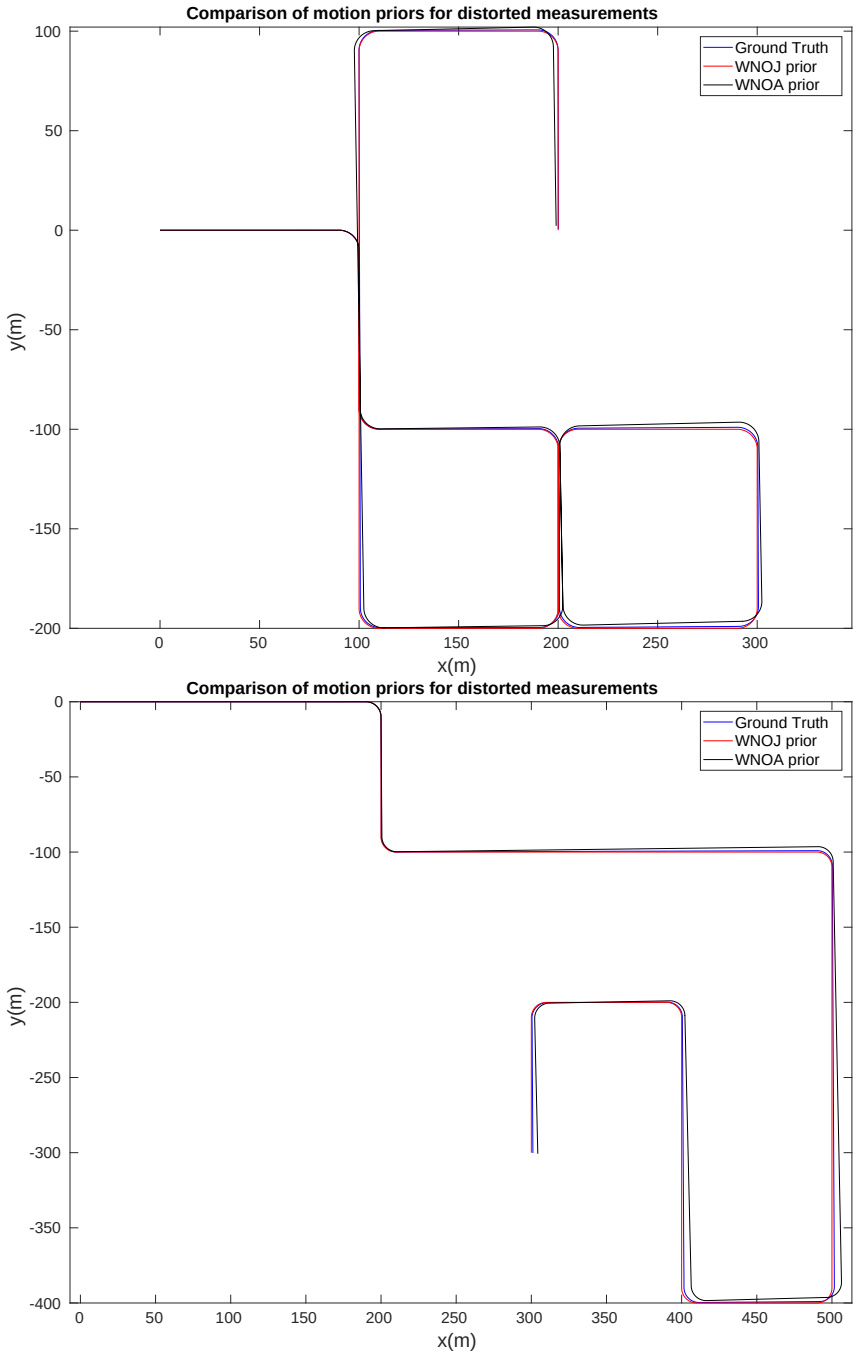

Figure 5.12: Simulated trajectories of driving in city blocks. The starting location is marked by a red dot.

