Unified Spatiotemporal Calibration of Egomotion Sensors and Planar Lidars in Arbitrary Environments

by

Jordan Marr

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Aerospace Science and Engineering
University of Toronto

Abstract

Unified Spatiotemporal Calibration of Egomotion Sensors and Planar Lidars in Arbitrary Environments

Jordan Marr
Master of Applied Science
Graduate Department of Aerospace Science and Engineering
University of Toronto
2018

This thesis aims to develop an automatic spatiotemporal calibration routine for lidars and egomotion sensors that relaxes many common requirements, such as the need for overlapping sensor fields of view, or calibration targets with known dimensions. In particular, a set of entropy-based calibration algorithms are extended to allow estimation of sensor clock time offsets in tandem with sensor-to-sensor spatial transformations. A novel Bayesian optimization routine is developed to address the non-smooth behaviour observed in the entropy cost function at small scales. The routine is tested on both simulation and real world data. Simulation results show that, given a set of lidar data taken from many different viewpoints, the calibration can be constrained to within less than 5 mm, 0.1 degrees, and 0.15 ms in the translational, rotational, and time-delay parameters respectively. For real-world data, in the absence of a reliable ground truth, we present results that show a repeatability of $\pm$ 4 mm, 1 degree, and 0.1 ms. When a monocular camera is used as the egomotion sensor, the routine is able to resolve the scale of the trajectory. A very brief analysis of the applicability of the method to Inertial Measurement Unit (IMU) to lidar calibration is presented.

Acknowledgements

I could not have completed this thesis without the support of some very important people. I would like to acknowledge first and foremost my supervisor, Dr. Jonathan Kelly. His professional guidance and intuition on sensor calibration has been invaluable to my work. My thanks also go to Dr. Steven Waslander for his review of this thesis. I am further indebted to my parents, Martha and Graham, and my brother Evan, for their personal support. Lastly, I'd like to thank my colleagues in STARS Lab, and in particular my good friend Emmett Wise, for creating a fun working environment that has been incredibly valuable to my professional development.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The exploitation of multiple sensor types is an important tool for the refinement of a mobile robotic platform's knowledge of the world. Sensor suites are usually chosen such that the disparate sensors' benefits complement each other. For example, the precise, lighting-invariant distance measurement of a Light-Detection-and-Ranging sensor (lidar) make it a popular choice for applications where collision avoidance is of critical importance, such as self-driving cars [1] [2] and fast moving aerial platforms [3]. However, the most ubiquitous, cheap, 2D scanning lidars lack the ability to measure outside of a single plane. Thus they are unable to measure their own motion when translating and rotating arbitrarily in 3D [4].

In contrast, camera systems are capable of measuring their own motion, provided there are enough identifiable features to track between frames [5]. Unfortunately, when estimating the trajectory from a single camera feed, there is an ambiguity in the scale of the trajectory, as a camera can only measure the bearing to a feature, and not the depth. A camera-lidar system is a naturally complementary sensor pair, as the precise metrical lidar data can be used to resolve scale ambiguities in the environment. This system's mapping capabilities are demonstrated in Figure 1.1b, where we reconstruct a point cloud by stitching together lidar scans based on a trajectory provided by the camera.

In any multi-sensor system, the data fusion process requires the system to be calibrated with precise knowledge of the 6 degree of freedom (6-DOF) inter-sensor spatial transforms, also known as the extrinsic sensor calibration. This permits the individual sensors' data to be combined into a single common reference frame. Manual measurement of the inter-sensor transforms is inaccurate for two reasons. First, any measurement is made difficult given that the sensor mounting surfaces may be in the way. Secondly, it is not known where the origin of a sensor's measurement frame is located within the sensor body. For these reasons, calibration methods that make use of the sensors' data

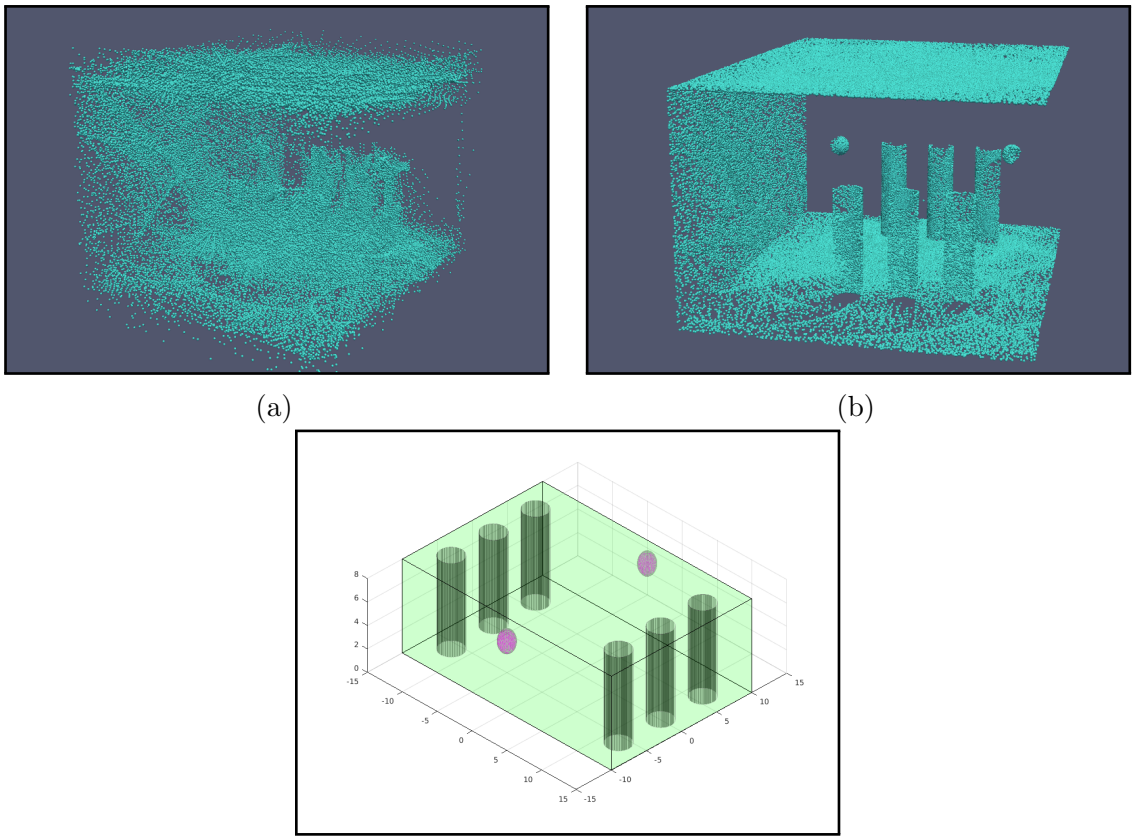


Figure 1.1: Point cloud reconstruction of a simulated environment, (a) before calibration (with calibration errors of 6 cm per translational parameter, 0.06 radians per rotational parameter, and 25 ms time delay error), and (b) after calibration has been performed. Note that planar surfaces appear to be planar after calibration. The simulation environment is pictured in (c) for reference.

usually result in more accurate estimates of the transforms. The canonical methods for calibration almost all make use of a specific calibration target, such as a checkerboard or other planar surfaces. A method that works in an arbitrary environment, absent any specific calibration targets, would be an improvement on these methods in allowing the re-calibration of a long-lifespan platform in the field.

Also of importance is the fusion of sensor data to a single temporal reference frame. The sensor clocks used to produce data timestamps might not be consistent. That is, they may differ from each other by some amount. Naively fusing the data based on the timestamps can result in inaccurate estimates about the environment, as there will be some mismatch in the sensor data when one "leads" the other. The temporal calibration of the sensor system permits the recovery of any offsets between the sensor clocks [6]. In [7], it was demonstrated that an accurate temporal calibration can be obtained if the spatial transform has already been calibrated to a sufficient degree of accuracy. We extend the method in [7] to simultaneously calibrate the time offsets in combination with the spatial transform. We calibrate by evaluating the "crispness" of the point cloud constructed with various calibration parameters. Specifically, we quantify the degree to which the points form surfaces, such as planes and curves. The temporal and spatial offsets are correlated through the structure of the point cloud. As shown in [7], if a poor time offset estimate is assumed, the "most crisp" point cloud will be achieved by an erroneous set of spatial calibration parameters, so the calibration routine will fail. The simultaneous calibration procedure ensures that this does not occur.

In this work, we make the following contributions:

- We show that time delay (temporal) calibration can be directly folded into the spatial calibration procedure using Renyi's Quadratic Entropy as an evaluation metric
- We allow for asynchronous data streams by utilizing a theoretically rigorous method for interpolating poses
- We demonstrate that full spatiotemporal calibration can be performed in environments with unknown and arbitrary structure
- We reduce the required number of cost function evaluations through the use of a fast Bayesian optimization procedure which does not require explicit Jacobian matrices to be computed
- We provide experimental evaluation on real world data

Our novel calibration procedure works by using a camera as an egomotion sensor, and passing its data to an existing Simultaneous Localization and Mapping (SLAM) routine which provides an estimated camera pose for each frame in the video. From these poses, we can formulate a continuous-time representation of the trajectory by interpolating between poses using a set of equations known as Simultaneous Trajectory Estimation and Mapping (STEAM). With a continuous-time camera trajectory, a set of timestamped lidar scans, and an estimate of the spatial transform and temporal offset between the sensors, we can stitch the lidar scans to the camera trajectory to create a point cloud. We then compute a cost value based on the entropy of the point cloud. We calibrate the sensor pair by formulating the problem as an optimization of this cost function, the inputs being the set of calibration parameters, and the outputs being an approximation of the entropy. We make use of Bayesian Optimization to circumvent the effects of local discontinuities of the gradient of the cost function.

We present simulation results that indicate calibration accuracies with errors of less than 5 mm, 0.1 degrees, and 0.15 ms in the translational, rotational, and time-delay parameters respectively, are readily achievable. For real-world data, in the absence of a reliable ground truth, we present results that show a repeatability of $\pm$ 4 mm, 1 degree, and 0.1 ms, over three datasets collected in different environments.

This thesis is structured as follows. In Chapter 2 we summarize related camera to lidar calibration methods. In Chapter 3, we provide motivation for our chosen cost function. Chapter 4 describes our method of estimating a continuous-time camera trajectory, and our chosen optimization routine. Chapter 5 provides our results on simulated and real data, Chapter 6 motivates the possibility of calibrating a 2D lidar and IMU system, and Chapter 7 provides conclusions and suggests future extensions.

Chapter 2

Related Work

We present a brief summary of the recent history of sensor-to-sensor calibration. We begin with a discussion of automatic spatial calibration, including target-based and target-free methods. We then discuss automatic temporal calibration—the recovery of sensor clock time offsets. We conclude with an analysis of the small body of literature concerning spatiotemporal calibration—the simultaneous estimation of spatial and temporal offsets—which presents a host of novel challenges.

2.1 Spatial Calibration

Spatial Calibration routines can generally be categorized as either target-based or target-free. The former category includes methods that require both sensors to view a known object, such as a checkerboard. The latter consists of methods designed to work with views of any arbitary scene, and constrain the problem with other assumptions.

2.1.1 Target-based Spatial Calibration

As we have discussed, manual spatial calibration is impractical for a number of reasons, including the difficulty involved in physically measuring distances when other parts of the sensor platform are in the way. For this reason, all of the calibration methods we will consider herein are "automatic" methods. That is to say they require only the data streams from both sensors, and no user intervention. However, this requires a numerical evaluation of the quality of the data fusion, which will be maximized with the true calibration parameters. An obvious method of evaluation is to validate that known calibration targets are reproduced faithfully. These calibration targets are objects with a known scale and pattern that can easily be picked up in both sensors data streams, and

that provide constraints to the optimization.

The method of Wasielewski and Strauss [8] is among the first target-based calibration routines. Their method calibrates a camera and planar lidar pair by having the sensors simultaneously view a target creased along a line seperating black and white sections. Edge detection is used to determine the equation of the line in the frame of the camera. Because of the crease in target, the lidar scan displays a kink at the point where it intersects with the line. The system builds up several sets of camera and lidar measurements with the target in different positions. The system uses the set of intersection points to solve for the calibration parameters under least-squares constraints. The idea of constraining the calibration with points of known location in each sensor frame has remained prevalent in calibraion research.

Zhang and Pless [9] expand upon the target-based method by introducing a checkerboard as the target. The camera can detect all of the checkerboard's interior corners, and so has far more features to constrain its estimate of the target's position and orientation using the Perspective n-Point algorithm [10]. The lidar scans the length of the checkerboard, partially constraining the orientation of the target relative to the lidar. Multiple measurements are required, each with the target in a different position, and the user must provide the dimensions of the checkerboard. The routine then solves for the lidar to camera transformation which produces the minimum least-square error in the projection of the lidar points into the checkerboard plane observed by the camera. This method does not estimate temporal offsets.

There have been many extensions and variations to the target-based calibration method. Guo and Roumeliotis [11] develop a method that analytically solves for calibration parameters using a modified planar target. Zhou [12] presents a method that intelligently selects the permutation of variables to avoid singularities that can occur with the given rotation parameterization. This method is numerically stable and allows a minimal solution to the problem requiring only three measurements (three camera images and three lidar scans).

Though it is outside the scope of this work, target-based methods are even better suited for the calibration of 3D lidars to cameras. In contrast with 2D lidars, these sensors measure outside of a single plane and thus provide a 3D snapshot of the world in the form of a depth map. Geiger et al. [13] demonstrate that the camera to 3D lidar transform can be accurately evaluated using a single camera image and lidar scan, by having each sensor observe multiple checkerboards simultaneously. The orientations of each checkerboard can be directly observed by each sensor (unlike with a 2D lidar) and aligned under least-squares constraints to obtain the calibration.

There are two main difficulties in using any of the above methods. First, both sensors must view the target at the same instant, so they must have overlapping fields of view. Secondly, they require a means of moving the target in between each measurement. Bok et al. [14] present a method aimed at solving the first issue. Their method allows for non-overlapping sensor fields of view by having the camera view a checkerboard and the lidar view either a planar surface or the line connecting two planar surfaces. However, for this routine to work, the user must supply the transformation between the checkerboard and the lidar target. Estimating this transform to a sufficient degree of accuracy presents a new challenge entirely.

2.1.2 Target-free Spatial Calibration

Many target-free camera-to-2D lidar calibration methods are designed to automatically select correspondences from whatever objects are viewed by the sensors. Yang et al. [15] calibrate a 2D lidar and camera pair by using mutually observed line and point features as correspondences. The use of direct correspondences between the data streams greatly simplifies the structure of the optimization, as the cost function is the squared sum of the reprojection error of the correspondences as viewed by the camera. This is simpler than minimizing the alignment error of a checkerboard viewed by both sensors. Similar approaches have been applied to camera-to-3D lidar calibration by Scaramuzza et al. [16] and Moghadam et al. [17]. The former requires manual selection of the point and line correspondences, while the latter extracts them automatically. The method of Castorena et al. [18] takes the edge alignment paradigm even further for camera to 3D lidar calibration. Rather than directly extracting edges from the lidar point cloud, the authors fuse the entire lidar point cloud with the camera image to obtain a fused depth map. They compute a score penalizing the misalignment of lidar data with edges extracted from the camera image. This allows for simultaneous calibration and data fusion by using the data fusion quality as the optimization metric.

The work of Pandey et al. [19] can be considered an abstraction on the targetless edge alignment approach to calibration. This method works by maximizing the mutual information between camera and lidar data, and can calibrate both camera-2D lidar and camera-3D lidar pairs. Work developed by Taylor and Nieto [20] around the same time presents a similar mutual information based approach. The authors calibrate a hyperspectral camera to 3D lidar.

Brooskhire and Teller develop a calibration method that requires only the individual trajectories of two or more sensors arranged within the same plane [21] or arbitrarily in

3D space [22]. The authors show that except for a few degenerate trajectories (which are discussed in detail), the calibration transform that best aligns the sensor trajectories under least-squares conditions is directly observable. The simplicity of this formulation is attractive, and it has been applied to the vehicle-mounted camera to 3D lidar calibration problem [23]. The drawback of this approach is that it requires every sensor have the capability to measure its own egomotion. 2D lidars can achieve this using scan matching if they are traveling within a single plane. However, when travelling arbitrarily in three dimensions, 2D lidars only observe the environment via a single planar scan. Thus, there is always an unconstrained rotation of the lidar about the scanning plane [4], so it can not directly observe its trajectory.

Le Gentil et al. [24] adapt egomotion based calibration for the transform between an IMU and 3D lidar by adding a few constraints. In particular, the lidar moves such that it always scans a set of three mutually orthogonal planes (for example, the vertex where two walls connect with the floor). The lidar can easily segment which parts of its line scan lie on each plane. Coupled with the knowledge that the three planes are mutually orthogonal, this allows the lidar to measure its relative movement between measurement times by a SLAM routine which represents the map entirely with planes [25, 26].

Several groups have developed entropy-based calibration methods for 2D lidar to 2D lidar [27], and 2D or 3D lidar to an arbitrary egomotion sensor [28, 7, 29]. These calibration algorithms all determine the extrinsic transform between one or more sensor pairs (including at least one lidar), by minimizing the Rényi Quadratic Entropy (RQE) [30] of the resulting lidar points cloud(s). Therefore, they fall under the category of simultaneous data fusion and calibration algorithms. The egomotion sensor is required to provide a base trajectory, which the lidar data can be registered to via the calibration transform. The work by Lambert et al. adapts [28] to allow use of a single monocular camera as the egomotion sensor. It does so by adding the scale of the camera trajectory to the calibration transform as a parameter to be estimated. They key discovery in this work is that there is a single global minimum to the cost function under the correct scale and calibration parameters. That is to say, the point cloud displays the most "crispness" with the correct parameters.

2.2 Temporal Calibration

Temporal calibration consists of the recovery of any offset from one sensor clock to another. This step must be performed every time the system is used, as it will change each time the sensors are powered on.

2.2.1 Two Way Timing Data and Hardware Methods

A sensor data packet is timestamped by the sensor's internal clock, which is extremely unlikely to perfectly agree on the time with other sensors' clocks. For this reason, it is imperative to consider the time offset required to align sensor data streams to properly fuse their data. Much of temporal calibration literature builds on the work of Cristian [31], who develops a probabilistic model for estimating the time delay between system clocks. This work, and the work that builds on it (e.g., [32, 6]) requires two way timing data. This means that a sensor client sends the original data packet and receives a response packet from the host computer. The specified synchronization routines require that both computers provide a timestamp when they send and receive a packet (either the original or the response).

Estimating sensor clock offset is complicated by the fact that the difference in packet sending and receiving time is caused by three factors. The first is the clock offset—the difference between the sensor clocks. The second factor is the non-zero network delay - the minimum amount of time it takes for a packet to travel between computers. The third factor is random packet delay, which is caused by high CPU load among other things. We wish to estimate the first quantity, but we can only measure the timestamp differences - the sum of the three quantities. To complicate matters further, the first quantity will change over time, as the various clocks exhibit different drift rates. That is, they count time at slightly different rates

The work by Harrison and Newman [6] uses the two way timing data to set upper and lower bounds on the offset and to probabilistically estimate the offset and offset drift rate (the time derivative of the offset). The key here is that the majority of packet transfers will be free from random packet delay. As the routine collects more measurements, it can continually refine the upper and lower bounds on the clock offset, improving its estimate of the offset and the drift rate.

Unfortunately, the vast majority of low cost sensors can not provide two way timing data. Instead, they provide a packet with a timestamp at the time of measurement. Other methods have been developed to synchronize sensor clocks under this condition, such as using external hardware [33], or interrupt signal lines [34] to detect the exact measurement instant. These methods are very precise, and should be considered if the calibration hardware is available, or the sensors have measurement interrupt lines. However, these are not available for most low cost platforms and sensors.

2.2.2 Data Driven Methods

In many cases, the only tool available for time delay estimation is the sensor data. Such data driven methods follow a similar structure to spatial calibration estimation, with the time delay being added as a parameter to estimate using some optimization criteria. Kelly et al. [35, 36] formulate a solution by aligning two sensor trajectories using a variant of Iterative Closest Point (ICP). The time delay is introduced as a parameter to ICP, meaning that the sensor trajectories can be slid backwards and forwards through time to achieve a better alignment. The optimized time-delay is that which achieves the best trajectory alignment.

Tungadi and Kleeman [37], and Furrer et al. [38] have developed similar methods for time delay estimation using sensor trajectory information. The former minimizes the error between relative sensor poses (i.e., the pose changes between measurement instants), and the latter minimizes error between interpolated angular velocity measurements.

Mair et al. [39] move the sensor platform through specialized sensor trajectories. This allows the authors to estimate the time delay by maximizing the cross-correlation and phase congruency of the sensor trajectories. This can be considered an abstraction on the other trajectory alignment based methods. It is worth noting then that all of the methods considered in this section require both sensors to be able to measure their own egomotion.

2.3 Simultaneous Spatiotemporal Calibration

Compared with the body of literature on spatial calibration and temporal calibration, the amount of work concerning simultaneous spatiotemporal calibration is small. The problems of calibrating either spatial or temporal offsets is made easier if the other has already been calibrated. For this reason, much of calibration literature assumes that time offsets have been pre-calibrated [7], spatial offsets have been calibrated [37], or alternates between optimizing the two [38]. There are very few methods that wrap spatial and temporal offsets into a single estimator.

Li and Mourikis [40] add sensor time delay to an Extended Kalman Filter routine that additionally estimates the trajectory of a base sensor (an IMU for their purposes) and the transform from the IMU to a camera. The use of an EKF allows the calibration to be performed online, at the expense of the improved accuracy available with offline batch estimation methods. Furthermore, adapting this method to work with a lidar would require the formulation of a measurement model for a 2D lidar in a 3D environment.

Unlike a camera, a lidar is a sparse sensor, so formulating a measurement model in 3D is difficult. The scanning plane is constantly changing, and it is hard to predict what the lidar should observe without making significant assumptions about the environment.

Rehder et al. [41, 42] develop a novel lidar measurement model by making the assumption that the environment contains many planes. Building on the method of Furgale et al. [43] for spatiotemporal calibration of an IMU and camera system, the lidar-compatible method works in a similar fashion. A set of batch least-squares estimators are derived based on each sensor's measurement model. The optimizer then solves for base sensor trajectory, sensor-to-sensor transforms, time delays, and other parameters such as gyroscope biases. The lidar measurement model is formulated by initializing a map of planes observed in the environment. This allows easy computation of the expected return value for the distance of a lidar laser beam aimed at any of these planes. An additional point to note is that the authors never directly solve the camera to lidar or IMU to lidar calibration problem. Instead, the lidar is only included as part of a suite containing all three sensors.

Chapter 3

Calibration Theory

The key insight to our method follows [27, 28, 7]. When one creates a point cloud of a real world environment, the majority of the points will lie on a set of a few surfaces. In man-made environments, these tend to be regular objects such as planes arranged at right angles. However, the principle holds true even in natural environments. In a forest for example, the surfaces include the ground, the trunks of tree, roots, etc. Any environment that we can scan is relatively structured.

To achieve an accurate reconstruction of the environment, we require accurate calibration parameters. If any parameters differ from their ground truth values, each lidar scan will be affected in a highly nonlinear fashion. The ultimate effect is to "muddle" the point cloud. That is to say the lidar points spread out from surfaces, creating a blurring effect demonstrated in Figure 1.1a. To highlight the mechanics of what causes this, Figures 3.1 and 3.2 present a trivial example of an egomotion sensor and a lidar arranged on a mobile platform for mapping purposes. The two sensors lie along a line, so the calibration is a single distance value (in general, the calibrations we wish to estimate will be arbitrary 3D transformations). Figure 3.2 shows what happens if this distance is misestimated. Scans of the same object taken at different times will be transformed in different ways, and the average distance between points in the point cloud will increase. Therefore, the point cloud becomes less structured. As in our example, scanned line segments taken at different times no longer neatly align. This is a minimal example of the same blurring effect as in Figure 1.1a.

We proceed in this chapter by detailing the registration of 2D lidar scans to a set of egomotion sensor poses to create a full 3D point cloud. We then present a brief background on the concept of entropy and the various mathematical formulations that exist to quantify it. We conclude with an explanation of how the Renyi's Quadratic Entropy formulation will allow us to compute the entropy of a 3D structured point

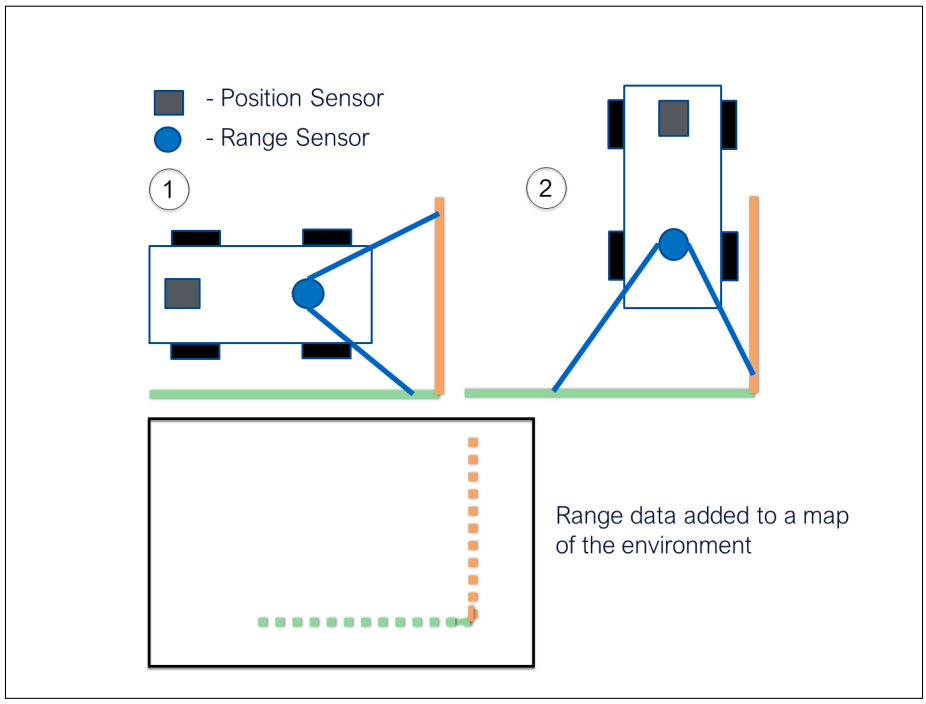


Figure 3.1: A mobile position sensor and lidar platform scans the same surface at two different timesteps. When the calibration - the distance between the sensors - is known accurately, the platform can construct an accurate map of the environment.

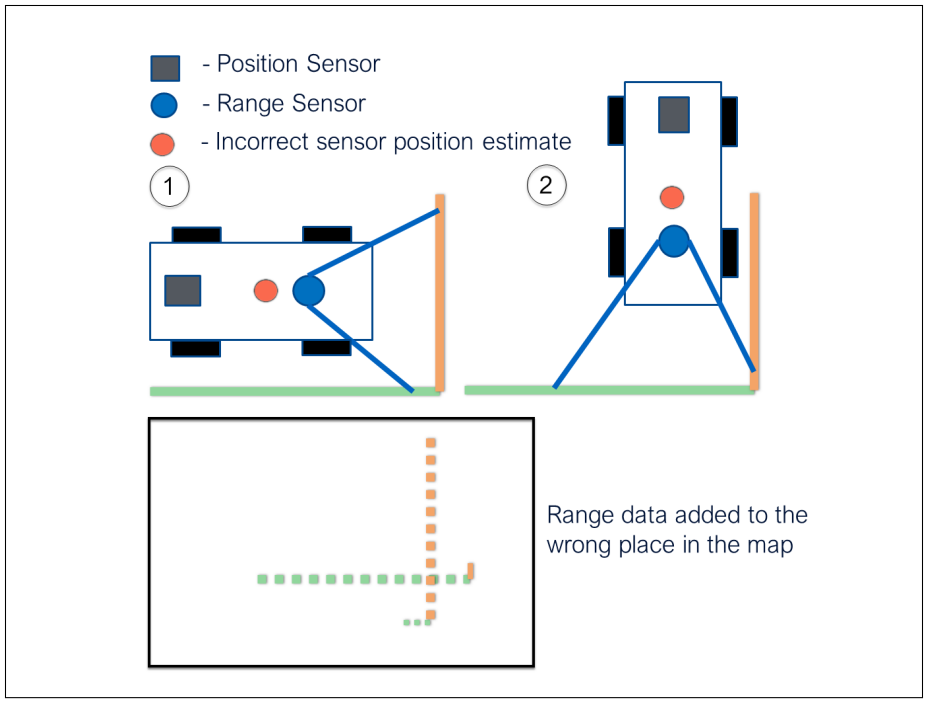


Figure 3.2: A mobile position sensor and lidar platform scans the same surface at two different timesteps. When the calibration - the distance between the sensors - is underestimated, the map that is constructed is muddled and has higher entropy.

cloud. This will serve as our cost function, taking poses and lidar scans as input, and returning a single scalar entropy value.

3.1 Kinematics

We make extensive use of Lie groups for representing sensor poses. A full discussion of the use of Lie groups in estimation is beyond the scope of this work. We refer readers to [44] for an excellent introduction to Lie groups, and a motivation for their use in estimation.

3.1.1 Point Cloud Construction

We choose to represent sensor poses with homogeneous transformation matrices of the form

$$\mathbf{T} = \begin{bmatrix} \mathbf{C} & \mathbf{r} \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{3.1}$$

where $\mathbf{C}$ is a 3×3 rotation matrix[1], and $\mathbf{r} \in \mathbb{R}^3$ is a translation vector. $\mathbf{C}$ and $\mathbf{T}$ are members of the SO(3) and SE(3) Lie groups respectively. We store transformations in a minimal, 6 parameter representation in the $\mathfrak{se}(3)$ tangent space.

We assume that the platform's base sensor (a monocular camera for our application, but any egomotion sensor could be used) provides a set of M 6-DOF poses, $\mathbf{Y}$:

$$\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_M\}, \quad \mathbf{y}_m = \left[\rho_{1,m}\ \rho_{2,m}\ \rho_{3,m}\ \phi_{1,m}\ \phi_{2,m}\ \phi_{3,m}\right]^T, \tag{3.2}$$

where $\boldsymbol{\rho}_m = \left[\rho_{1,m}\ \rho_{2,m}\ \rho_{3,m}\right]^T$ and $\boldsymbol{\phi}_m = \left[\phi_{1,m}\ \phi_{2,m}\ \phi_{3,m}\right]^T$ are the translational and rotational portions of $\mathbf{y}_m \in \mathfrak{se}(3)$ respectively. We can obtain a homogeneous representation for $\mathbf{y}_m$ by converting from $\mathfrak{se}(3)$ to SE(3) through the matrix exponential.

$$\mathbf{T}'_{G,C_m} = \exp\left(\begin{bmatrix} \boldsymbol{\phi}_m^\wedge & \boldsymbol{\rho}_m \\ \mathbf{0}^T & 0 \end{bmatrix}\right), \tag{3.3}$$

$$\boldsymbol{\phi}^\wedge = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix}. \tag{3.4}$$

[1] A rotation matrix is an orthonormal matrix whose columns represent the projection of one coordinate frame's base axes into a second coordinate frame that is rotated arbitrarily from the first.

The $'$ superscript indicates that the pose has not been scaled, and is therefore not the same as the actual camera pose, which we will denote $\mathbf{T}_{G,C_m}$. We further note that each pose $\mathbf{y}_m$ has an associated timestamp t_m and pose covariance matrix $\mathbf{Q}_m$.

The lidar provides a set of K observations at N timesteps, $\mathbf{Z}$, where

$$\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_N\}, \quad \mathbf{z}_n = \{\mathbf{z}_n^{(1)}, \mathbf{z}_n^{(2)}, ..., \mathbf{z}_n^{(K)}\}, \tag{3.5}$$

$$\mathbf{z}_n^{(k)} = \begin{bmatrix} x_n^{(k)} & y_n^{(k)} \end{bmatrix}^T, \tag{3.6}$$

and each scan $\mathbf{z}_n$ has associated timestamp t_n. We express each point in lidar frame $\underrightarrow{\mathcal{F}}_{L_n}$ as

$$\mathbf{p}_{L_n}^{(k)} = \begin{bmatrix} x_n^{(k)} & y_n^{(k)} & 0 & 1 \end{bmatrix}^T. \tag{3.7}$$

Here we see the main limitation of a 2D lidar as opposed to a 3D lidar. The former only measures within a single scanning plane, so the measured point $\mathbf{z}_n^{(k)}$ only has two coordinates, $x_n^{(k)}$ and $y_n^{(k)}$.

We likewise express each camera pose as a homogeneous transformation from camera frame $\underrightarrow{\mathcal{F}}_{C_m}$ to a fixed global frame $\underrightarrow{\mathcal{F}}_{G}$, where matrix $\mathbf{T}_{G,C_m}$ is the homogeneous representation of pose $\mathbf{y}_m$.

Our goal is to estimate the set of (constant) transform parameters from the lidar frame $\underrightarrow{\mathcal{F}}_{L}$ to the camera frame $\underrightarrow{\mathcal{F}}_{C}$. We omit $_n$ and $_m$ in the coordinate frame notation here to indicate these frames never change with respect to each other (the lidar and camera are rigidly attached).

$$\boldsymbol{\Xi} = \begin{bmatrix} x_L & y_L & z_L & \phi_L & \theta_L & \psi_L & s & t_d \end{bmatrix}^T. \tag{3.8}$$

The six spatial parameters are used to construct $\mathbf{T}_{C,L}$, the rigid body transformation matrix from the lidar frame $\underrightarrow{\mathcal{F}}_{L}$ to the camera frame $\underrightarrow{\mathcal{F}}_{C}$. The scale factor s is applied to the translational component of the camera pose (the top-right 3x1 elements of $\mathbf{T}'_{G,C_m}$).

$$\mathbf{T}_{G,C_m} = \begin{bmatrix} \mathbf{C}_{G,C_m} & s\mathbf{r}_G^{C_m,G'} \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{3.9}$$

where $\mathbf{C}_{G,C_m}$ and $\mathbf{r}_G^{C_m,G'}$ are the rotational and translational portions of $\mathbf{T}'_{G,C_m}$ obtained following the conversion from $\mathfrak{se}(3)$ to SE(3).

We account for time delay by adopting a continuous time representation of the sensor trajectories:

$$\mathbf{T}_{G,L}(t) = \mathbf{T}_{G,C}(t + t_d)\mathbf{T}_{C,L}. \tag{3.10}$$

$\mathbf{T}_{G,L}(t)$ is evaluated at $t = t_1, t_2, ..., t_N$ to obtain a corresponding lidar pose for every scan $\mathbf{z}_n$. We obtain $\mathbf{T}_{G,C}(t+t_d)$ by interpolating between camera poses. If t_d is positive, the camera clock is leading the lidar clock, and vice-versa for negative t_d. We use the STEAM model [45, 46, 47] for interpolation, explained in greater detail in Section 4.1.2.

We estimate the position of a lidar point in the global frame via the inverse sensor model:

$$\hat{\mathbf{p}}_{G,n}^{(k)} = h^{-1}(\mathbf{p}_{L_n}^{(k)} \mid \mathbf{Y}, \boldsymbol{\Xi}) = \mathbf{T}_{G,L}(t_n)\mathbf{p}_{L_n}^{(k)}. \tag{3.11}$$

We omit the homogeneous component so that $\hat{\mathbf{x}}_{G,n}^{(k)} \leftarrow \hat{\mathbf{p}}_{G,n}^{(k)}$.

$$\hat{\mathbf{x}}_{G,n}^{(k)} = \begin{bmatrix} x_{G,n}^{(k)} \\ y_{G,n}^{(k)} \\ z_{G,n}^{(k)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \hat{\mathbf{p}}_{G,n}^{(k)} \tag{3.12}$$

We use the Jacobian of this model and the camera pose covariances to obtain a covariance estimate of points in the world frame,

$$\boldsymbol{\Sigma}_n^{(k)} = \mathbf{J}_n^{(k)} \mathbf{S} \mathbf{Q}_n \mathbf{S}^T \mathbf{J}_n^{(k)^T}, \tag{3.13}$$

$$\mathbf{S} = \begin{bmatrix} s\mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_3 \end{bmatrix}, \quad \mathbf{J}_n^{(k)} = \frac{\partial h^{-1}(\mathbf{x}_{L_n}^{(k)} | \mathbf{Y}, \boldsymbol{\Xi})}{\partial \mathbf{y}_n},$$

where $\mathbf{y}_n$ is the interpolated 6-DOF camera pose used to generate $\mathbf{T}_{G,C}(t_n + t_d)$ and $\mathbf{Q}_n$ is its associated uncertainty. $\mathbf{S}$ is included to appropriately scale the uncertainty of the translational part of the poses by the same scale factor as the translations themselves. The form of $\mathbf{J}_n^{(k)}$ is given in Appendix A.2. Note that $\mathbf{Q}_n$ is the covariance matrix of the interpolated pose, rather than the covariance of any of the camera poses in $\mathbf{Y}$. The requirement for an interpolation scheme that is able to provide an estimate of uncertainty was a major motivation in choosing the STEAM model. Finally, we obtain a set of 3D points $\hat{\mathbf{x}}_{G,n}^{(k)} \in \hat{\mathbf{X}}$ expressed in the global frame, each with associated 3×3 covariance matrix $\boldsymbol{\Sigma}_n^{(k)}$ and timestamp t_n.

3.2 Entropy

3.2.1 Background

The use of entropy to quantify the amount of information provided by a series of events is attributed to Shannon [48]. In our context, an "event" is the measurement of a lidar

point at a specific point in the world. There is some uncertainty about the measured value, so it can be formulated as a probability distribution. For a series of N independent events, with respective probabilities $p_1, p_2, \ldots, p_N$ defined by the probability distribution $\mathcal{P}$, the Shannon entropy is given by

$$H_{SHANNON}(\mathcal{P}) = \sum_{i=1}^{N} p_i \log_b \frac{1}{p_i}. \tag{3.14}$$

The choice of base b is arbitrary to the overall pattern of the entropy. b is commonly chosen to be 2, in which case the units of $H_{SHANNON}(\mathcal{P})$ are referred to as "bits". Shannon's measure of entropy was formulated to satisfy three requirements for information [48][49].

1. **Continuity**: $H(\mathcal{P})$ must be continuous in p_i.

2. **Additivity**: If $\mathcal{P}$ and $\mathcal{Q}$ are two independent probability distributions, the entropy of their direct product must be additive:

$$H_{SHANNON}(\mathcal{P} * \mathcal{Q}) = H(\mathcal{P}) + H(\mathcal{Q}). \tag{3.15}$$

3. **Uniform property**: $H(\mathcal{P})$ is maximized when all events are equally probable, $p_i = 1/N \quad \forall \quad i \in \{1, 2, ..., N\}$. Moreover, if this is the case, then $H(\mathcal{P})$ increases monotonically with the value of N.

The Additivity property follows from the definition that $\mathcal{P}$ and $\mathcal{Q}$ are independent. All of the information gained from observing their events is novel, as neither has any bearing on the other. The total information gained must therefore be the sum of the information provided by both sets of events.

The Uniform property follows from the understanding that the uncertainty in the potential outcome is highest when all events are equally probable. Furthermore, if all potential outcomes are equally probable, then uncertainty will be higher the more potential outcomes there are.

Renyi [30] formulated a more general family of equations to quantify entropy by relaxing the Additivity requirement:

$$H_\alpha(\mathcal{P}) = \frac{1}{1-\alpha} \log \left(\sum_{i=1}^{N} p_i^\alpha \right), \ \alpha > 0, \ \alpha \neq 1. \tag{3.16}$$

This family of equations is referred to as α entropy, and tends toward Shannon entropy

as $\alpha \to 1$. Although this function lacks the Additivity property of Shannon entropy, it has been shown to be maximized at the same points as Shannon entropy[50].

3.2.2 Point Cloud Entropy

As hypothesized in previous work [27, 28, 7] a proper reconstruction of the environment will represent the most ordered or lowest entropy of all possible reconstructions. One can suggest examples where this would not be the case (for example, a set of points distributed evenly throughout a volume). However, it is hard to imagine a real world environment that has this form. Therefore, the entropy of the reconstructed point cloud will serve as our metric for evaluating the calibration parameters.

We quantify the probability of scanning a point at location $\mathbf{x}$ in $\mathbb{R}^D$ using Parzen window density estimation [51], which introduces a bandwidth parameter h. This allows us to formulate a probabilistic model of what the world looks like. As there is positional uncertainty in the coordinates of a scanned lidar point, the Parzen window density allows us to "validate" a point's position. If the point coordinate is within a cluster of many other points, it is likely accurate. Parzen window density estimation is thus well-suited to our previously stated assumptions about environment structure as it implicitly assumes points in a point cloud, $\mathbf{x}_i, \;\; i = 1, 2, \ldots, M$, are structured into surfaces. The probability is given by

$$p(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^{M} \frac{1}{h^D} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right). \tag{3.17}$$

The function $K(\cdot)$ is a kernel function used to weight the distances. For our purposes, we can use a Gaussian kernel function, with mean $\mathbf{x}_i$, introducing a covariance matrix in place of the bandwidth parameter h:

$$p(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^{M} \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}_i, \mathbf{\Sigma}_i + \sigma^2 \mathbf{I}), \tag{3.18}$$

where $\mathcal{N}(\mathbf{x}; \mu, \mathbf{\Sigma})$ is the value at $\mathbf{x}$ of a Gaussian distribution with mean μ and covariance matrix $\mathbf{\Sigma}$.

Intuitively, we are weighting point $\mathbf{x}$'s distance to $\mathbf{x}_i$ by the uncertainty in the latter's position. This uncertainty is made up of the sum of point $\mathbf{x}_i$'s covariance matrix ($\mathbf{\Sigma}_i$) and isotropic noise ($\sigma^2\mathbf{I}$) added to capture the uncertainty in range measurements.