Table 5.1: Odometry errors for measurements with and without motion distortion, solved using WNOJ prior and WNOA prior

sequence no.	WNOJ no distortion (%)	WNOA no distortion (%)	WNOJ distorted (%)	WNOA distorted (%)	discrete-time (%)
1	**0.0151**	0.1068	**0.0935**	0.2432	0.6686
2	**0.0083**	0.0744	**0.0668**	0.2367	0.5024
3	**0.0098**	0.0998	**0.0766**	0.2639	0.6583
4	**0.0085**	0.0916	**0.0769**	0.2461	0.5833
5	**0.0089**	0.0836	**0.0852**	0.2419	0.5790
6	**0.0082**	0.0733	**0.0684**	0.2133	0.4968
7	**0.0118**	0.1036	**0.0976**	0.3343	0.7192
8	**0.0155**	0.1068	**0.0677**	0.1974	0.7328
9	**0.0112**	0.0922	**0.0712**	0.1912	0.6211
10	**0.0124**	0.1103	**0.0918**	0.3180	0.7293

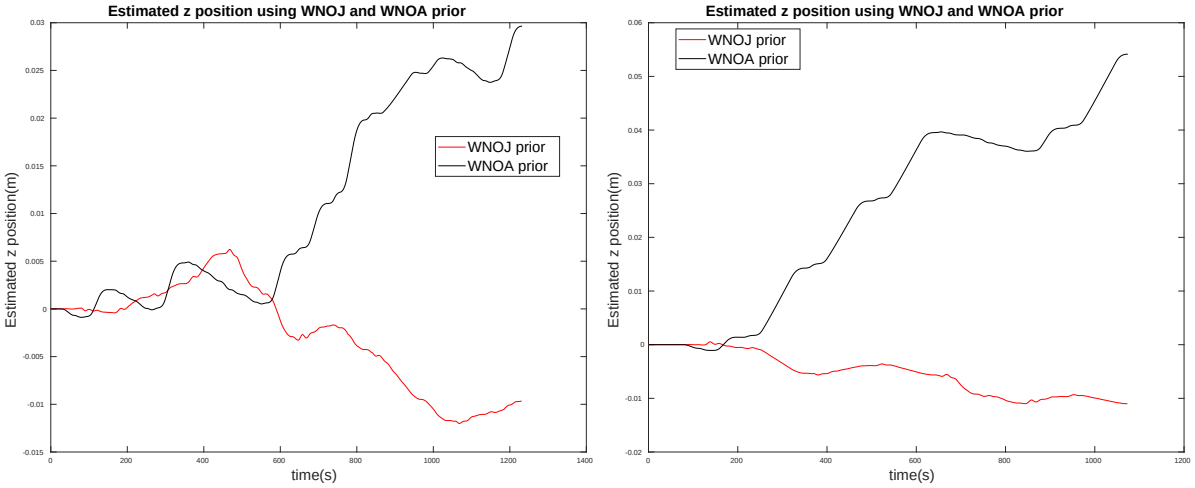

Figure 5.13: Estimated z position of the vehicle when using WNOA prior (black) and WNOJ prior (red), with sequence 1 on the left and sequence 8 on the right. The WNOJ prior results in noticeably smaller z bias than the WNOA prior.

5.4.3 Estimator Bias

The simulation is generated where the z position of the vehicle is constrained to be zero. However, due to bias in the estimator, the estimated z position of the vehicle will not necessarily be zero, as discussed in Section 5.2. The simulated measurements are noise-free, and we have perfect point correspondences, therefore the only source of estimator bias comes from the motion prior not able to describe the underlying trajectory. Examples of the estimated z position of the vehicle are shown in Figure 5.13.

It can be seen that there is a greater bias when the estimator uses a WNOA prior, than when it uses a WNOJ prior. This is due to the WNOJ prior being more appropriate for representing the underlying trajectories, which are piece-wise constant-acceleration or constant-velocity. There is still bias, though, even when the estimator uses a WNOJ prior, as shown in Figure 5.13. This is due to the simulated trajectory being only piece-wise constant-acceleration, but not constant-acceleration throughout. In other words, there are places where the acceleration undergoes a non-smooth change, such as the instant where the vehicle begins to speed up.

A simple experiment has been done where we do not use a prior at these places where there is a sudden change in acceleration, while the prior is used everywhere else along the trajectory. In this case, the estimated z position of the vehicle is zero along the entire trajectory when we use a WNOJ prior, since the motion prior now matches with the underlying trajectory at everywhere it is being used.