With this formula for probability, the Shannon entropy can not be solved analytically, and computational approximations are expensive to compute. However, our aim is to

minimize the entropy, so we are free to choose another metric that is minimized at the same point. As shown in previous work [27, 28, 7], using the α-entropy of order 2 allows us to solve the integral in the entropy formula analytically. This equation, also known as the Renyi's Quadratic Entropy (RQE), is given by :

$$H(\hat{\mathbf{X}}) = -\log \int p(\mathbf{x})^2 d\mathbf{x}. \tag{3.19}$$

We can insert our equation for $p(\mathbf{x})$ and expand into a double sum over all points in the point cloud:

$$H(\hat{\mathbf{X}}) = -\log \int \left(\frac{1}{M} \sum_{i=1}^{M} \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}_i, \mathbf{\Sigma}_i + \sigma^2 \mathbf{I}) \right)^2 d\mathbf{x} \tag{3.20}$$

$$= -\log \left(\frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \int \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}_i, \mathbf{\Sigma}_i + \sigma^2 \mathbf{I}) \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}_j, \mathbf{\Sigma}_j + \sigma^2 \mathbf{I}) d\mathbf{x} \right). \tag{3.21}$$

The integral above is a convolution of two Gaussian distributions and can be simplified as follows

$$\int \mathcal{N}(x; \mu_1, \sigma_1^2) \mathcal{N}(x; \mu_2, \sigma_2^2) dx = \mathcal{N}(\mu_1 - \mu_2; 0, \sigma_1^2 + \sigma_2^2) = \mathcal{N}(\mu_2 - \mu_1; 0, \sigma_1^2 + \sigma_2^2), \tag{3.22}$$

$$H(\hat{\mathbf{X}}) = -\log \left(\frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \mathcal{N}(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j; \mathbf{0}, \mathbf{\Sigma}_i + \mathbf{\Sigma}_j + 2\sigma^2 \mathbf{I}) \right). \tag{3.23}$$

Our choice of $\alpha = 2$ has allowed us to make the above simplifications, leaving us with a very attractive formula for the entropy. The term inside the logarithm is simply the sum of all point-to-point distances within the point cloud, weighted through the Gaussian kernel. This weighting makes the equation smooth, and robust to outliers, as the value of the Gaussian tends to 0 for large point-to-point distances. Furthermore, using the covariance sum as a bandwidth parameter means that large point-to-point distances have less effect on the entropy if we are highly uncertain about either point's position.

Chapter 4

Calibration Algorithm

It was shown empirically in [7, 29], that the entropy of a point cloud is convergent with respect to the spatial calibration parameters, and the scale of the camera trajectory. That is, there is a single global minimum entropy at or very near the ground truth parameters. We demonstrate herein that the same is true for the time-delay between the sensors, and that there is furthermore a single global minimum in the 8-DOF parameter space that is highly accurate with respect to the ground truth parameters. We proceed by outlining our procedure for constructing a point cloud from a set of camera poses, lidar scans, and an estimate of the 8-DOF calibration vector. We then present our formula for point cloud entropy, and highlight a number of simplifications we make to the computation to make it tractable on a CPU. This will allow us to formulate the calibration as an optimization problem, with the camera poses and lidar scans as static data, and the calibration vector as the input to the cost function, which the entropy will be minimized with respect to. The entire routine is presented visually in Figure 4.1.

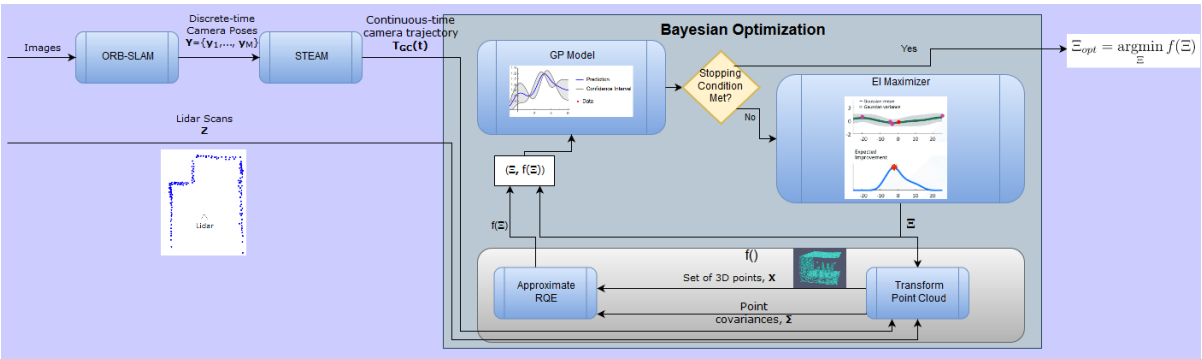


Figure 4.1: Summary of the calibration routine. Proceeding with camera images and lidar scans as given, static data, we solve for the calibration parameters resulting in the point cloud reconstruction with the minimum entropy value.

4.1 Camera Trajectory Estimation

We formulate the camera trajectory in continuous time, proceeding in two steps. First, we estimate a set of discrete time camera poses at the measurement instants (each frame in the video stream). We then allow for interpolation of the camera pose at any time by using the Simultaneous Trajectory Estimation and Mapping (STEAM) model [45, 46, 47]. Each measurement time pose has an associated 6x6 covariance matrix representing the uncertainty in the transform. Furthermore, we can use STEAM to compute a covariance matrix for any interpolated pose. This allows us great flexibility to process asynchronous camera and lidar data, and to evaluate the effect of the inter-sensor time delay on the point cloud entropy.

4.1.1 Discrete Time Camera Pose Estimation

Our algorithm is designed to work with any egomotion sensor. To this end, we abstract out the specifics of the egomotion sensor used, and assume that we are able to supply a set of M sensor poses $\mathbf{y}_m$ with associated timestamps $t_m,\ m = 1, \ldots, M$. Therefore, as a preprocessing step, we require a means of converting from the data supplied by the sensor (e.g., frames of a camera stream, IMU inertial rates) to a set of poses and timestamps. For our monocular camera based system, we use ORB-SLAM [52].

As with all monocular SLAM algorithms, ORB-SLAM identifies a set of visually distinctive features in each video frame. Through the inverse camera model, it estimates the 3D coordinates of each point relative to the camera focal point.

$$\boldsymbol{\pi}_i = \begin{bmatrix} f_u \frac{x_{i,j}}{z_{i,j}} + c_u \\ f_v \frac{y_{i,j}}{z_{i,j}} + c_v \end{bmatrix}, \tag{4.1}$$

where $\boldsymbol{\pi}_i$ is the pixel coordinates of landmark j in camera frame i, $\begin{bmatrix} x_{i,j} & y_{i,j} & z_{i,j} \end{bmatrix}^T$ are the coordinates of the point relative to the camera, f_u and f_v are the horizontal and vertical focal lengths, and $c_u$and c_v are the horizontal and vertical principal points. The focal lengths and principal points are known, so the algorithm can solve for the values $\frac{x_{i,j}}{z_{i,j}}$ and $\frac{y_{i,j}}{z_{i,j}}$ from the measured pixel coordinates. As we have mentioned, monocular VO can determine camera trajectory up to an ambiguous scale factor. The cause is apparent from Equation (4.1), as we can only compute the relative values of the x and y coordinates to the depth z, but there is no way of determining the depth value.

As the camera moves and captures more video frames, it tracks features, and can estimate its own movement based on the perceived motion of features. ORB-SLAM is

the state of the art algorithm for monocular SLAM largely because of its effectiveness in determining correspondences, that is, correctly matching point landmarks between frames, despite appearance changes caused by viewing from a different viewpoint. The system uses ORB features, which are very fast to compute relative to other popular feature choices such as SIFT [53] and SURF [54], and invariant to lighting and viewpoint changes. This allows the algorithm to track many features accurately, contributing to the overall accuracy in camera pose estimation.

ORB-SLAM's design informs what kind of environments work well for tracking. In particular, we want to ensure the camera views visually salient areas so that there will be many features available for tracking. We want to capture areas with lots of transitions between light and dark colours. Anecdotally, we have found that posters and printed copies of research papers provide accurate tracking because ORB-SLAM is very effective at tracking black characters on light coloured backgrounds. Furthermore, we want to ensure lighting is fairly consistent throughout the video, so well-lit areas work well for data capture.

Ultimately, we obtain a camera pose for each frame of the camera data. The timestamp is given by the frame's associated timestamp. ORB-SLAM does not directly provide covariance matrices for the camera poses. However, as with [29], we find that a standard deviation of 0.01 m and 0.025 rads for each parameter is an accurate reflection of the pose uncertainty.

4.1.2 SE(3) Pose Interpolation

To optimize the time delay parameter with respect to its effect on RQE, we require a method of evaluating the pose of the camera-lidar system at any arbitrary time during its trajectory. Interpolation is carried out using the STEAM model [45]. This model interpolates between 6-DOF poses that are members of the SE(3) group (i.e., homogeneous matrices) by representing the continuous time trajectory with a Gaussian process. The benefits of using this model are threefold: 1) the interpolation equations are derived from the physically motivated prior that the trajectory is smooth and accelerations are small, 2) the interpolation model operates in the $\mathfrak{se}(3)$ algebra, avoiding the issue of singularities inherent to other 6-DOF pose representations, and 3) the STEAM model allows principled estimation of the uncertainty on interpolated poses [46, 47]. The third point is particularly important for our purposes as the pose uncertainty affects the weight given to a lidar point's RQE contribution. Lidar points captured from very uncertain poses will be given less weight in the entropy calculation, and therefore have less effect

on the optimization.

Use of the STEAM model requires each camera pose to have an associated body-centric velocity variable. The authors of [45, 46, 47] formulate the error metric for poses and velocities as

$$\mathbf{e}_i = \begin{bmatrix} \ln(\mathbf{T}_{i+1}\mathbf{T}_i^{-1})^\vee - (t_{i+1} - t_i)\boldsymbol{\varpi}_i \\ \boldsymbol{\mathcal{J}}(\ln(\mathbf{T}_{i+1}\mathbf{T}_i^{-1})^\vee)^{-1}\boldsymbol{\varpi}_{i+1} - \boldsymbol{\varpi}_i \end{bmatrix}, \tag{4.2}$$

where $\mathbf{T}_i$ is the i'th measured pose, $\boldsymbol{\varpi}_i$ is its associated 6-DOF velocity, and t_i is the associated measurement time. The matrix logarithm and $^\vee$ operator serve to transform an SE(3) member to the $\mathfrak{se}(3)$ algebra, and $\boldsymbol{\mathcal{J}}$ is the SE(3) left jacobian.

To solve for the velocities, we minimize the weighted least-squares cost function J.

$$J = \frac{1}{2}\sum_{i=1}^{M-1} \mathbf{e}_i{}^T\, \mathbf{Q}_i^{-1}\mathbf{e}_i \tag{4.3}$$

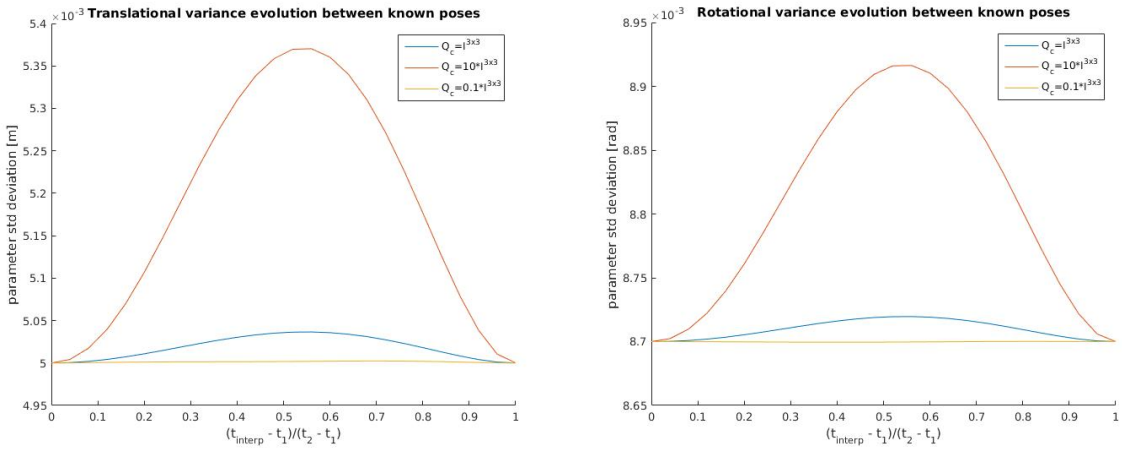
$\mathbf{Q}_i$ serves to weight the error components, and has the following analytic expression, resulting from the kinematics of an object travelling in SE(3) [55],

$$\mathbf{Q}_i = \begin{bmatrix} \frac{1}{3}(t_{i+1} - t_i)^3\mathbf{Q}_c & \frac{1}{2}(t_{i+1} - t_i)^2\mathbf{Q}_c \\ \frac{1}{2}(t_{i+1} - t_i)^2\mathbf{Q}_c & (t_{i+1} - t_i)\mathbf{Q}_c \end{bmatrix}, \tag{4.4}$$

where $\mathbf{Q}_c$ represents the power spectral density of the white noise on the acceleration. $\mathbf{Q}_c$ weights how strictly to hold to the assumption that accelerations are small.

We assume that the camera poses estimated by ORB-SLAM are highly accurate, up to the scale factor. Therefore, unlike [45, 46, 47], we optimize only the velocities, and not the poses. This makes the optimization a linear least-squares problem (the $\mathbf{e}_i$ terms are linear in the $\boldsymbol{\varpi}_i$), and we can directly solve for the minimum of the cost function. Therefore, our choice of $\mathbf{Q}_c$ has no effect on the estimated camera velocities. It does however have a small effect on the computed covariance matrices of the velocities, which are used in the calculation of the interpolated pose covariance. This effect is demonstrated in Figure 4.2, which plots the variance of two representative interpolated pose parameters. In all cases, the variance is greatest in the middle, when we are furthest from the measured poses. $\mathbf{Q}_c$ affects the degree to which the variance grows. We tune $\mathbf{Q}_c$ by finding a value which suitably captures the variation of the estimated measurement time velocities from a simulated ground truth.

With the STEAM model, the interpolation between two poses $\mathbf{T}_i$ and $\mathbf{T}_{i+1}$ is formulated to estimate a local $\mathfrak{se}(3)$ perturbation on the first pose. The interpolation thus



(a) (b)

Figure 4.2: Effect of the matrix $\mathbf{Q}_c$ on uncertainty of interpolated pose parameters *a) x b) ϕ.*

consists of three steps: 1) transforming variables into local perturbations on $\mathbf{T}_i$, 2) performing the interpolation, and 3) transforming the result back into a global variable.

$$\boldsymbol{\gamma}_i(t_i) = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\varpi}_i \end{bmatrix}, \quad \boldsymbol{\gamma}_i(t_{i+1}) = \begin{bmatrix} \ln(\mathbf{T}_{i+1}\mathbf{T}_i^{-1})^\vee \\ \boldsymbol{\mathcal{J}}(\ln(\mathbf{T}_{i+1}\mathbf{T}_i^{-1})^\vee)^{-1}\boldsymbol{\varpi}_{i+1} \end{bmatrix}, \tag{4.5}$$

$$\begin{bmatrix} \boldsymbol{\vartheta}_i(t) \\ \boldsymbol{\varpi}(t) \end{bmatrix} = \boldsymbol{\Lambda}(t)\boldsymbol{\gamma}_i(t_i) + \boldsymbol{\Psi}(t)\boldsymbol{\gamma}_i(t_{i+1}), \quad t_i \leq t \leq t_{i+1}, \tag{4.6}$$

$$\hat{\mathbf{T}}(t) = \exp(\boldsymbol{\vartheta}_i(t)^\wedge)\mathbf{T}_i. \tag{4.7}$$

The subscript $_i$ on the $\boldsymbol{\gamma}$ and $\boldsymbol{\vartheta}$ variables denotes that these are local $\mathfrak{se}(3)$ perturbations relative to pose $\mathbf{T}_i$. $\boldsymbol{\Psi}(t)$ and $\boldsymbol{\Lambda}(t)$ are state transition matrices which have the following form [55],

$$T = t_{i+1} - t_i, \quad r = \frac{t - t_i}{T},$$

$$\boldsymbol{\Psi}(t) = \begin{bmatrix} (3r^2 - 2r^3)\mathbf{I}_6 & (r^3T - r^2T)\mathbf{I}_6 \\ \frac{1}{T}(6r - 6r^2)\mathbf{I}_6 & (3r^2 - 2r)\mathbf{I}_6 \end{bmatrix}, \tag{4.8}$$

$$\boldsymbol{\Lambda}(t) = \begin{bmatrix} (2r^3 - 3r^2 + 1)\mathbf{I}_6 & (rT + r^3T - 2r^2T)\mathbf{I}_6 \\ \frac{1}{T}(6r^2 - 6r)\mathbf{I}_6 & (1 + 3r^2 - 4r)\mathbf{I}_6 \end{bmatrix}. \tag{4.9}$$

To compute a covariance on the interpolated pose, we follow a similar pattern: transforming global pose covariances into covariances of the local perturbations, performing the interpolation, and transforming the interpolated value back to a global representation.

Let $\mathbf{P}(t_i, t_i)$ and $\mathbf{P}(t_{i+1}, t_{i+1})$ be the global self-covariances of poses $\mathbf{T}_i$ and $\mathbf{T}_{i+1}$. These are available either from simulation or from the visual odometry routine. Let $\mathbf{P}_i(t', t'')$ be an arbitrary self (if $t' = t''$) or cross (if $t' \neq t''$) covariance of the local $\mathfrak{se}(3)$ perturbations to $\mathbf{T}_i$. These must be computed as follows [47],

$$\mathbf{P}_i(t_i, t_i) = \mathbf{\Gamma}(t_i)(\mathbf{P}(t_i, t_i) - \mathbf{\Omega}(t_i)\mathbf{P}(t_i, t_i)\mathbf{\Omega}(t_i)^T)\mathbf{\Gamma}(t_i)^T, \tag{4.10}$$

$$\mathbf{P}_i(t_{i+1}, t_{i+1}) = \mathbf{\Gamma}(t_{i+1})(\mathbf{P}(t_{i+1}, t_{i+1}) - \mathbf{\Omega}(t_{i+1})\mathbf{P}(t_i, t_i)\mathbf{\Omega}(t_{i+1})^T)\mathbf{\Gamma}(t_{i+1})^T, \tag{4.11}$$

$$\mathbf{P}_i(t_i, t_{i+1}) = \mathbf{P}_i(t_i, t_i)\begin{bmatrix} \mathbf{I}_6 & \mathbf{0} \\ T\mathbf{I}_6 & \mathbf{I}_6 \end{bmatrix}, \tag{4.12}$$

$$\mathbf{P}_i(t_{i+1}, t_i) = \mathbf{P}_i(t_i, t_{i+1})^T, \tag{4.13}$$

where

$$\mathbf{\Gamma}(t) = \begin{bmatrix} \boldsymbol{\mathcal{J}}(\ln(\hat{\mathbf{T}}(t)\mathbf{T}_i^{-1})^\vee)^{-1} & \mathbf{0} \\ \frac{1}{2}\boldsymbol{\varpi}(t)^\curlywedge \boldsymbol{\mathcal{J}}(\ln(\hat{\mathbf{T}}(t)\mathbf{T}_i^{-1})^\vee)^{-1} & \boldsymbol{\mathcal{J}}(\ln(\hat{\mathbf{T}}(t)\mathbf{T}_i^{-1})^\vee)^{-1} \end{bmatrix}, \tag{4.14}$$

$$\mathbf{\Omega}(t) = \begin{bmatrix} \mathrm{Ad}\left(\hat{\mathbf{T}}(t)\mathbf{T}_i^{-1}\right) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{4.15}$$

The forms of the $^\curlywedge$, $\boldsymbol{\mathcal{J}}()$, and $\mathrm{Ad}\,()$ operators are given in Appendix A.3

To get the local interpolated covariance [47]:

$$\mathbf{P}_i(t, t) = \begin{bmatrix} \mathbf{\Lambda}(t) & \mathbf{\Psi}(t) \end{bmatrix} \begin{bmatrix} \mathbf{P}_i(t_i, t_i) & \mathbf{P}_i(t_i, t_{i+1}) \\ \mathbf{P}_i(t_{i+1}, t_i) & \mathbf{P}_i(t_{i+1}, t_{i+1}) \end{bmatrix} \begin{bmatrix} \mathbf{\Lambda}(t)^T \\ \mathbf{\Psi}(t)^T \end{bmatrix} + \boldsymbol{\mathcal{Q}}_i(t), \tag{4.16}$$

where

$$\boldsymbol{\mathcal{Q}}_i(t) = \mathbf{Q}_i(t) - \mathbf{Q}_i(t)\begin{bmatrix} \mathbf{I}_6 & \mathbf{0} \\ (t_{i+1} - t)\mathbf{I}_6 & \mathbf{I}_6 \end{bmatrix} \mathbf{Q}_i(t_{i+1})^{-T} \begin{bmatrix} \mathbf{I}_6 & (t_{i+1} - t)\mathbf{I}_6 \\ \mathbf{0} & \mathbf{I}_6 \end{bmatrix} \mathbf{Q}_i(t)^T, \tag{4.17}$$

and $\mathbf{Q}_i(t')$ is a generalization of Equation (4.4),

$$\mathbf{Q}_i(t') = \begin{bmatrix} \frac{1}{3}(t' - t_i)^3\mathbf{Q}_c & \frac{1}{2}(t' - t_i)^2\mathbf{Q}_c \\ \frac{1}{2}(t' - t_i)^2\mathbf{Q}_c & (t' - t_i)\mathbf{Q}_c \end{bmatrix}. \tag{4.18}$$

To get the global interpolated pose covariance [47]

$$\mathbf{P}(t, t) = \mathbf{\Gamma}(t)^{-1}\mathbf{P}_i(t, t)\mathbf{\Gamma}(t)^{-T} + \mathbf{\Omega}(t)\mathbf{P}(t_i, t_i)\mathbf{\Omega}(t)^T. \tag{4.19}$$

4.2 Entropy Computation Approximation

With a continuous-time representation of the camera trajectory, and an estimate of the calibration parameters, we register the set of lidar scans to the camera trajectory as outlined in Section 3.1.1. This results in a point cloud of $P = KN$ 3D points (where, as before, K is the number of points per lidar scan, and N is the total number of lidar scans in the dataset). Evaluating the analytic expression for RQE (Equation (3.23)) requires $O(P^2)$ evaluations of a Gaussian distribution function for a point cloud of P points. Evaluating the value of a Gaussian is an expensive operation. Furthermore, while there are many algorithms designed to efficiently compute values from a Gaussian distribution for a large number of inputs [56, 57], they require the distribution to have the same covariance matrix for every computation. In our case, the covariance will change with both indices i and j.

However, we are able to apply the same simplifications to the cost function as in [28, 7]. We will recount those here and provide motivation for each step. In addition, we make a novel simplification based on the contribution of lidar points in the same scan.

As both indices i and j loop over every point in the point cloud, we will evaluate the distance between every pair of points twice. This is redundant since the distance and covariance values will remain the same regardless of the direction we are evaluating. It is helpful to imagine the lidar points as nodes on a graph, and the distances between them as edges. There are $\binom{P}{2} = \frac{P(P-1)}{2}$ unique edges we wish to evaluate. The original formulation evaluates each edge twice, $P(P-1)$ computations. Ultimately, we can ignore duplicate edges, and simply double the contribution of each edge to arrive at the same entropy value.

$$H(\hat{\mathbf{X}}) = -\log\left(\frac{2}{P^2}\sum_{i=1}^{P}\sum_{j=i+1}^{P}\mathcal{N}(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j; \mathbf{0}, \boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j + 2\sigma^2\mathbf{I})\right) \tag{4.20}$$

Given that the logarithm is a monotonic function and the $\frac{2}{P^2}$ term is constant, we can remove them from the cost function $f(\hat{\boldsymbol{\Xi}})$. Note that we have changed notation from $H(\hat{\mathbf{X}})$ to $f(\hat{\boldsymbol{\Xi}})$ to denote that this does not evaluate to the same value as Equation (3.23), but will be minimized at the same point.

$$f(\hat{\boldsymbol{\Xi}}) = -\sum_{i=1}^{P}\sum_{j=i+1}^{P}\mathcal{N}\left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j, \boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j + \sigma^2\mathbf{I}\right) \tag{4.21}$$

Next, we include the novel simplification of not including distances between points captured as part of the same lidar scan. All points within a single lidar scan are assumed

to be captured at the same instant, and are registered to the same place in the lidar trajectory. There is in fact a non-zero delay between the capture of each point, but it is small enough (e.g., 24 ms per 1081 point scan on the Hokuyo lidar) that it can be ignored unless the lidar is moving fast enough to have a noticeable displacement in that time. The spatial and scale parameters affect the shape of the lidar trajectory, and the time-delay parameter affects where in the trajectory each scan is registered to. None of the parameters affect the distances between points within a scan, so these distances can be ignored for the purposes of our optimization.

Let $S(p)$ be the function that takes the index of a lidar point and returns the index of the scan from which it is captured. We modify the cost function as follows,

$$f(\hat{\boldsymbol{\Xi}}) = -\sum_{i=1}^{P} \sum_{\substack{j=i+1 \\ S(j)\neq S(i)}}^{P} \mathcal{N}\left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j, \boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j + \sigma^2\mathbf{I}\right). \tag{4.22}$$

We have found that this typically reduces the total number of Gaussian evaluations by $3-4$ %.

Lastly, following [28, 7], it is possible to speed up the nominally $\mathrm{O}(P^2)$ RQE computation by ignoring points that contribute negligible entropy. Introducing a tuning parameter, k, one can then make the approximation

$$\begin{aligned} &\mathcal{N}\left(\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j, \boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j + \sigma^2\mathbf{I}\right) \approx 0 \ \text{ if} \\ &\|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\| \geq 2k(\max(\lambda_1(\boldsymbol{\Sigma}_i), \lambda_1(\boldsymbol{\Sigma}_j)) + \sigma^2), \end{aligned} \tag{4.23}$$

where $\lambda_1(\boldsymbol{\Sigma})$ is the largest eigenvalue of matrix $\boldsymbol{\Sigma}$. The intuition here is that the contribution of any point pair to the entropy calculation is the exponential of the squared Mahalanobis distance between them, and as such, drops off exponentially. The most relevant point pairs then, are those that are close together relative to the uncertainty in their positions.

Smaller k values will reduce computation time at the expense of accuracy. However, increasing k yields diminishing returns in accuracy with respect to the total number of Gaussian computations that must be performed. This effect is displayed in Figure 4.3. We can obtain highly accurate entropy values by considering only a small subset of point-to-point distances. For example, in Figure 4.3, a k value of 3 corresponds to 3.3×10^8 Gaussian computations out of a potential total of 5.1×10^{12}. We determine which points to consider at any given k-value by constructing a k-d tree to efficiently determine which point pairs do and do not have a significant impact on the RQE calculation [28].

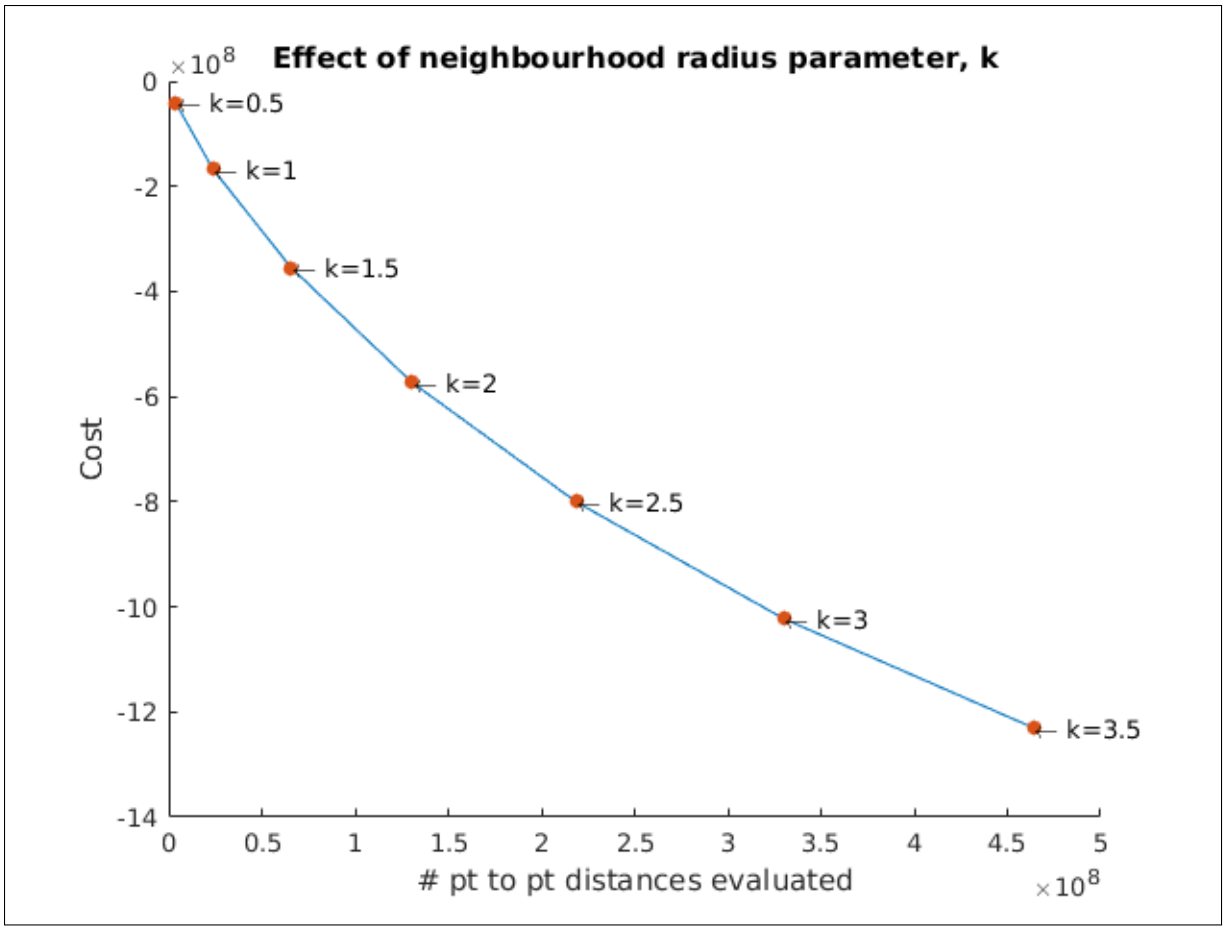


Figure 4.3: Relationship between cost function value and the number of Gaussian computations performed. There are diminishing returns on the improved RQE accuracy obtained by considering a larger neighbourhood around each point.

Criterion 4.23 makes the computation of the RQE tractable on a CPU, and is therefore critical to our calibration routine. However, as we will discuss in Section 4.3.2, this simplification results in non-smooth behaviour of the cost function over small regions in the parameter space. This will motivate our use of Bayesian optimization, as we will discuss in Section 4.3.3.

4.3 Optimization

4.3.1 Cost Function Validation

It was found in [7, 29] that the calibration can be formulated as a 7 degree of freedom optimization problem. The cost function displays a single global minimum at the correct calibration (6 parameters) and scale (1 parameter). The macro scale cost function

behaviour we present below in Section 4.3.2 confirms this finding.

To accommodate for time delay as an eighth calibration parameter, it is worth asking whether or not it has the same effect on entropy as the other parameters. That is, will an incorrect time delay estimate increase point cloud entropy? Moreover, will it display the nearly convex behaviour we desire to make it amenable to standard optimization methods? Intuition would suggest that the answer to the first question is yes. Much like with spatial calibration, incorrect time delay estimates will have a different effect on each lidar scan. At different times in the data collection, the sensor platform will be moving different directions and at varying speeds. A small time delay misestimate will have a larger effect on lidar scans captured when the system is moving very fast. We should expect the point cloud to be most structured when we have correctly estimated the time delay.

Figure 4.4 shows that our assumption is correct, and also demonstrates that the entropy is nearly convex with respect to the time delay. This graph was created using a simulated dataset, by evaluating the entropy of the point cloud at a series of time delay values around the ground truth of 0.02 s. All other spatial and scale parameters were held constant at their ground truth values when performing this line search. It is worth noting that although this line search shows a minimum to the one-dimensional time delay problem, this does not confirm that there is a global minimum to the 8DOF problem at the ground truth parameters. However, as we present in Chapter 5, the optimization does converge to such a global minimum at or very near the true calibration parameters.

4.3.2 Complications of Gradient-Based Optimization

To solve for the calibration parameters, we must optimize Equation (4.22). One would expect this equation to have continuous analytical derivatives with respect to the calibration parameters, since it is ultimately the sum of a set values of Gaussian distributions. The inputs, for which we compute the Gaussian distributions values, are distances between lidar points, which should be expected to change smoothly with the calibration parameters. Indeed, the authors of [28] claim that the cost function can be optimized by Newton's method or similar gradient-based methods. However, we have found that, over small regions in the parameter space (e.g., 10^{-4} m or radians) the local behaviour of the cost function is jagged and unpredictable (Figure 4.5). This is a result of the simplification in Equation 4.23, where different subsets of points are selected on each iteration; very small changes to the parameters affect each lidar point differently, causing variations in those points that meet or fail the criterion in Equation 4.23.

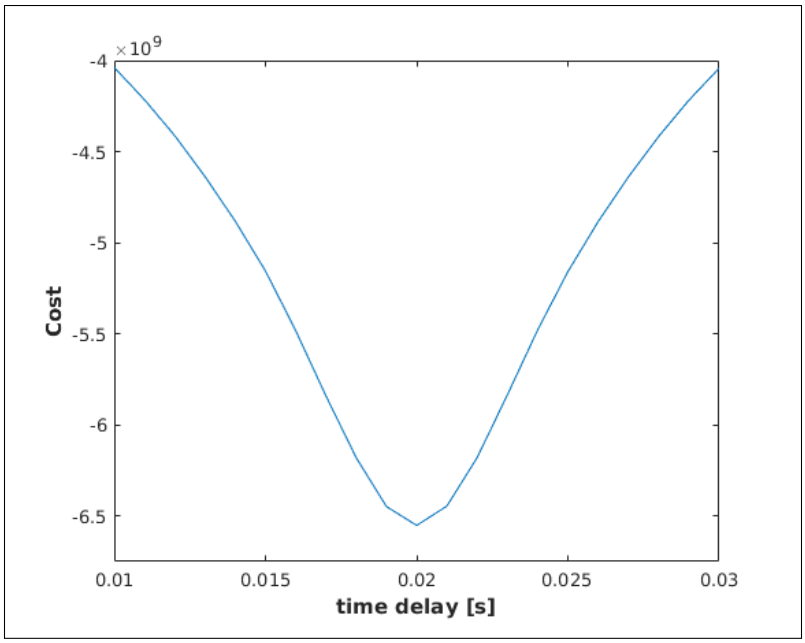


Figure 4.4: Effect of the time delay parameter on the entropy of the point cloud. The ground truth time delay is 0.02 s. The spatial calibration and trajectory scale parameters were held constant at their ground truth values when plotting this graph.

We confirmed that the jagged behaviour is caused by the changes in point selection by evaluating cost function for a specific set of lidar points, not making use of criterion 4.23. To ensure computational tractability, we tested a much smaller set of points than we would use for calibration. We stored the indices of the lidar point pairs that pass criterion 4.23 when the point cloud is constructed using the ground truth parameter values. We then use the same set of point pairs when evaluating the entropy elsewhere in the state space. We found that the cost function is smooth under these conditions. The results are shown in Figure 4.6. The cost function is now totally smooth even at the micro scale (the jumps in entropy value shown in Figure 4.6a, 4.6b, and 4.6c are simply the result of the limited precision used for computing cost function values).

This begs the question, why not always evaluate the cost with a small, constant set of point pairs? The major roadblocks to doing so are that 1) evaluating every set of point pairs in the point cloud is computationally intractable, and 2) there is not currently an informed method of selecting which subset of point pairs will provide enough information to converge to the correct calibration parameters. In Figure 4.6, we initialized the set of point pairs by finding those that pass criterion 4.23 when the point cloud is constructed with the correct calibration parameters. In other words, from that point forward, we are trying to bring point pairs close together that were originally close together when constructed with the correct parameters. It is no surprise then that all of the parameters

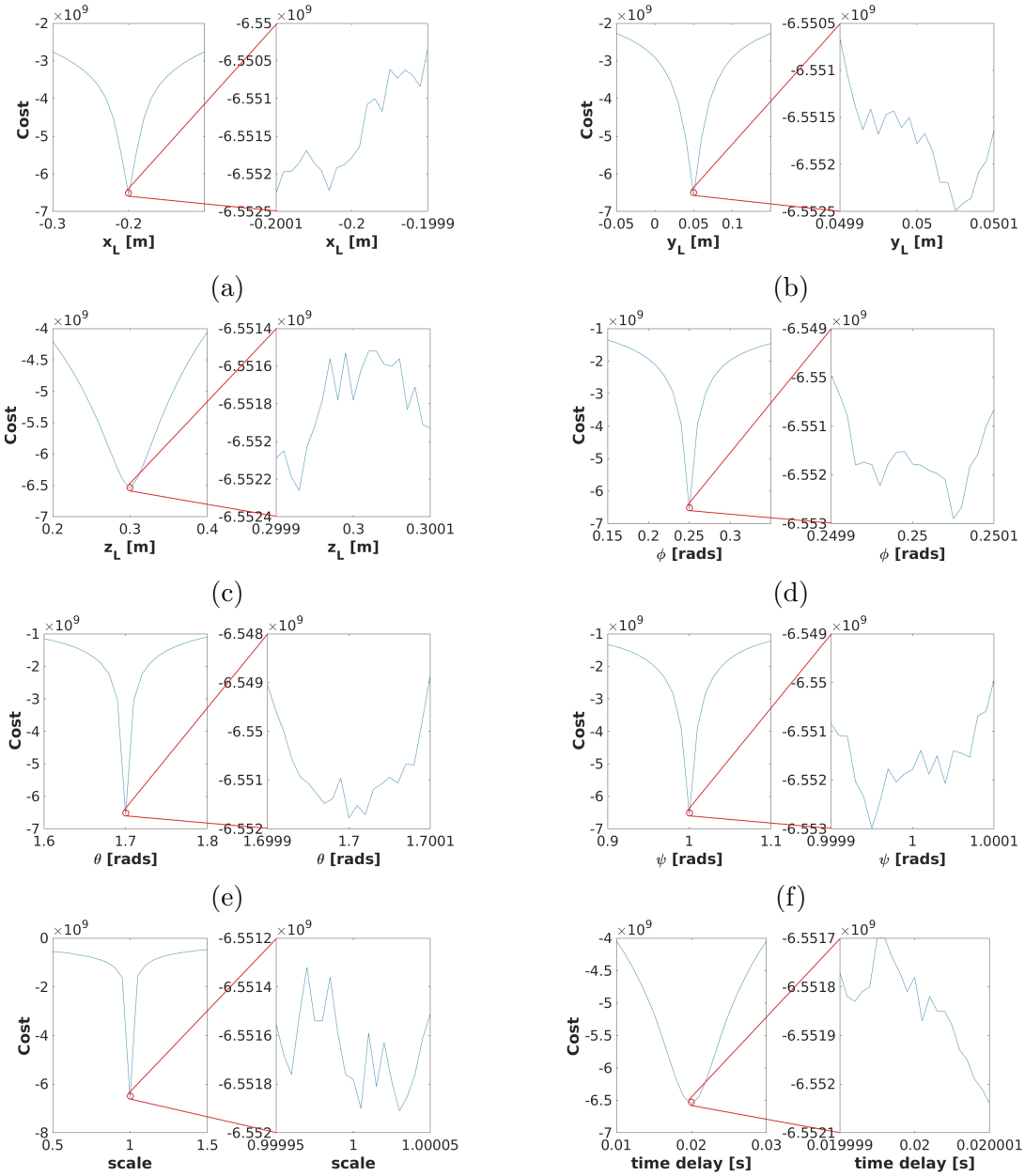


Figure 4.5: Macro and micro scale behaviour of entropy for each calibration parameter. In each plot, all seven other parameters are held constant at ground truth values. *a)* x_L *b)* y_L *c)* z_L *d)* ϕ_L *e)* θ_L *f)* ψ_L *g)* scale *h)* time delay

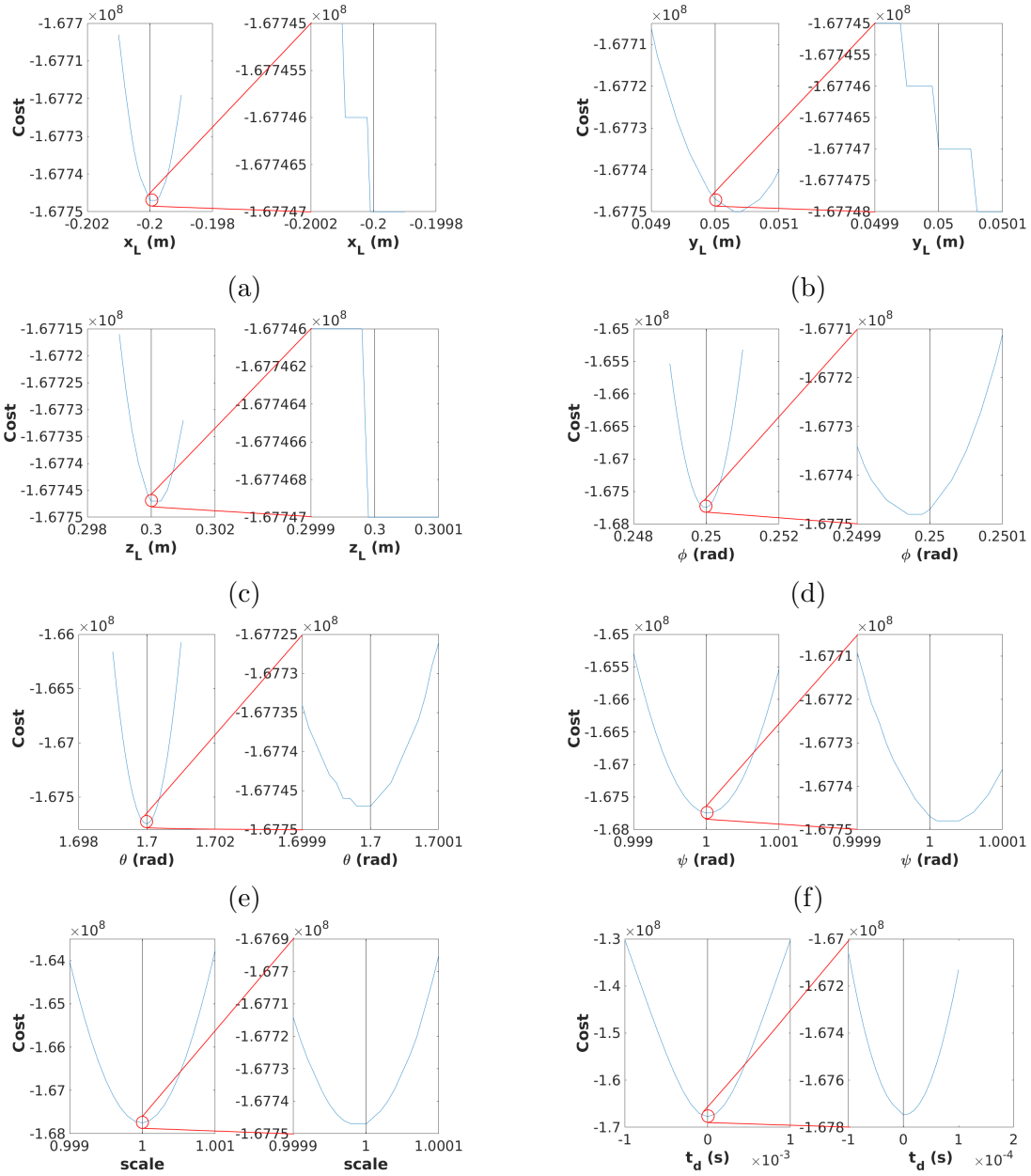


Figure 4.6: Macro and micro scale behaviour of entropy for each calibration parameter when the set of lidar points is held constant at each cost function evaluation. *a)* x_L *b)* y_L *c)* z_L *d)* ϕ_L *e)* θ_L *f)* ψ_L *g)* scale *h)* time delay

converge to (close to) ground truth. However, in the real world, we do not know what the true calibration parameters are, so the best we can do is to initialize the point pairs with some set of calibration parameters that is hopefully close to ground truth.