5.5 Summary

In this chapter we explored the theoretical aspects of $SE(3)$ estimation, and how we can improve the estimator from a mathematical and algorithmic perspective. We showed analytically that when a WNOA prior is used while the underlying trajectory has acceleration, the interaction between the prior term and the measurement terms will induce a bias in the estimation, even towards degrees of freedom without any motion.

The highlight of this chapter is in Section 5.3, where we derive a WNOJ motion prior. Specifically, our estimation pipeline now contains body-centric acceleration, $\dot{\boldsymbol{\varpi}}$, as part of the state vector. Theoretically, continuous-time estimation using a WNOJ prior is more suitable for scenarios where the sensor undergoes frequent changes in velocity, such as a car in traffic. For these scenarios, we expect a smaller estimator bias when we use a WNOJ prior, than using a WNOA prior.

We then proceeded to develop a simulation environment, which simulates simple

problems of estimating motion while driving in city-blocks. The simulated trajectories are piece-wise constant-acceleration or constant-velocity. After performing estimation using both types of motion priors, we have shown that the WNOJ prior is superior as it produces greater accuracy and smaller bias.

Chapter 6

Conclusion and Future Work

6.1 Summary of Contributions

The first contribution of this thesis is the development of a real-time, accurate lidar odometry pipeline, which utilizes previous work on simultaneous trajectory estimation and mapping (STEAM) [3]. Our lidar odometry algorithm is capable of handling motion-distorted point-cloud data in a principled way, and has been tested to produce reasonably accurate results on large amounts of lidar datasets. Specifically, at the time of submission, our lidar odometry algorithm ranked #3 on the KITTI odometry benchmark among lidar-only methods. It remains the most accurate lidar-only method that is strictly odometric (ie., does not use mapping) on the KITTI leader-board at the writing time of this thesis. Our work on lidar odometry contributed towards the publication in [27].

Evaluation on datasets shows that, however, our pipeline exhibits bias in certain degrees of freedom, causing the odometry to drift over time. With this in mind, the rest of this thesis explores ideas for reducing drift in lidar-based motion estimation. In Chapter 4 we showed two methods for learning a bias and applying the learned bias as a correction to odometry. The first method uses GP regression, and requires choosing hand-picked features from the geometry of point-clouds as inputs to the model. The second method uses deep learning, and takes in 2D range images converted from 3D point-clouds which are passed through a CNN. Both methods resulted in overall improvements to the odometry after applying the learned bias correction. This work has led to the publication in [43].

Methods shown in Chapter 4 need prior training data, and most importantly, require the training data to be sufficiently representative of the test data. In Chapter 5 we focused on more generic methods for reducing bias from a mathematical and algorithmic perspective. We performed theoretical analysis of $SE(3)$ estimation, and showed a source

of bias results from the motion prior not being able to describe the underlying trajectory. The majority of Chapter 5 centres around the derivation and evaluation of a novel WNOJ motion model, as an alternative to the existing WNOA motion model used in STEAM. We argue that the WNOJ model is more appropriate for trajectories containing rapid changes in velocity. Our argument is evaluated in a simulated environment for simple estimation problems in urban driving scenarios.

6.2 Future Work

Our lidar odometry algorithm currently uses CPU only. A potential extension would be to enable GPU implementation, which may allow us to afford more refined solutions while maintaining real-time performances.

In Chapter 4 we showed that for our method of learning a bias correction, a model trained on the KITTI training data did not improve the odometry on the U of T dataset, due to the differences in measurements between the two datasets. An extension would be to train a model across a large number of different datasets, and evaluate whether a larger and more diverse training set can make the model generalize better. A future work to our deep learning approach (Section 4.3) would be to utilize recent network architectures that take in 3D point-clouds as inputs. This reduces the necessity to convert 3D point-clouds into 2D range images, and avoids any loss of information which may happen during this process.

The WNOJ motion model derived in Section 5.3 has only been evaluated in simple simulations. Therefore, an obvious next step would be to evaluate our new motion prior on real-world lidar datasets, and compare against results obtained using the existing STEAM framework with the WNOA motion model. Moreover, our derivation relies on a number of approximations, with Equation (5.46) being an example. A future work is therefore to come up with exact derivations rather than approximate derivations where possible. Finally, our derivation of the WNOJ prior follows the idea of local pose variables as in [3], whereas there is a different way for deriving a motion prior shown in [6]. Therefore, an interesting extension would be to derive the WNOJ motion model using an alternative approach, and compare against results presented in this thesis.