However, when we do so, we change the behaviour of the cost function such that it converges to the initialization point. As before, it is attempting to bring points together that were brought close together by the initialization parameters. This effect is demonstrated in Figure 4.7. In this example, we initialized the set of point pairs by selecting those that pass criterion 4.23 when the calibration parameters are roughly one quarter of the way from the ground truth parameter values toward their negative bounds. As can be seen, each of the parameters now converges to some point with a lower value than ground truth. The exception is x_L, which converges to a point with higher value than ground truth. The problem remains however that the convergence point has shifted away from ground truth.

Ultimately, we find it is best to allow the set of point pairs used for entropy computation to change throughout the state space, as selected by criterion 4.23. There are methods for dealing with the micro scale jaggedness it causes in the cost function. We discuss our approach for doing so herein.

As a result of the cost function jaggedness, the behaviour of the gradient is unpredictable, and will not perform well for gradient-based minimization techniques. It is clear from Figure 4.5 though, that the cost function can be approximated by a smooth curve over large enough regions in the parameter space.

4.3.3 Bayesian Optimization

Despite local jaggedness in the cost function, it can be well approximated at the macro scale by a smooth curve. Bayesian Optimization [58] is an algorithm for efficiently optimizing functions that are a) expensive to evaluate (as is true of RQE), and b) noisy. We note that the RQE is not noisy in the same sense as many other problems Bayesian Optimization is applied to. Unlike these functions, RQE is deterministic and will always return the same value if repeatedly evaluated with the same inputs. However, the micro-scale jaggedness of RQE is very similar to the effect displayed by a non-deterministic cost function with low amplitude noise corrupting some smooth behaviour.

Bayesian Optimization works by sampling the state space to build up a surrogate model of the cost function as a Gaussian Process. This parametric approximation to the cost function ignores the "noise" in the true cost function. The program can therefore quickly find the minimum of the surrogate model. If the surrogate model is accurate

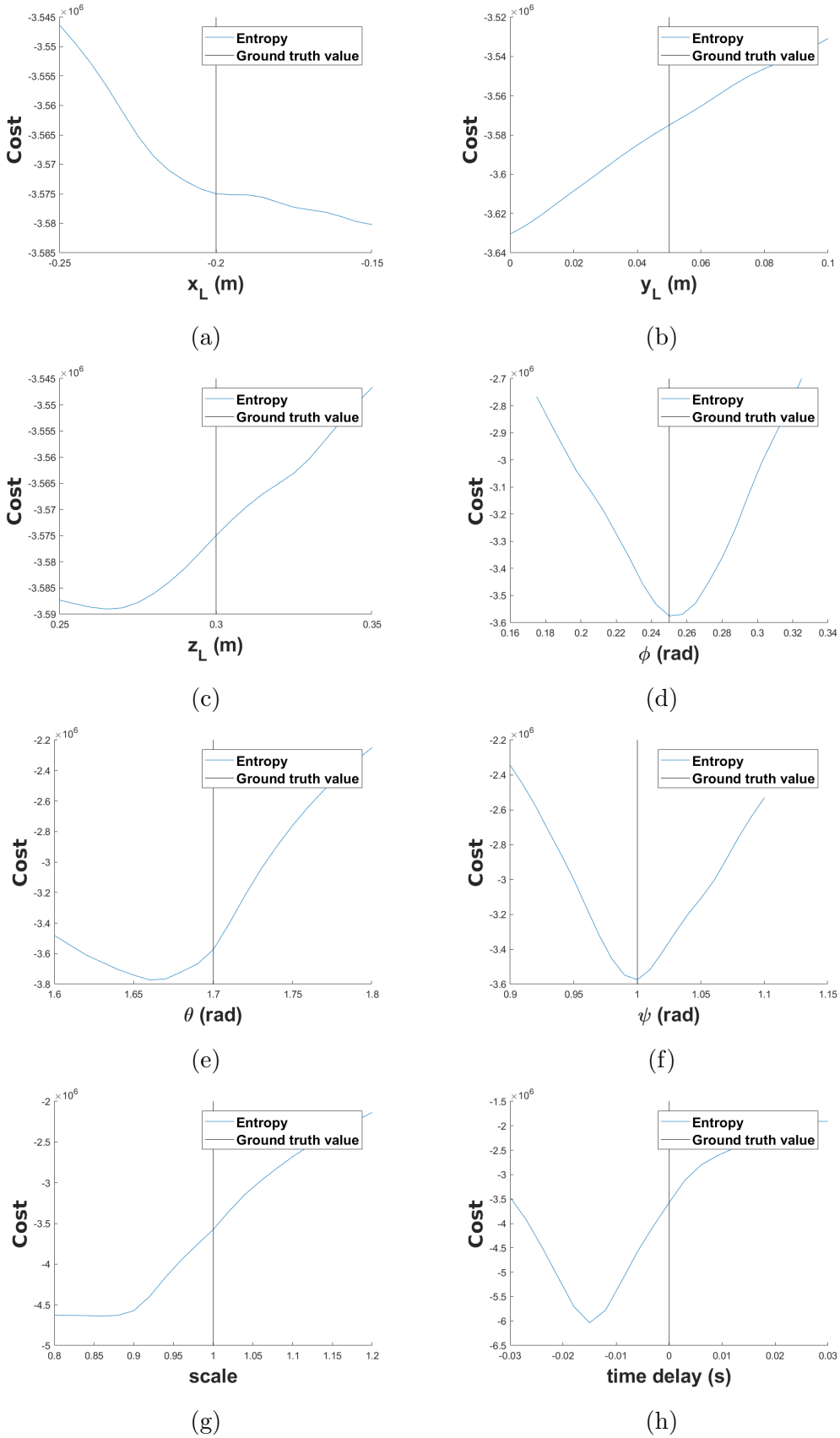


Figure 4.7: Macro scale behaviour of entropy for each calibration parameter, when a constant set of lidar points is chosen by initializing at a point other than the ground truth parameters. *a)* x_L *b)* y_L *c)* z_L *d)* ϕ_L *e)* θ_L *f)* ψ_L *g)* scale *h)* time delay

enough, this point will approximately coincide with the minimum of the real cost function.

The optimizer seeks to minimize the number of cost function evaluations by balancing exploitation (evaluating the cost function at points it believes to be near the minimum) with exploration (evaluating the cost function in other regions to build up a more accurate surrogate model). The optimizer achieves this balance by computing an expected improvement score. For an input $\boldsymbol{\Xi}$ which has not yet been sampled, let us denote that there is uncertainty about the value Y of the cost function at this point.

$$Y(\boldsymbol{\Xi}) = \mathcal{N}\left(\hat{y}, \sigma^2\right) \tag{4.24}$$

Letting y^* be the minimum value of the cost function obtained so far, then the potential improvement I at $\boldsymbol{\Xi}$ is a random variable [58],

$$I(\boldsymbol{\Xi}) = \max(y^* - Y(\boldsymbol{\Xi}), 0). \tag{4.25}$$

From [59], the expected value of the improvement is

$$\mathbb{E}\left[I(\boldsymbol{\Xi})\right] = (y^* - \hat{y})\boldsymbol{\Phi}(\frac{y^* - \hat{y}}{\sigma}) + \sigma\boldsymbol{\phi}(\frac{y^* - \hat{y}}{\sigma}), \tag{4.26}$$

where $\boldsymbol{\Phi}$ and $\boldsymbol{\phi}$ are the standard normal density and distribution functions respectively. The Gaussian process model gives us a value of $\hat{y}$ and σ for all inputs $\boldsymbol{\Xi}$. Therefore, with user-supplied lower and upper bounds for $\boldsymbol{\Xi}$, Equation (4.26) can be maximized with respect to $\boldsymbol{\Xi}$ with branch and bound algorithms [60].

From Equation (4.26), one can note that expected improvement will be highest when either a) there is a high likelihood that $Y(\boldsymbol{\Xi}) < y^*$, or b) the value of $Y(\boldsymbol{\Xi})$ is highly uncertain (large σ). Thus, it is a useful score for balancing exploitation and exploration. Using Bayesian Optimization allows us to achieve an eightfold or greater speedup in computation when compared with [7] ($\sim$250 iterations as opposed to $\sim$2000).

4.4 Implementation

The calibration routine is written in C++. We use `nanoflann` [61] to efficiently store and query the point cloud as a k-d tree. We use `Eigen3` [62] for efficient matrix operations. We use `BayesOpt`[63] to implement Bayesian Optimization.

Algorithm 1 describes the entire optimization process, beginning with a set of images, lidar scans, and user-supplied bounds for the calibration parameters. We use ORB-SLAM to obtain the set of camera poses $\left[\mathbf{y}_1, \ldots, \mathbf{y}_M\right]$, their associated $\mathfrak{se}(3)$ space velocities

Algorithm 1 BayesianRQEOptimization

Data In: $\{Images\}$, $\mathbf{Z}$, $\mathbf{\Xi}_{init}$, $\mathbf{\Xi}_{lb}$, $\mathbf{\Xi}_{ub}$
$\mathbf{Y}$ = ORBslam($\{Images\}$)
Set: $\mathbf{\Xi} = \mathbf{\Xi}_{init}, N_{sample} = 51, N_{iter} = 250$
function BayesianOptimizationRoutine($\mathbf{\Xi}_{init}, \mathbf{\Xi}_{lb}, \mathbf{\Xi}_{ub}$)
    CostFunction = PointCloudRQEApprox
    Set: $y^* = \infty$
    for $i = 1, ..., N_{sample}$ **do**
        $\mathbf{\Xi}_i$ = LatinHypercubeSampling($i, N_{sample}, \mathbf{\Xi}_{lb}, \mathbf{\Xi}_{ub}$)
        y_i = CostFunction($\mathbf{Y}$, $\mathbf{Z}$, $\mathbf{\Xi}_i$)
        BayesOpt::AddToSurrogateModel($\mathbf{\Xi}_i$,y_i)
        if $y_i < y^*$ **then**
            $y^* = y_i$, $\mathbf{\Xi}^* = \mathbf{\Xi}_i$
        end if
    end for
    for $i = 1, ..., N_{iter}$ **do**
        $\mathbf{\Xi}_i$ = BayesOpt::MaximizeExpectedImprovement(y^*)
        y_i = CostFunction($\mathbf{\Xi}_i$)
        BayesOpt::AddToSurrogateModel($\mathbf{\Xi}_i$,y_i)
        if $y_i < y^*$ **then**
            $y^* = y_i$, $\mathbf{\Xi}^* = \mathbf{\Xi}_i$
        end if
    end for
    return $\mathbf{\Xi}^*$
end function

Algorithm 2 PointCloudRQEApprox

Data In: $\mathbf{Y},\ \mathbf{Z},\ \boldsymbol{\Xi}$
Parameters: $k = 2, \sigma = 0.01$
Set: $H = 0$
function TransformPointCloud($\mathbf{Y}, \mathbf{Z},\ \boldsymbol{\Xi}$)
    $\mathbf{T}_{C,L} \leftarrow$ ConvertToTMatrix($\boldsymbol{\Xi}[0:5]$)
    $s \leftarrow \boldsymbol{\Xi}[6]$
    $t_d \leftarrow \boldsymbol{\Xi}[7]$
    for $n = 1 : N$ **do**
        $[m, m+1] \leftarrow$ FindBoundingCameraPoses($t_n + t_d$)
        $\hat{\mathbf{T}}_{G,C_n} \leftarrow$ InterpPose($t_n + t_d, \mathbf{y}_m, \boldsymbol{\varpi}_m, t_m, \mathbf{y}_{m+1}, \boldsymbol{\varpi}_{m+1}, t_{m+1}$)
        $\mathbf{Q}_n \leftarrow$ InterpCovariance($t_n + t_d, \mathbf{y}_m, \boldsymbol{\varpi}_m, \mathbf{Q}_m, t_m, \mathbf{y}_{m+1}, \boldsymbol{\varpi}_{m+1}, \mathbf{Q}_{m+1}, t_{m+1}$)
        $\hat{\mathbf{T}}_{G,C_n}[0:2,\ 3] = s\hat{\mathbf{T}}_{G,C_n}[0:2,\ 3]$
        $\mathbf{Q}_n[0:2,\ 0:2] = s^2\mathbf{Q}_n[0:2,\ 0:2]$
        for $k = 1 : K$ **do**
            $\hat{\mathbf{X}} \leftarrow \hat{\mathbf{p}}_{G,n}^{(k)} = \mathbf{T}_{G,C_n}\mathbf{T}_{C,L}\mathbf{p}_{L_n}^{(k)}$
            $\mathbf{J}_n^{(k)} = \dfrac{\partial h^{-1}(\mathbf{x}_{L_n}^{(k)} | \mathbf{y}_m, \boldsymbol{\Xi})}{\partial \mathbf{y}_m}$
            $\boldsymbol{\Sigma}_n^{(k)} = \mathbf{J}_n^{(k)}\mathbf{Q}_n{\mathbf{J}_n^{(k)}}^T$
            $\boldsymbol{\lambda} \leftarrow \lambda_n^{(k)} =$ MaxEigenvalue($\boldsymbol{\Sigma}_n^{(k)}$)
            $\hat{\mathbf{X}}, \boldsymbol{\Sigma}, \boldsymbol{\lambda} =$ SortDescending($\boldsymbol{\lambda}$)
        end for
    end for
    return $\hat{\mathbf{X}},\ \boldsymbol{\Sigma},\ \boldsymbol{\lambda}$
end function
function ApproximateRQE($\hat{\mathbf{X}}, \boldsymbol{\Sigma},\ \boldsymbol{\lambda}, k, \sigma$)
    for $i = 1 : P$ **do**
        $R = 2k(\lambda_p + \sigma^2)$
        for all $\{\mathbf{x}_j \in \hat{\mathbf{X}} \mid \|\mathbf{x}_i - \mathbf{x}_j\| \leq R\}$ **do**
            if $\boldsymbol{\lambda}_i > \boldsymbol{\lambda}_j$ **and** ScanIndex($\mathbf{x}_i$) $\neq$ ScanIndex($\mathbf{x}_j$) **then**
                $H \leftarrow H + \mathcal{N}(\mathbf{x}_i - \mathbf{x}_j, \Sigma_i + \Sigma_m + 2\sigma^2\mathbf{I})$
            end if
        end for
    end for
    return H
end function

$\left[\boldsymbol{\varpi}_1, \ldots, \boldsymbol{\varpi}_M\right]$, and associated covariances $\left[\mathbf{Q}_1, \ldots, \mathbf{Q}_M\right]$. Bayesian Optimization begins by sampling the cost function at N_{sample} points in the 8DOF calibration space, arranged evenly throughout the state space. It adds the point-cost pairs to its Gaussian process surrogate model of the function. The algorithm also remembers the optimal calibration vector it finds in the sampling phase, for comparison with future values.

The Bayesian Optimization algorithm then performs N_{iter} iterations probing the cost function. At each iteration, it uses a branch and bound routine to determine the calibration parameter that maximizes expected improvement score (as explained in Section 4.3.3) based on the surrogate model. The algorithm probes the cost function at this point and incorporates the result into its surrogate model. In this way, each successive probing of the cost function is better informed than the one preceding it. Finally, the algorithm returns the calibration parameters resulting in the minimum cost function value it encountered.

In practice, we have found that 50 initial samples and 250 optimization iterations are sufficient to minimize the RQE cost function. However, this is an unscientific way of ensuring convergence. One could replace the set number of iterations with another stopping criteria, such as an absolute cost value threshold, or absolute or relative changes in cost function value from one iteration to the next.

Algorithm 2 presents the cost function, as described in Section 4.2. It takes as input the structure $\mathbf{Y}$ (containing the camera poses, velocities, and associated covariance matrices), the set of lidar scans $\mathbf{Z}$, and a vector of calibration parameters $\boldsymbol{\Xi}$. The first six calibration parameters are used to construct the lidar-camera transformation matrix. The seventh and eighth are used for the scale and time delay parameters respectively.

The point cloud is constructed by iterating over all N lidar scans, and finding the camera poses immediately before and after the lidar measurement instant, accounting for time delay. The interpolated camera pose and covariance at the lidar measurement instant are found using the STEAM model. The translational portions of the interpolated pose and covariance are scaled accordingly.

For all K points in a lidar scan, the point's position in the global coordinate frame is found via the composition of the interpolated camera pose and the lidar to camera transformation. The point's 3×3 covariance matrix is computed using the interpolated camera pose covariance and the Jacobian of the point position with respect to the camera pose. The form of the Jacobian is given in Appendix A.2. Points are sorted and stored in a k-d tree according to the maximum eigenvalue of their covariance matrix.

To compute the RQE, we loop over all P points in the point cloud ($P = KN$) and find all points within a radius R whose value depends on the point's maximum

eigenvalue, and the user supplied parameters k and σ. We then ensure that we are not double counting the entropy between points, and that we are not counting the entropy between points from the same scan. If two points pass this condition, we add to H their entropy contribution, which is the exponential of the squared Mahalanobis distance between them. The resultant value of H after the loop exits is the approximated RQE, and is returned as the cost function value.

Chapter 5

Results

Here we present results demonstrating the accuracy in the calibration parameters estimated by our routine. We first present results from simulated data, demonstrating the ability of the routine to converge to ground truth values specified in the creation of the simulated data. We then present results on real world data. As it is difficult to obtain accurate ground truth calibration parameter values in the real world, we show that the estimated calibration parameter values are repeatable, even with loose bounds for the optimizer's search space.

5.1 Simulation

We simulated a 2D lidar rigidly attached to an egomotion sensor following a smooth trajectory through one of five environments, shown in Figures 5.2-5.5. In the real world, a smooth sensor trajectory is required to ensure the camera maintains accurate tracking with a VO routine. Sharp discontinuities or severe accelerations can cause motion blur, resulting in ambiguity about the location of landmarks, and a loss of tracking [64]. To mimic this, the simulation trajectories were created by setting each parameter of the camera's position ($x\ y\ z\ \phi\ \theta\ \psi$) as a sinusoidal function of time. We are using the roll, pitch, yaw standard of rotations. We use this standard for formulating the simulation trajectory because it is easier to understand and visualize than the $\mathfrak{so}(3)$ rotation vectors we use in the calibration step. As we will discuss in Section 5.1.1, the shape of the sensor trajectory has a significant effect on the calibration results. It is critical to find a set of amplitude and frequency values for each parameter sinusoid that allows for accurate calibration. To introduce some variability into each simulated dataset, we set each amplitude and frequency value with a normal distribution centered on values that are found to work well.

We tested the effect of noisy pose estimates from the egomotion sensor by adding zero-mean uncorrelated Gaussian noise with standard deviation of 5 mm (positions) and 0.5^o (roll, pitch, and yaw angles) to the output poses. Trajectory scale uncertainty was simulated by scaling the positional component of the published poses. Both sensors published at 40 Hz, but were offset in time to simulate the effect of time-delay between sensor clocks. The datasets were 90 seconds long and the lidar published 1081 points per scan in a 270^o FOV for a total of nearly 4 million lidar points per dataset.

The environments were designed to vary in the amount of structure they exhibit, to test the response of the algorithm to less structured targets. For example, the first consists of a set of planes, while the last contains no planes at all. We tested five simulation environments.

1. **Simple Room -** consists of six planes shaped as a rectangular prism. (Figure 5.1)

2. **Plane City -** consists of 11 planes (including floor and ceiling) arranged as a rectangular prism with additional interior planes at irregular angles. (Figure 5.2)

3. **Underground Parking Lot -** consists of a six plane rectangular prism shaped room. In addition, there are six cylindrical "pillars" (such as one might find in a parking garage) and two spherical targets. (Figure 5.3)

4. **Circular Room -** consists of a single cylindrical room with two planes making up the floor and ceiling. (Figure 5.4)

5. **Forest -** is the most irregular environment. It consists of a wavy "ground", and cylindrical "trees" with attached spherical "leaves". The trees and leaves are placed randomly. (Figure 5.5)

The ordering of the above list represents our expectation about the suitability of each environment as a calibration environment. Environments with more structure are expected to be more suitable for calibration. Entropy minimization dictates that lidar points hitting surfaces be clustered together as tightly as possible. Having more surfaces is therefore expected to constrain the optimized calibration transforms. Furthermore, planes are considered the best type of surface to scan for calibration, as they provide more certainty about where the points should be, since they constrain the points' positions along the axis normal to the plane. However, unlike most calibration routines, we will demonstrate that our routine works even if the lidar is mainly scanning non-planar surfaces, such as Circular Room.

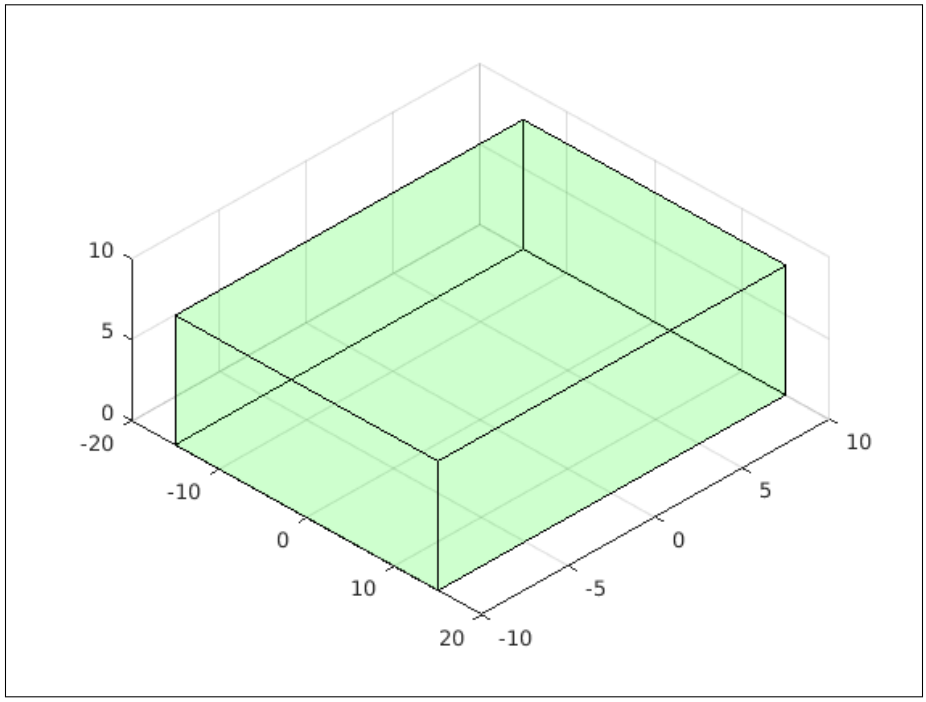


Figure 5.1: Simple Room simulation environment.

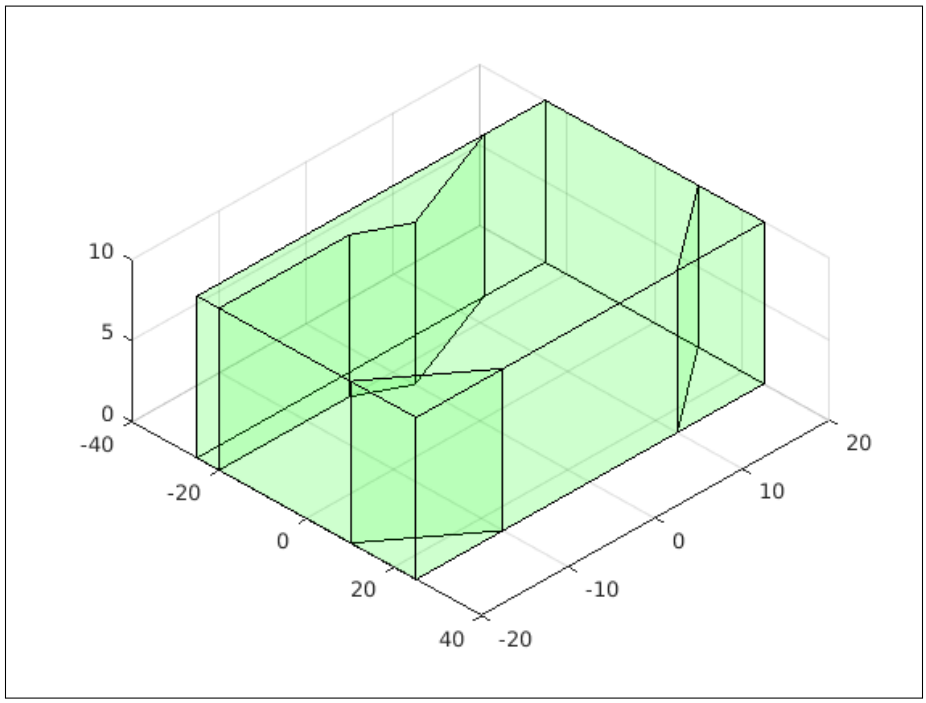


Figure 5.2: Plane City simulation environment.

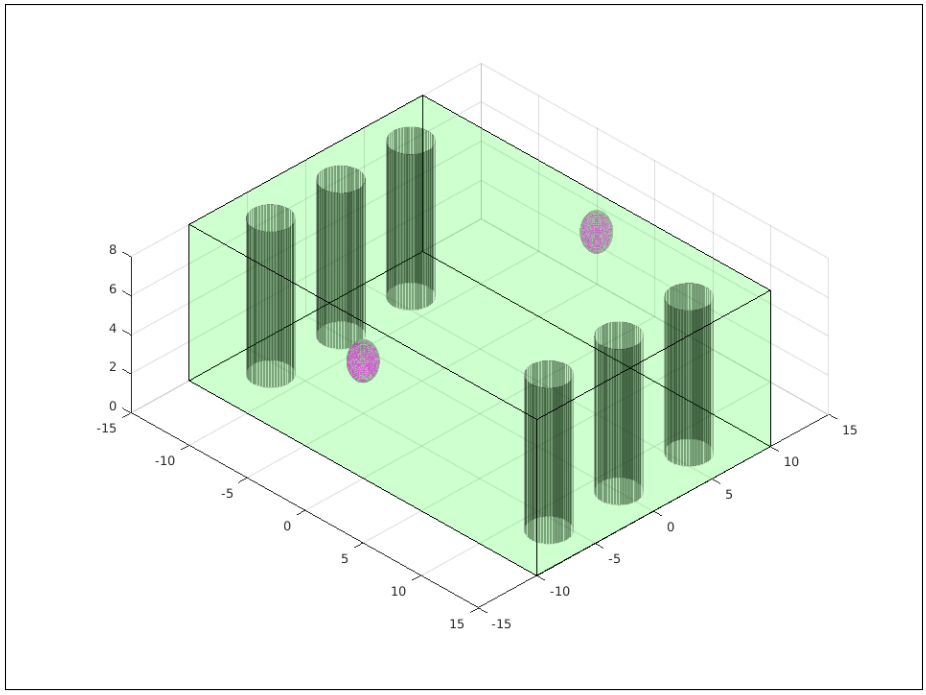


Figure 5.3: Underground Parking Lot simulation environment.

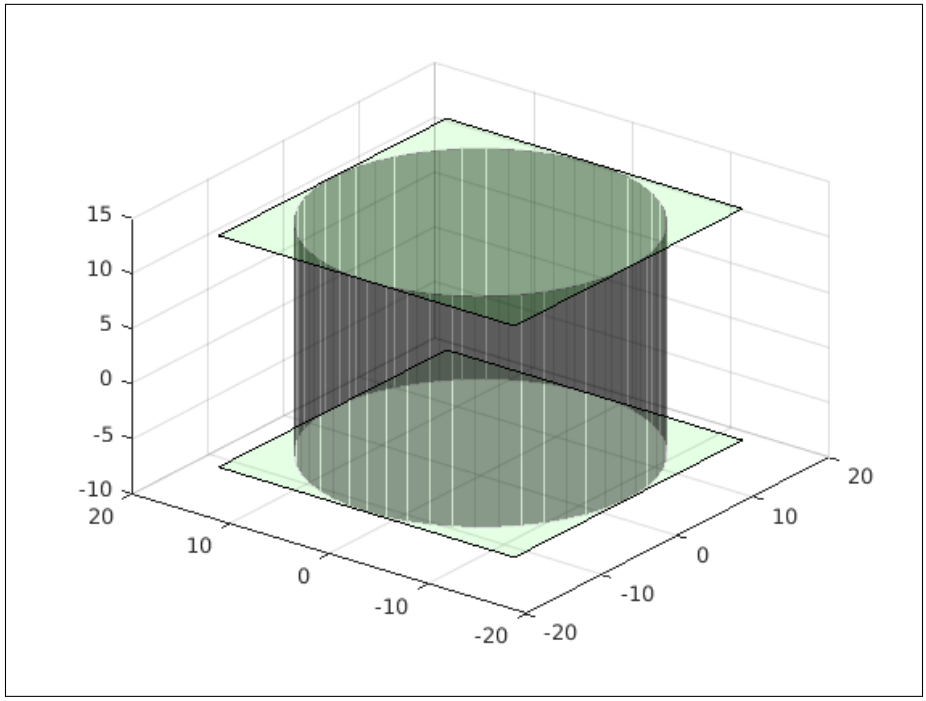


Figure 5.4: Circular Room simulation environment.

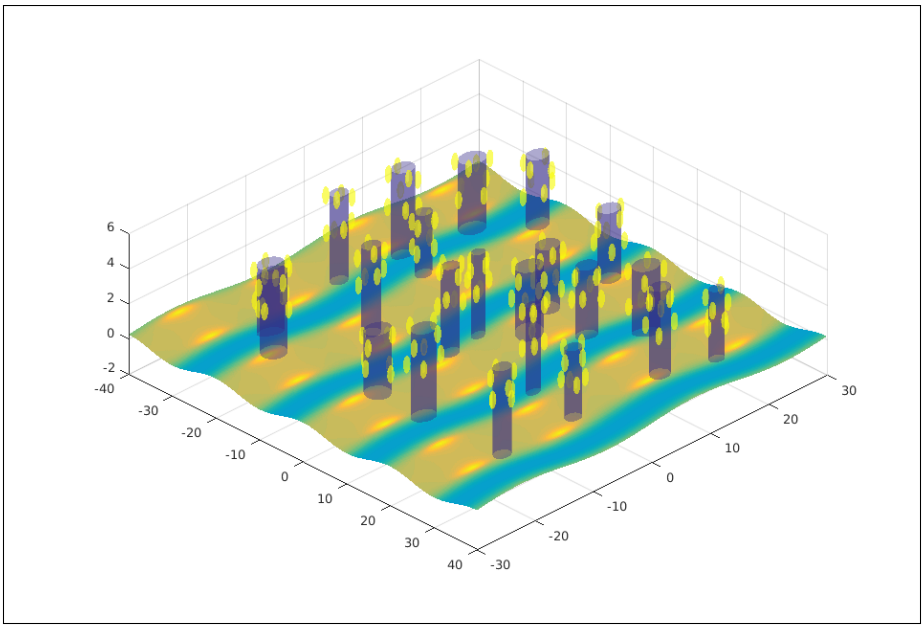


Figure 5.5: Forest simulation environment.

5.1.1 Effect of simulated trajectory

We began by simulating datasets using the values in Table 5.1 to construct the sensor trajectory. An example sensor trajectory in the Plane City environment is shown in Figure 5.6. Formally, the sensor position was represented as a homogenous position matrix $\mathbf{T}_c(t)$, constructed as

Table 5.1: Original amplitude and frequency values used for constructing the simulated sensor trajectory.

Parameter	**Nominal amplitude (m or rad)**	**Nominal frequency (Hz)**
x	3.2	0.5
y	2.5	0.29
z	2.3	0.4
ϕ	1	0.27
θ	0.63	0.2
ψ	1.26	0.28

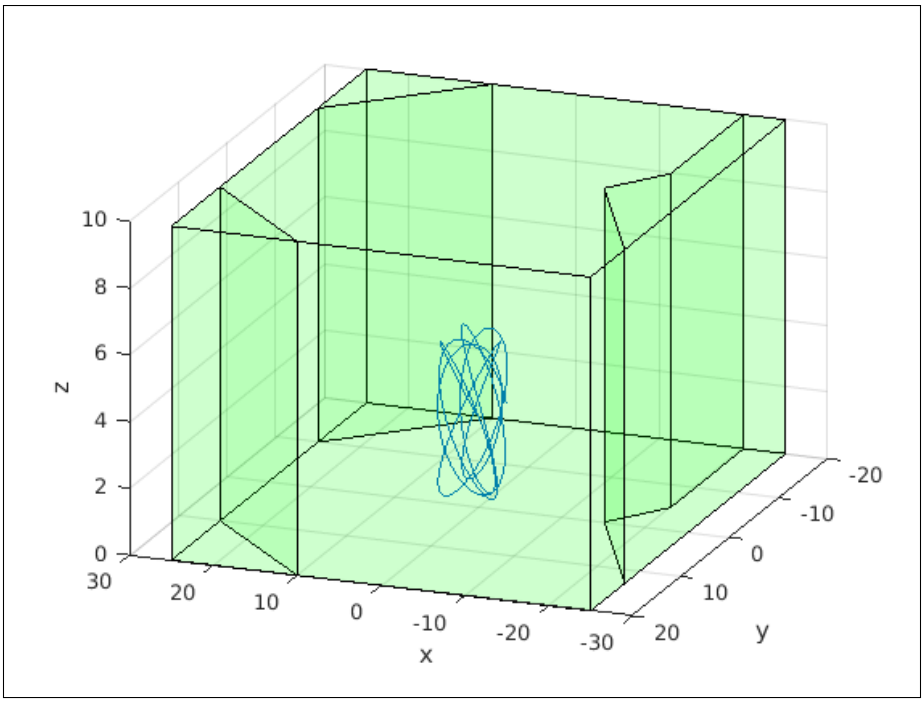


Figure 5.6: Example of the original simulation sensor trajectory.

$$\begin{bmatrix} x_c(t) \\ y_c(t) \\ z_c(t) \\ \phi_c(t) \\ \theta_c(t) \\ \psi_c(t) \end{bmatrix} = \begin{bmatrix} a_x \sin(f_x t) \\ a_y \sin(f_y t) \\ a_z \sin(f_z t) \\ a_\phi \sin(f_\phi t) \\ a_\theta \sin(f_\theta t) \\ a_\psi \sin(f_\psi t) \end{bmatrix}, \tag{5.1}$$

$$\mathbf{T}_c(t) = \begin{bmatrix} \mathbf{C}_c(t) & \mathbf{t}_c(t) \\ \mathbf{0} & 1 \end{bmatrix}, \tag{5.2}$$

$$\mathbf{C}_c(t) = \mathbf{C}_1(\phi_c(t))\mathbf{C}_2(\theta_c(t))\mathbf{C}_3(\psi_c(t)), \tag{5.3}$$

$$\mathbf{t}_c(t) = \begin{bmatrix} x_c(t) \\ y_c(t) \\ z_c(t) \end{bmatrix}, \tag{5.4}$$

where $\mathbf{C}_1(\cdot), \mathbf{C}_2(\cdot), \mathbf{C}_3(\cdot)$ are rotation matrices representing a principle rotation (rotations strictly about a single base axis). The matrices represent rotations about the x, y, and z axes respectively. We can also construct $\mathbf{C}_c(t)$ directly from the three rotation angles in a single step. The construction of the matrix is given in Appendix A.1.

Using this trajectory formulation, we simulate the sensor platform travelling through

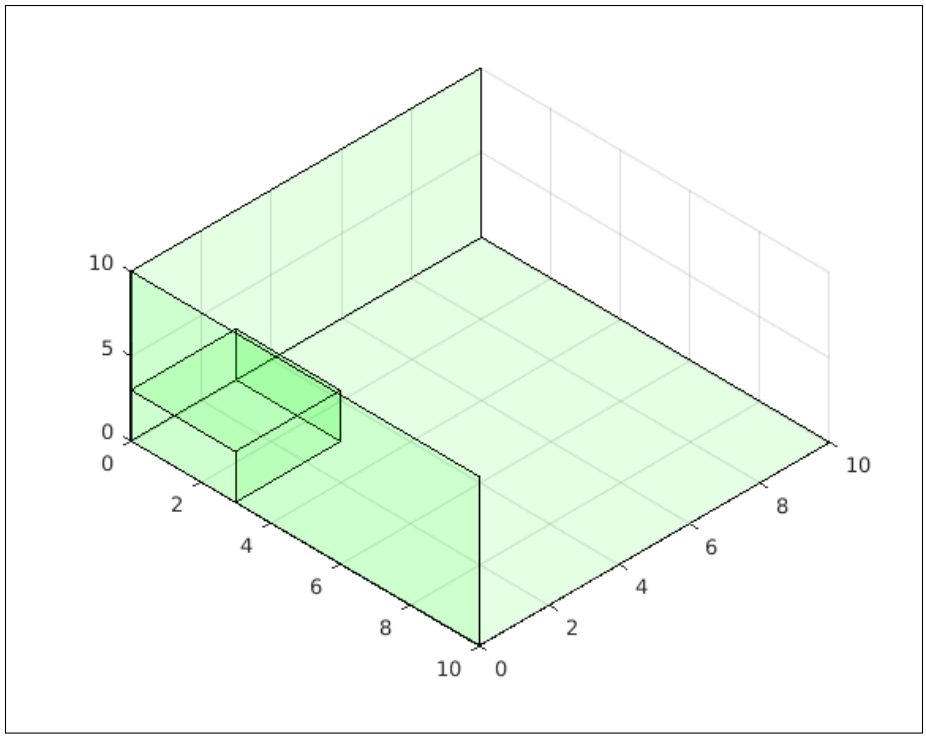


Figure 5.7: Corner Nook simulation environment.

each of the simulation environments and collecting lidar data. We forward the lidar data, along with estimates of the egosensor pose in $\mathfrak{se}(3)$ (the $\mathbf{y}_m$ in Section 3.1.1), to the calibration routines. We tested the calibration routine under two conditions: 1) with the ground truth camera positions, and 2) with noise-corrupted camera positions, simulating the inherent uncertainty of estimating camera position with VO. We used a sixth environment, Corner Nook, displayed in Figure 5.7 instead of Circular Room because, at the time, the latter had not yet been created.

The results in Table 5.2 demonstrate that accurate calibration is possible when we forward the ground truth poses to the calibration routine. The results for the Corner Nook, Plane City, and Underground Parking Lot datasets are of sufficient accuracy for most industrial applications. However, the results for the other two environments are not sufficient. Furthermore, the results are much worse when we corrupt the estimates of the camera positions with noise (a more realistic situation) in Table 5.3. For all results considered herein, we use 51 sampling iterations [1] (N_{sample} in Algorithm 1), 250 expected improvement maximizations (N_{iter}), and a k-value of 2.

[1] 51 may seem an oddly specific number, but it follows from the suggestion in [59] that $\frac{1}{N_{sample}-1}$ should result in a finite decimal value for the spacing between sample points

Table 5.2: Average absolute error over 2 unique trajectories in simulation, using ground truth camera position values for calibration. These datasets use the original amplitude and frequency values.

	Average absolute error							
Environment	x [mm]	y [mm]	z [mm]	ϕ [deg]	θ [deg]	ψ [deg]	Scale [$\times 10^{-3}$]	t_d [ms]
Simple Room	19.0	9.7	22.5	0.23	0.29	0.60	3.5	13.2
Corner Nook	1.0	6.1	10.0	0.72	0.0	0.63	0.0	1.4
Plane City	0.0	0.3	1.0	0.029	0.0	0.029	0.0	0.1
Underground Parking Lot	1.5	1.0	8.0	0.20	0.0	0.29	0.0	0.6
Forest	15.0	49.6	40.0	2.52	2.29	2.86	10.0	6.5

Table 5.3: Average absolute error over 2 unique trajectories in simulation, using noisy estimates of camera position for calibration. These datasets use the original amplitude and frequency values.

	Average absolute error							
Environment	x [mm]	y [mm]	z [mm]	ϕ [deg]	θ [deg]	ψ [deg]	Scale [$\times 10^{-3}$]	t_d [ms]
Simple Room	1.0	8.2	32.5	1.20	0.0	1.43	0.0	6.8
Corner Nook	3.5	5.8	2.0	1.29	0.29	1.29	0.50	9.7
Plane City	5.0	3.9	26.5	0.43	0.0	0.46	0.50	6.1
Underground Parking Lot	13.5	12.1	27.5	0.95	0.29	0.95	0.50	6.3
Forest	62.0	71.4	33.0	0.46	2.29	0.057	19.0	7.4

Table 5.4: New amplitude and frequency values used for constructing the simulated sensor trajectory.