Bibliography

[1] *MATLAB: Statistics and Machine Learning Toolbox; User's Guide*. MathWorks, 2017.

[2] Francesco Amigoni, Monica Reggiani, and Viola Schiaffonati. An insightful comparison between experiments in mobile robotics and in science. *Autonomous Robots*, 27(4):313, 2009.

[3] Sean Anderson and Timothy D Barfoot. Full steam ahead: Exactly sparse gaussian process regression for batch continuous-time trajectory estimation on se (3). In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 157–164. IEEE, 2015.

[4] Sean William Anderson. *Batch Continuous-Time Trajectory Estimation*. PhD thesis, University of Toronto, 2016.

[5] Tim D Barfoot, Chi Hay Tong, and Simo Särkkä. Batch continuous-time trajectory estimation as exactly sparse gaussian process regression. In *Robotics: Science and Systems*. Citeseer, 2014.

[6] Timothy D Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017.

[7] Timothy D Barfoot and Paul T Furgale. Associating uncertainty with three-dimensional poses for use in estimation problems. *IEEE Transactions on Robotics*, 30(3):679–693, 2014.

[8] Laszlo-Peter Berczi, Ingmar Posner, and Timothy D Barfoot. Learning to assess terrain from human demonstration using an introspective gaussian-process classifier. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3178–3185. IEEE, 2015.

[9] Ronald Clark, Sen Wang, Hongkai Wen, Andrew Markham, and Niki Trigoni. Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem. In *AAAI*, pages 3995–4001, 2017.

[10] Jing Dong, Byron Boots, and Frank Dellaert. Sparse gaussian processes for continuous-time trajectory estimation on matrix lie groups. *arXiv preprint arXiv:1705.06020*, 2017.

[11] Jing Dong, John Gary Burnham, Byron Boots, Glen Rains, and Frank Dellaert. 4d crop monitoring: Spatio-temporal reconstruction for agriculture. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3878–3885. IEEE, 2017.

[12] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[13] Renaud Dubé, Hannes Sommer, Abel Gawel, Michael Bosse, and Roland Siegwart. Non-uniform sampling strategies for continuous correction based trajectory estimation. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4792–4798. IEEE, 2016.

[14] Sara Farboud-Sheshdeh, Timothy D Barfoot, and Raymond H Kwong. Towards Estimating Bias in Stereo Visual Odometry. In *Computer and Robot Vision (CRV), 2014 Canadian Conference on*, pages 8–15. IEEE, 2014.

[15] Paul Furgale, Timothy D Barfoot, and Gabe Sibley. Continuous-time batch estimation using temporal basis functions. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2088–2095. IEEE, 2012.

[16] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[17] Natasha Gelfand, Leslie Ikemoto, Szymon Rusinkiewicz, and Marc Levoy. Geometrically stable sampling for the ICP algorithm. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on*, pages 260–267. IEEE, 2003.

[18] Craig Glennie and Derek D Lichti. Static calibration and analysis of the Velodyne HDL-64E S2 for high accuracy mobile scanning. *Remote Sensing*, 2(6):1610–1624, 2010.

[19] Javier Hidalgo-Carrió, Daniel Hennes, Jakob Schwendner, and Frank Kirchner. Gaussian process estimation of odometry errors for localization and mapping. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 5696–5701. IEEE, 2017.

[20] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[21] Alireza G Kashani, Michael J Olsen, Christopher E Parrish, and Nicholas Wilson. A review of LiDAR radiometric processing: From ad hoc intensity correction to rigorous radiometric calibration. *Sensors*, 15(11):28099–28128, 2015.

[22] Kirk MacTavish and Timothy D Barfoot. At all costs: A comparison of robust cost functions for camera correspondence outliers. In *Computer and Robot Vision (CRV), 2015 12th Conference on*, pages 62–69. IEEE, 2015.