Parameter	**Nominal amplitude (m or rad)**	**Nominal frequency (Hz)**
x	12.8	0.5
y	10.0	0.29
z	9.2	0.4
ϕ	4.0	1.08
θ	2.52	0.8
ψ	5.04	1.12

When camera position estimates are corrupted by noise, it has a highly non-linear effect on the lidar point positions, because they are functions of both the camera position and the calibration (the lidar to camera transform). It is possible for the camera positions to be altered in such a way that an inaccurate calibration transform ultimately results in a lower entropy cloud (lower average point-to-point distances). The assumption in this work and its predecessors is that with enough lidar data, the effect of zero-mean Gaussian noise on the camera positions estimates should cancel out. However that is not the case for the datasets in 5.3.

Ultimately, we discovered that a key to obtaining accurate calibration is to use a large sensor trajectory. We increased the nominal trajectory amplitude and frequency values so that the platform would move farther and faster through the simulation environment. We believe that this alleviates the impact of noise in trajectory estimates, because it effectively reduces the size of the error relative to the trajectory. When neighbouring scans are further apart, there are fewer possibilities for calibration transforms that form lidar points into common surfaces. Furthermore, using a faster trajectory also increases the temporal sensitivity of the cost function, as misestimating the sensor clock offsets will cause proportionally larger error in the estimated lidar poses. The new amplitude and frequency values are given in Table 5.4.

After creating new datasets with the trajectories defined in Table 5.4, we were able to obtain calibration accuracy on the order of 5 mm or less for translation, 0.1 degrees for orientation, and 0.15 ms for the time delay. This compares favourably to state of the art methods—such as [42] which achieves calibration accuracy of less than 1 mm, 0.01 degrees, and 0.05 ms—given that we remove the need for a known target. The results are given in Table 5.5.

The exception to the accurate results is the Forest dataset, which continues to display poor calibration accuracy. We believe that the poor accuracy is primarily the result of scanning mostly free space, ultimately giving far fewer usable lidar points. 2 million of the 3.8 million simulated lidar beams (slightly more than 50 %) did not return a measurement for the forest dataset. In contrast, all 3.8 million lidar beams returned a measurement for the other four simulation environments.

Table 5.5: Average absolute error over 5 unique trajectories in simulation, using noisy estimates of camera position for calibration. These datasets use the new amplitude and frequency values.

	Average absolute error							
Environment	x [mm]	y [mm]	z [mm]	ϕ [deg]	θ [deg]	ψ [deg]	Scale [$\times 10^{-3}$]	t_d [ms]
Simple Room	1.07	1.68	2.21	0.0118	0.0252	0.0115	0.0132	0.0196
Circular Room	0.963	3.12	3.54	0.026	0.0498	0.0078	0.0547	0.135
Plane City	2.45	3.37	2.8	0.0096	0.0383	0.0088	0.126	0.0292
Underground Parking Lot	2.45	4.33	3.69	0.0282	0.0645	0.0139	0.178	0.0811
Forest	12.9	11.4	13.9	1.06	0.548	0.235	35.8	4.76

5.2 Real World Data

In order to confirm the applicability of the algorithm, we tested its performance on several datasets collected in the real world. For many of the reasons discussed before (e.g., difficulty of measuring around sensor platform, unknown measurement origin within sensor body), we do not have an accurate ground truth spatial transformation. The same is true for the sensor clock offsets. As we do not have access to two way timing data, we cannot make use of something like TICSync [6] to obtain an accurate value to evaluate against. Instead, we evaluate the repeatibility of the output of our routine. We follow with a description of the data capture platform and an analysis of the results of our routine.

5.2.1 Experimental Setup

We tested several datasets collected using a hand-held sensor rig with attached Hokuyo UTM-30LX 2D lidar, and PointGrey Flea3 monocular greyscale camera. The lidar collects data in a 270° arc with 0.25° angular resolution. However, we found it difficult to avoid scanning the operator at the edges of the scan. To avoid corrupting the algorithm with any lidar beams hitting the operator, we only use the middle 180° arc of the scan, for a total of 721 lidar returns per scan. The lidar collected scans at a rate of 30 Hz. The camera collects images in greyscale with a 640 x 512 resolution. The camera collects data at 200 frames per second, but we downsample it to obtain a data stream of roughly 30 frames per second. Unlike [29], we do not assume the lidar and camera data streams are synced. We instead maintain a seperate set of timestamps for the streams from each sensor, and handle asynchronous data by interpolating between camera poses, as explained in Section 4.1.2. We used ORB-SLAM2 [52] to estimate the trajectory of the camera (with unknown scale). This dataset was used previously in [7, 29].

5.2.2 Real World Results

We seeded the Bayesian optimization procedure with fairly loose bounds on the calibration parameters as follows,

$$\boldsymbol{\Xi}_{lb} = \begin{bmatrix} -50 & -100 & -350 & 149 & -17 & -115 & 0.05 & -30 \end{bmatrix}, \tag{5.5}$$

$$\boldsymbol{\Xi}_{ub} = \begin{bmatrix} 150 & 100 & -150 & 206 & 17 & -69 & 0.8 & 30 \end{bmatrix}, \tag{5.6}$$

where values are in units of ([mm] [mm] [mm] [°] [°] [°] [unitless] [ms]) respectively.

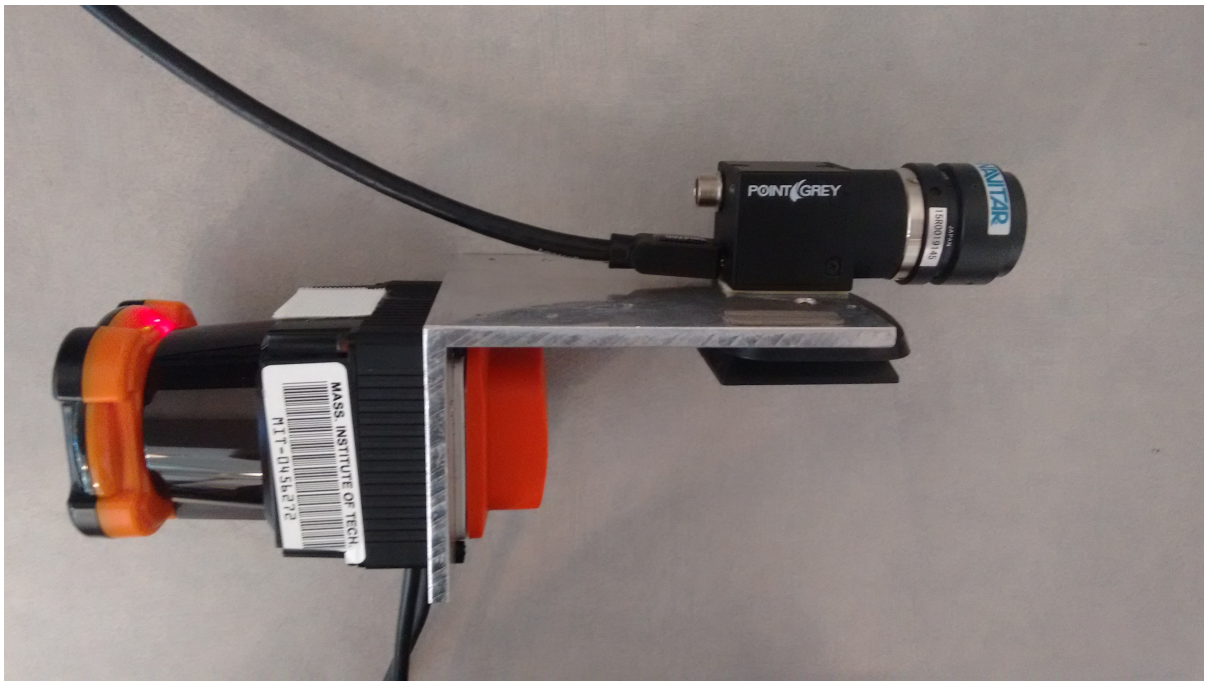


Figure 5.8: Sensor platform used for real world data collection. Note that the sensors do not have overlapping fields of view.

Figure 5.9: Office space at the MIT STATA Center used for data collection.

Table 5.6: Calibration results for three real-world datasets.

	Estimated calibration parameter							
	x [mm]	y [mm]	z [mm]	ϕ [°]	θ [°]	ψ [°]	Scale	t_d [ms]
Dataset 1	50.5	0.24	-211.0	181.7	-1.1	-88.8	0.216	15.2
Dataset 2	55.1	7.68	-244.2	186.1	1.04	-89.4	0.212	18.8
Dataset 3	47.8	-0.35	-224.2	184.2	-1.45	-89.7	0.214	18.2

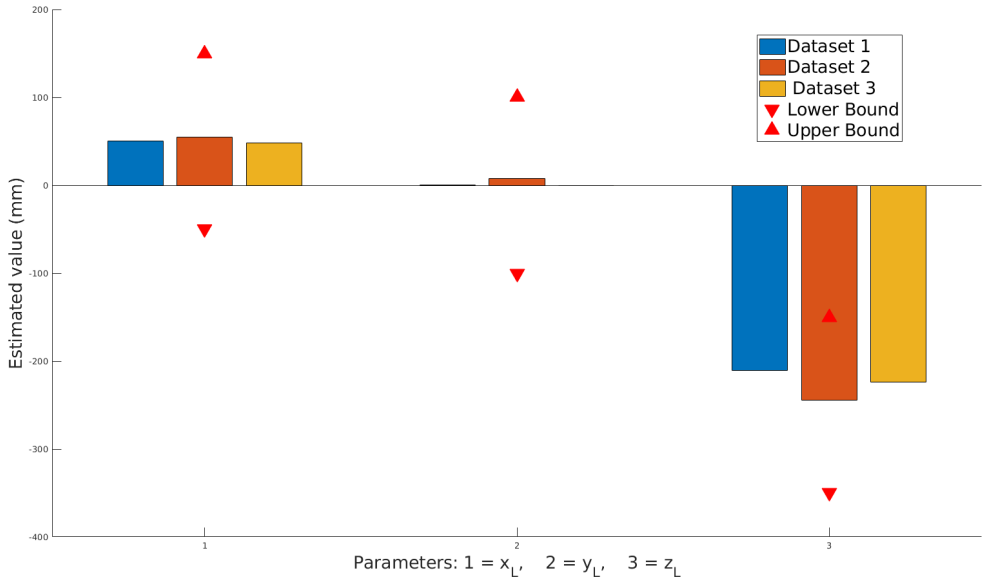


Figure 5.10: Repeatability of the estimation of translational calibration parameters with respect to parameter bounds.

For each optimization, we used 51 sampling iterations and 250 expected improvement maximizations. We found that a k-value of 10 (Criterion 4.23) was the largest we could feasibly allow while still finishing the computation in a realistic amount of time.

The results are summarized in Table 5.6. We find that the estimated calibration parameters are highly repeatable between datasets, with the exception of some outliers (e.g., z in Dataset 2). Discounting these outliers, the results show a repeatability of $\pm$ 4 mm, 1 degree, and 0.1 ms in the translational, rotational, and time-delay parameters respectively. Figures 5.10 and 5.11 further demonstrate the repeatability of the calibration. For all six spatial calibration parameters, the estimated values for each dataset agree to within a small fraction of the total width of the bounds supplied to the optimizer.

Lastly, we perform a subjective validation of the algorithm by comparing the resultant point cloud of a dataset collected at the University of Toronto Institute for Aerospace

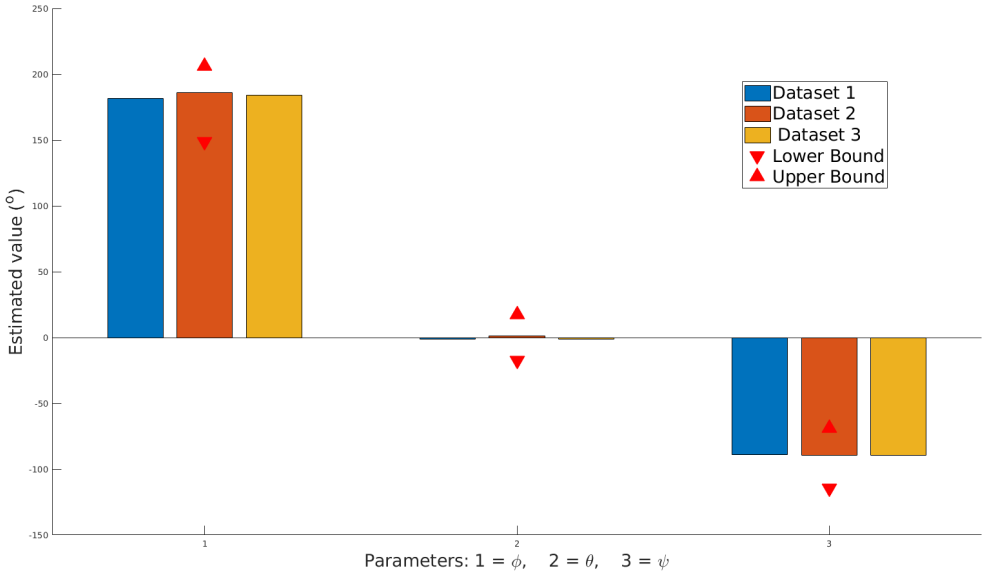
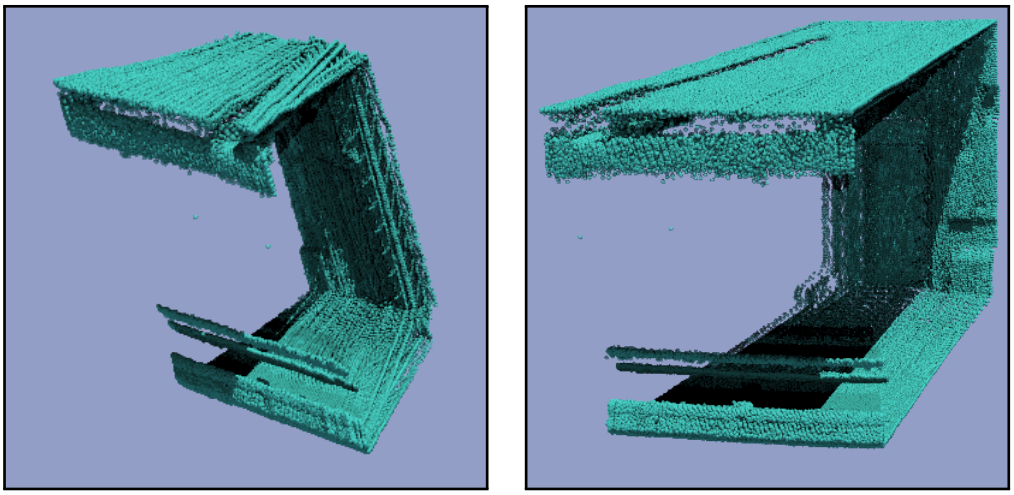


Figure 5.11: Repeatability of the estimation of rotational calibration parameters with respect to parameter bounds.

Studies. When we construct the point cloud using parameter values at the optimizer bounds (Figure 5.12a), walls appear crooked, and scans of planar surfaces do not align properly. Using the output calibration values from our algorithm, the constructed point cloud in Figure 5.12b looks as it should, a set of 3 flat surfaces arranged at 90 degree angles.



(a) (b)

Figure 5.12: Lidar sweep of a portion of the experimental laboratory space at the Institute for Aerospace Studies, (*a*) before calibration, and (*b*) after calibration has been performed. Note that planar surfaces appear to be planar after calibration. This dataset was collected with a different lidar than the MIT data, the Hokuyo UST-20LX.

Chapter 6

Extension to IMU-Lidar Calibration

As we have noted, the egomotion sensor in our calibration routine is used solely in the preprocessing step to supply a base sensor trajectory. Our choice to use a camera is motivated primarily by the ability to obtain highly accurate trajectory estimates using visual odometry. In theory, any sensor can be used if it is possible to analyze its data and obtain a set of estimated poses. The calibration of a planar lidar directly to an IMU is an intriguing research problem. Some methods, such as [42] calibrate a multi-sensor suite containing one or more of each of a planar lidar and IMU, but in this case a camera is also included in the sensor suite. Although this adds an additional calibration transform to be estimated, adding a camera actually simplifies the process as it allows the fusion of camera and IMU data to obtain a highly accurate base sensor trajectory. To our knowledge, there is no existing calibration routine designed for direct planar lidar to IMU extrinsic calibration. Such a routine would be valuable for applications where designers may want to use only a planar lidar and IMU, and forgo a camera to minimize costs.

6.1 Problem Formulation

An IMU provides a stream of measurements of its own angular velocity and linear acceleration. We can estimate an IMU's trajectory by storing its position, rotation, and linear velocity as the sensor state, and integrating the angular velocities and linear accelerations obtained as measurements. However, the use of an IMU as a platform's sole egomotion sensor is complicated by inertial rate measurement biases which occur in MEMS IMUs. These biases corrupt the measurement of the true velocity/acceleration, and evolve as Gaussian random walks over short periods of time [65]. The IMU system can be modelled

by

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}_{IW} & \boldsymbol{\phi}_{IW} & \mathbf{v}_{IW} \end{bmatrix}^T, \tag{6.1}$$

$$\dot{\mathbf{p}}_{IW} = \mathbf{v}_{IW}, \quad \dot{\mathbf{v}}_{IW} = \mathbf{a}_{IW}, \quad \dot{\boldsymbol{\phi}}_{IW} = \boldsymbol{\omega}_{IW},$$

$$\mathbf{a}_m = \mathbf{a}_{IW} + \mathbf{b}_a, \quad \boldsymbol{\omega}_m = \boldsymbol{\omega}_{IW} + \mathbf{b}_\omega,$$

$$\dot{\mathbf{b}}_a \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\Sigma}_a\right), \quad \dot{\mathbf{b}}_\omega \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\Sigma}_\omega\right),$$

where the subscript $_m$ denotes a measured quantity, the subscript $_{IW}$ denotes a velocity or acceleration of the IMU relative to the inertial (or "world") frame, and a dotted quantity denotes a time derivative.

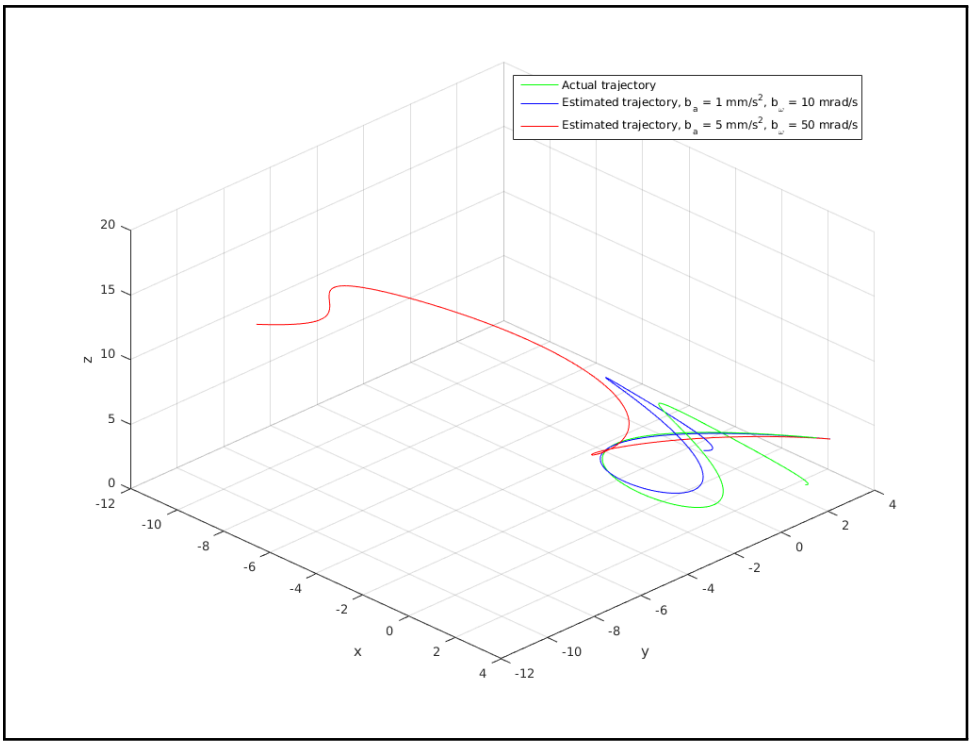
It is clear that ignoring the IMU bias will result in an inaccurate trajectory (Figure 6.1). Even a small constant bias results in linear error growth in the estimated rotation, and quadratic error growth in the estimated position. It is clear that to obtain an IMU trajectory of sufficient accuracy, we must filter out the measurement biases. Just as we did with trajectory scale for a camera, we can add the IMU biases as parameters to be optimized. Our hypothesis is that given enough lidar data captured from many different viewpoints, the RQE will be minimized when we register the lidar data to the true base sensor trajectory, rather than an inaccurate trajectory resulting from bias misestimation.

We can simplify the problem by assuming that the rates of change of the IMU biases are not too severe, such that the biases are nearly constant over short periods of time (60 - 90 s). For our purposes then, we will need to estimate 6 additional parameters: the 6 measurement biases. However, we no longer have the issue of trajectory scale. Furthermore, for our first attempt at solving this problem, we will ignore time-delay effects. Ultimately, we estimate a 12DOF state using our RQE formulation.