[23] Will Maddern, Alastair Harrison, and Paul Newman. Lost in translation (and rotation): Rapid extrinsic calibration for 2d and 3d lidars. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3096–3102. IEEE, 2012.

[24] Martin Magnusson, Achim Lilienthal, and Tom Duckett. Scan registration for autonomous mining vehicles using 3D-NDT. *Journal of Field Robotics*, 24(10):803–827, 2007.

[25] Martin Magnusson, Andreas Nuchter, Christopher Lorken, Achim J Lilienthal, and Joachim Hertzberg. Evaluation of 3D registration reliability and speed-A comparison of ICP and NDT. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3907–3912. IEEE, 2009.

[26] Martin Magnusson, Narunas Vaskevicius, Todor Stoyanov, Kaustubh Pathak, and Andreas Birk. Beyond points: Evaluating recent 3D scan-matching algorithms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3631–3637. IEEE, 2015.

[27] P. McGarey, D. Yoon, T. Tang, F. Pomerleau, and T. D. Barfoot. "Field Deployment of the Tethered Robotic eXplorer to Map Extremely Steep Terrain". In *Field and Service Robotics*, 2017.

[28] Mustafa Mukadam, Jing Dong, Frank Dellaert, and Byron Boots. Simultaneous trajectory estimation and planning via probabilistic inference. In *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

[29] Austin Nicolai, Ryan Skeele, Christopher Eriksen, and Geoffrey A Hollinger. Deep learning for laser based odometry estimation. In *RSS workshop Limits and Potentials of Deep Learning in Robotics*, 2016.

[30] Simon T O'Callaghan and Fabio T Ramos. Gaussian process occupancy maps. *The International Journal of Robotics Research*, 31(1):42–62, 2012.

[31] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[32] Valentin Peretroukhin, Lee Clement, and Jonathan Kelly. Reducing drift in visual odometry by inferring sun direction using a Bayesian Convolutional Neural Network. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2035–2042. IEEE, 2017.

[33] Valentin Peretroukhin and Jonathan Kelly. Dpc-net: Deep pose correction for visual localization. *IEEE Robotics and Automation Letters*, 2017.

[34] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing ICP variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, 2013.

[35] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

[36] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[37] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-ICP. In *Robotics: science and systems*, volume 2, page 435, 2009.

[38] Jacopo Serafin and Giorgio Grisetti. NICP: Dense normal based point cloud registration. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 742–749. IEEE, 2015.

[39] Jacopo Serafin, Edwin Olson, and Giorgio Grisetti. Fast and robust 3D feature extraction from sparse point clouds. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4105–4112. IEEE, 2016.

[40] Mark Sheehan, Alastair Harrison, and Paul Newman. Self-calibration for a 3d laser. *The International Journal of Robotics Research*, 31(5):675–687, 2012.

[41] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. Point feature extraction on 3D range scans taking into account object boundaries. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 2601–2608. IEEE, 2011.

[42] Todor Stoyanov, Martin Magnusson, and Achim J Lilienthal. Point set registration through minimization of the L-2 distance between 3D-NDT models. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5196–5201. IEEE, 2012.

[43] T. Y. Tang, D. J. Yoon, F. Pomerleau, and T. D. Barfoot. "Learning a Bias Correction for Lidar-only Motion Estimation". 2018.

[44] Chi Hay Tong, Paul Furgale, and Timothy D Barfoot. Gaussian process gauss–newton for non-parametric simultaneous localization and mapping. *The International Journal of Robotics Research*, 32(5):507–525, 2013.

[45] Yanghai Tsin and Takeo Kanade. A correlation-based approach to robust point set registration. In *European conference on computer vision*, pages 558–569. Springer, 2004.

[46] William Vega-Brown, Abraham Bachrach, Adam Bry, Jonathan Kelly, and Nicholas Roy. Cello: A fast algorithm for covariance estimation. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3160–3167. IEEE, 2013.

[47] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2043–2050. IEEE, 2017.

[48] Ji Zhang, Michael Kaess, and Sanjiv Singh. On degeneracy of optimization-based state estimation problems. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 809–816. IEEE, 2016.

[49] Ji Zhang and Sanjiv Singh. LOAM: Lidar Odometry and Mapping in Real-time. In *Robotics: Science and Systems*, volume 2, 2014.