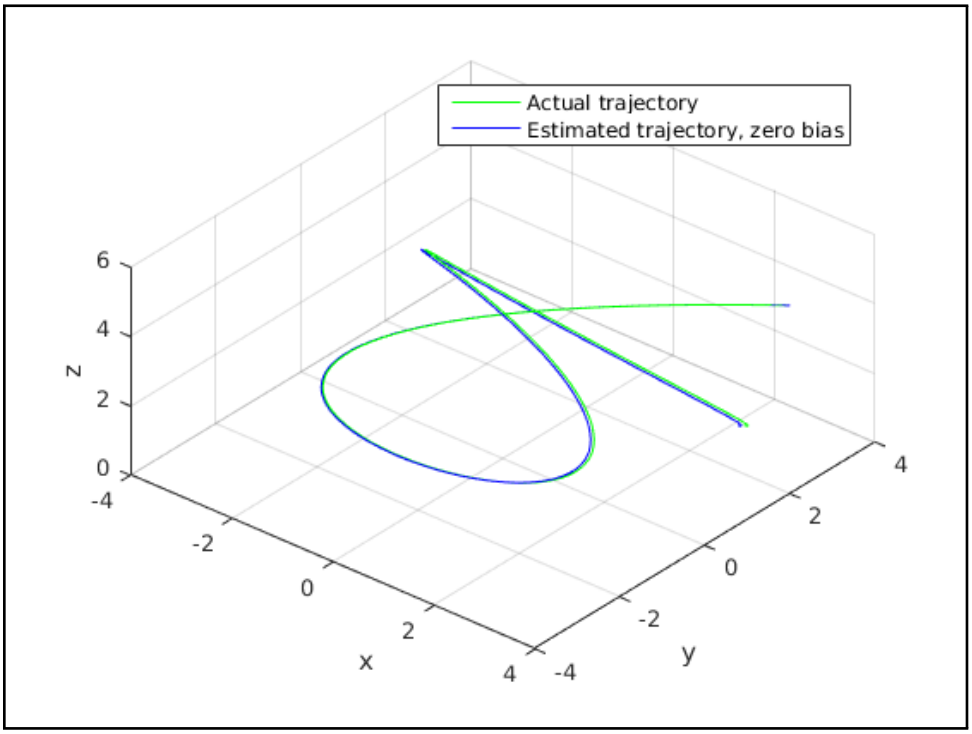
$$\boldsymbol{\Xi} = \begin{bmatrix} x_L & y_L & z_L & \phi_L & \theta_L & \psi_L & \mathbf{b}_a & \mathbf{b}_\omega \end{bmatrix}^T \tag{6.2}$$

$$\mathbf{b}_a = \begin{bmatrix} b_x & b_y & b_z \end{bmatrix}^T, \quad \mathbf{b}_\omega = \begin{bmatrix} b_\phi & b_\theta & b_\psi \end{bmatrix}^T$$

We construct the point cloud and compute its entropy just as before. The only difference is that instead of a set of egomotion sensor poses, our input is a set of linear acceleration and angular velocity measurements. We subtract the estimated biases from these measurements, and update the IMU state according to Equation (6.1). We update the position, velocity, and rotation angles from their time derivatives using the Runge-Kutta 45 integration scheme, at the update rate of the sensor (100 Hz in our case). Once we have propogated the measurements forward to obtain the set of IMU poses,we proceed



(a)



(b)

Figure 6.1: *a*) Effect of level of IMU bias on the estimated IMU trajectory. Note that, even in the absence of bias (*b*), the estimated trajectory still has some small error as a result of the finite IMU update rate used for integrating velocities and accelerations.

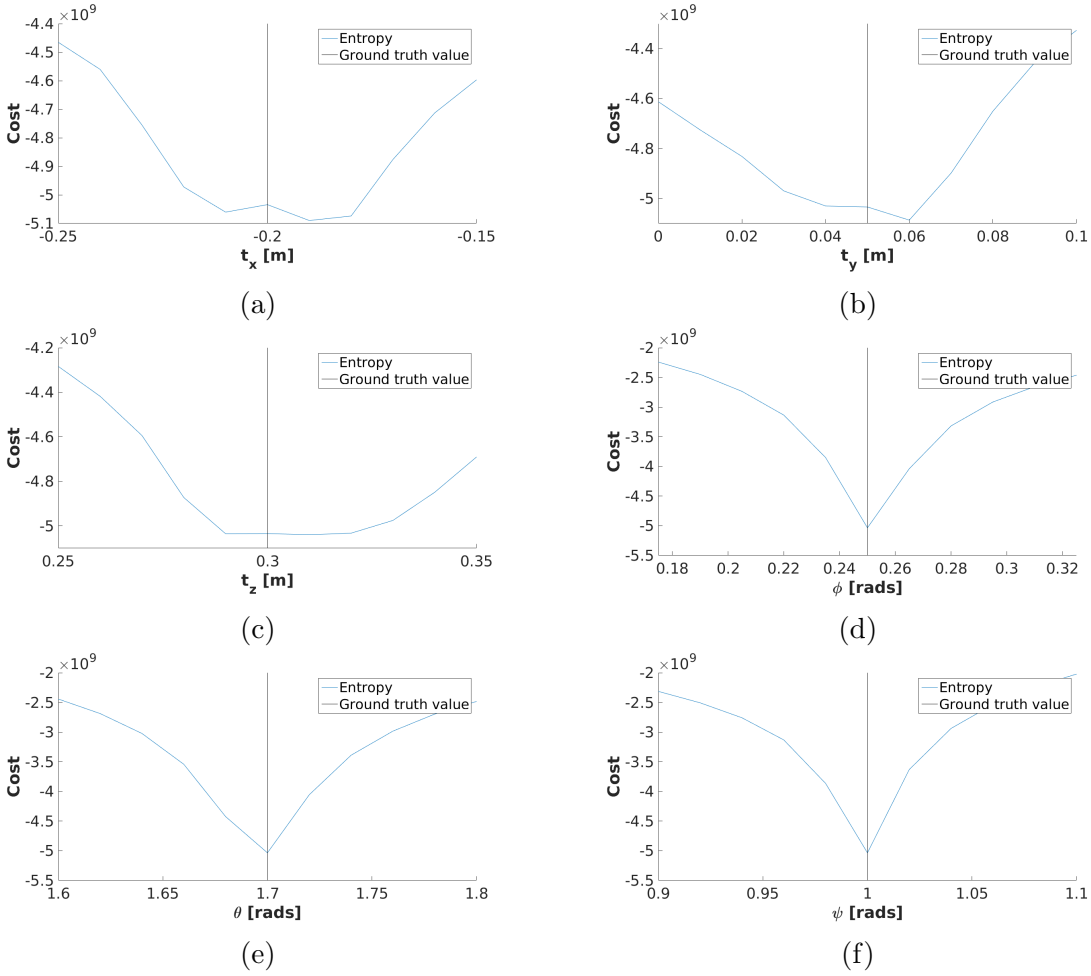


Figure 6.2: Macro scale behaviour of entropy for each IMU to planar lidar spatial calibration parameter. In each plot, all 11 other parameters are held constant at ground truth values. *a)* x_L *b)* y_L *c)* z_L *d)* ϕ_L *e)* θ_L *f)* ψ_L

just as before.

6.2 Calibration

As in Section 4.3, we would first like to confirm that our cost function will converge to the correct calibration parameters. We conduct a line search on each of the 12 calibration parameters (six spatial, six IMU biases) to see their effect on point cloud entropy. For each line search, we hold the 11 other calibration parameters at their ground truth values, in an attempt to isolate the effect of each calibration parameter on its own. The results are plotted in Figure 6.2 (spatial calibration parameters) and Figure 6.3 (IMU biases).

The cost is roughly convex for most of the parameters, and demonstrates a minimum

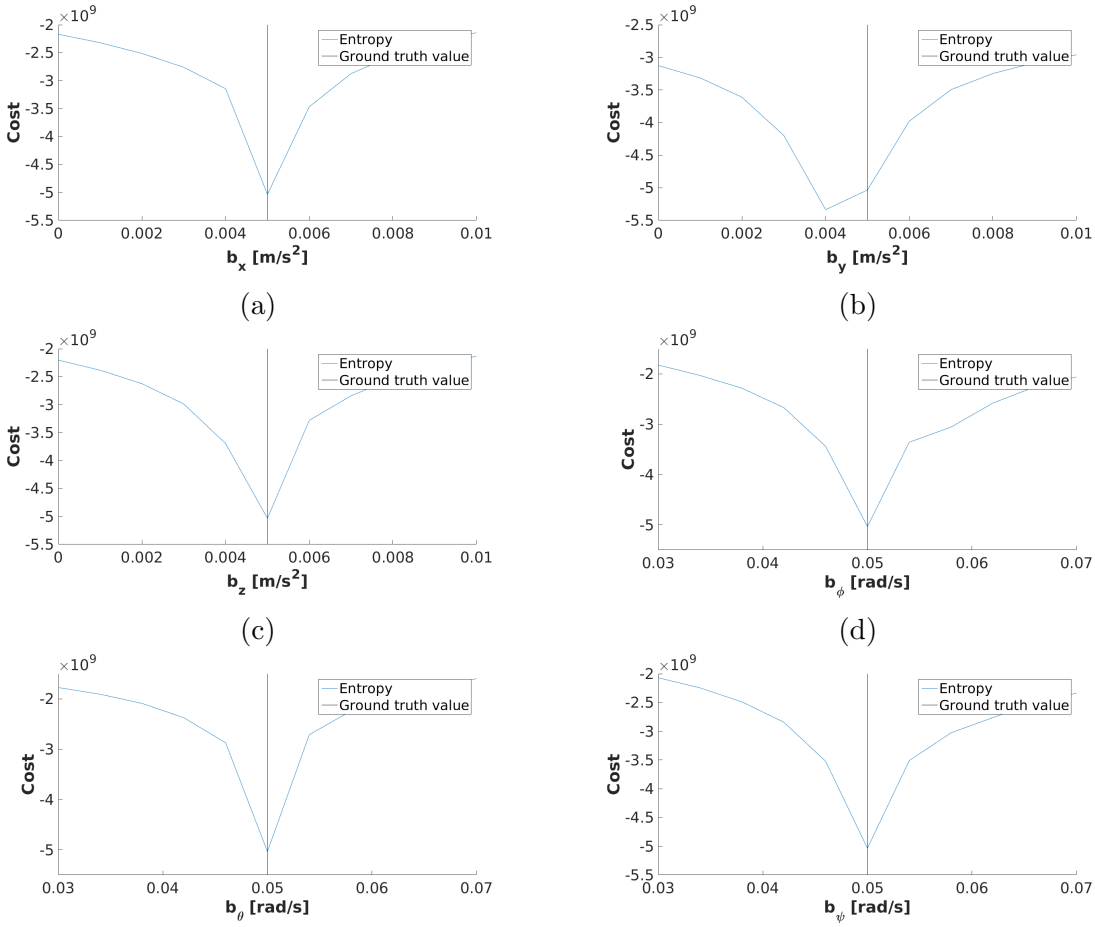


Figure 6.3: Macro scale behaviour of entropy for each IMU bias parameter. In each plot, all 11 other parameters are held constant at ground truth values. *a)* b_x *b)* b_y *c)* b_z *d)* b_ϕ *e)* b_θ *f)* b_ψ

at ground truth. The exception is the translational parameters, t_x, t_y, and t_z. These exhibit a larger basin of convergence, and it is hard to tell where the minima occur, though they are at least close to ground truth. This is further reflected in the calibration results. Using the same Bayesian Optimization method as for the camera-lidar case, we optimized the 12 IMU-lidar calibration parameters for a dataset simulated on the Plane City environment. The results are given in Table 6.1. As might be expected from the plots, we are able to estimate the rotational and bias parameters very accurately (sub-degree, sub-mm/s^2, and sub-mrad/s). However, the translational parameters demonstrate errors of between 14 and 50 mm. This is much worse than the camera-lidar case.

Table 6.1: Average absolute IMU-lidar calibration parameter error in simulation.

	Average absolute error					
Environment	x [mm]	y [mm]	z [mm]	ϕ [deg]	θ [deg]	ψ [deg]
Plane City	27.20	14.39	49.08	0.15	0.083	0.36
Environment	b_x [mm/s^2]	b_y [mm/s^2]	b_z [mm/s^2]	b_ϕ [mrad/s]	b_θ [mrad/s]	b_ψ [mrad/s]
Plane City	0.24	2.84	0.10	0.028	0.25	0.034

Ultimately, we are not able to accurately calibrate the translational parameters of the IMU-lidar transform at this time. However, we are encouraged by the results for the nine other parameters, and the somewhat convex behaviour of the entropy with respect to the translational parameters.

Chapter 7

Conclusion

We have developed a full spatiotemporal calibration routine for a 2D lidar and egomotion sensor platform, based on the prinicple of entropy minimization. We have demonstrated that a cost function of point cloud entropy converges to highly accurate spatial, scale, and time-delay parameters. Importantly, our method may be applied to platforms containing a lidar and any sensor that can accurately measure its own egomotion. These include monocular, stereo, and depth cameras, 3D lidars, IMUs, or even radar. The application of this method to calibration of sensor clock time delays has important implications. The analysis presented in Section 4.3.1 suggests that our method can likely be extended to calibration of any parameter that would have an effect on the construction of a point cloud. This may include lidar and camera intrinsics, stereo baseline distance, or IMU inertial rate biases, though these may not all be observable strictly by analysing the entropy.

We examined the suitability of this method to simultaneous IMU bias and spatial transform calibration in Chapter 6. Therein, we found that much like the spatiotemporal parameters, the point cloud entropy appears to be minimized at the true IMU inertial rate bias values. This allowed for highly accurate calibration of the biases for a dataset 1-2 minutes in length. Unfortunately, the lidar-IMU problem formulation did not exhibit accurate calibration of the translational calibration parameters. We believe that this problem is open to future research including 1) a study of the gain in calibration accuracy from adding more planar objects to the environment, and 2) a spline-based trajectory representation to reduce the error associated with forward integration of inertial rates.

While we demonstrate highly accurate calibration both in simulation and on real world datasets, this approach would benefit from a more in depth study of its behaviour under different conditions (that is, how the shape of the cost function is affected). For our part, we present figures for the accuracy of the calibration in environments of varying

complexity. However, the difference between these environments is hard to quantify as they contain varying numbers of different shapes. A more fundamental study of the gain in accuracy as a function of the number of planar, cylindrical, or spherical targets would help to provide insight into the environmental requirements of the calibration routine. The effect of the sensor trajectory could also be studied in greater detail by quantifying the effects of different trajectories of varying complexity. For example, in our simulation we parameterize each of the 6 degrees of freedom of the trajectory by a single sinusoid. A trajectory consisting of additional sinusoids, or a trajectory formulated entirely differently might make the problem more observable.

Lastly, there a number of improvements that could be made to the problem to make it more computationally tractable, and the end result more accurate. In Section 4.3.2, we discussed in great detail the need for a simplification to problem, criterion 4.23. This is necessary because for a dataset large enough to constrain the problem, computing the full set of point-to-point entropy contributions is a computationally intractable $O(N^2)$ operation. Adding criterion 4.23 greatly speeds up the computation, but results in jagged behaviour of the cost function at the micro scale, meaning it does not have continuous analytical derivatives. Theoretically, the full entropy computation could be performed in parallel on a GPU, as the point-to-point distances are set once the point cloud has been constructed. However, as can be seen from the cost function (Equation (4.22)), the covariance matrix for the Gaussian kernel is not constant. In fact, it depends on both indices i and j, so it is different for every set of point pairs. Therefore, implementation on a GPU would require some manner of parallelizing an operation with a dynamic kernel. Were this made possible, we believe the calibration could be even more precise, as the cost function could account for the entropy of the entire point cloud. Moreover, the runtime of the algorithm would be dramatically decreased by the parallelization of the entropy calculation, and the additional ability to calculate analytical derivatives of the cost function, allowing the use of Newton's or other gradient-based optimization methods.

Bibliography

[1] P. Wu et al. A novel algorithm of autonomous obstacle-avoidance for mobile robot based on lidar data. In *Proc. IEEE Int. Conf. Robot. Biomimetics*, pages 2377–2382, Dec. 2015.

[2] B. Suger, B. Steder, and W. Burgard. Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3D-lidar data. In *Proc. IEEE Int. Conf. Robot. Autom.*, pages 3941–3946, May 2015.

[3] A Bry, C Richter, A Bachrach, and N Roy. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *International Journal of Robotics Research*, 37(7):969–1002, June 2015.

[4] I Baldwin and P Newman. Road vehicle localization with 2d push-broom lidar and 3d priors. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '12)*, pages 2611–2617, Saint Paul, USA, 2012.

[5] R Munguia and A Grau. Monocular slam for visual odometry: A full approach to the delayed inverse-depth feature initialization method. *Mathematical Problems in Engineering*, 2012(5):1–26, 2012.

[6] A Harrison and P Newman. Ticsync: Knowing when things happened. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '11)*, pages 356–363, Shanghai, China, 2011.

[7] J Lambert, L Clement, M Giamou, and J Kelly. Entropy-based sim(3) calibration of 2d lidars to egomotion sensors. In *Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI '16)*, pages 455–461, Baden-Baden, Germany, 2016.

[8] S. Wasielewski and O. Strauss. Calibration of a multi-sensor system laser rangefinder / camera. In *Proceedings of the Intelligent Vehicles Symposium*, pages 472–477, Detroit, USA, 1995.

[9] Qilong Zhang and Robert Pless. Extrinsic calibration of a camera and laser range finder (improves camera calibration). In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2301–2306, Sendai, Japan, 2004.

[10] R.M. Haralick, C. Lee, K. Ottenberg, and M. Nlle. Review and analysis of solutions of the three point perspective pose estimation problem. *International Journal of Computer Vision*, 13(3):331–356, 1994.

[11] C X Guo and S I Roumeliotis. An analytical least-squares solution to the line scan lidar-camera extrinsic calibration problem. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '13)*, pages 2943–2948, Karlsruhe, Germany, 2013.

[12] L. Zhou. A new minimal solution for the extrinsic calibration of a 2d lidar and a camera using three plane-line correspondences. *IEEE Sensors Journal*, 14(2):442–454, 2014.

[13] A. Geiger, F. Moosmann, . Car, and B. Schuster. Automatic camera and range sensor calibration using a single shot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '12)*, pages 3936–3943, Saint Paul, USA, 2012.

[14] Y Bok, D Choi, P Vasseur, and Kweon I S. Extrinsic calibration of non-overlapping camera-laser system using structured environment. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '14)*, pages 436–443, Chicago, USA, 2014.

[15] H. Yang, X. Liu, and I. Patras. A simple and effective extrinsic calibration method of a camera and a single line scanning lidar. In *Proceedings of the International Conference on Pattern Recognition (ICPR '12)*, pages 1439–1442, Tsukuba, Japan, 2012.

[16] D. Scaramuzza, A. Harati, and R. Siegwart. Extrinsic self calibration of a camera and a 3d laser range finder from natural scenes. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '07)*, pages 4164–4169, San Diego, USA, 2007.

[17] P. Moghadam, M. Bosse, and R. Zlot. Line-based extrinsic calibration of range and

image sensors. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '13)*, pages 3685–3691, Karlsruhe, Germany, 2013.

[18] J. Castorena, U. S. Kamilov, and Boufonos P. T. Autocalibration of lidar and optical cameras via edge alignment. In *Proceedings of the IEEE International Conference on Acostics, Speech and Signal Processing (ICASSP '16)*, pages 2862–2866, Shanghai, China, 2016.

[19] Gaurav Pandey, James McBride, Silvio Savarese, and Ryan Eustice. Automatic extrinsic calibration of vision and lidar by maximizing mutual information. *Journal of Field Robotics*, 32(5):696–722, 2014.

[20] Z. Taylor and J. Nieto. A mutual information approach to automatic calibration of camera and lidar in natural environments. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA '12)*, Wellington, New Zealand, 2012.

[21] Jonathan Brookshire and Seth Teller. Automatic calibration of multiple coplanar sensors. In *Proceedings of Robotics: Science and Systems*, Los Angeles, USA, June 2011.

[22] Jonathan Brookshire and Seth Teller. Extrinsic calibration from per-sensor ego-motion. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[23] Z. Taylor and J. Nieto. Parameterless automatic extrinsic calibration of vehicle mounted lidar-camera systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '14)*, pages 1–3, Hong Kong, China, 2014.

[24] C. Le Gentil, T. Vidal-Calleja, and S. Huang. 3d lidar-imu calibration based on upsampled preintegrated measurements for motion distortion correction. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '18)*, Brisbane, Australia, 2018.

[25] T. Lee, S. Lim, S. Lee, S. An, and S. Oh. Indoor mapping using plane extracted from noisy rgb-d sensors. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '12)*, pages 1727–1733, Vilamoura, Portugal, 2012.

[26] M. Kaess. Simultaneous localization and mapping with infinite planes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '15)*, pages 4605–4611, Seattle, USA, 2015.

[27] M Sheehan, A Harrison, and P Newman. Automatic self-calibration of a full field-of-view 3d n-laser scanner. In *Proceedings of the International Symposium on Experimental Robotics (ISER '10)*, pages 165–178, New Delhi and Agra, India, 2010.

[28] Will Maddern, Alastair Harrison, and Paul Newman. Lost in translation (and rotation): Rapid extrinsic calibration for 2d and 3d lidars. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '12)*, pages 3096–3102, Saint Paul, USA, 2012.

[29] J Lambert. Automatic and featureless sim(3) calibration of planar lidars to egomotion sensors. Masc thesis.

[30] Alfréd Rényi. On measures of entropy and information. In *Proc. Berkeley Symp. Math. Stat. and Probability*, volume 1, pages 547–561. University of California Press, 1961.

[31] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, September 1989.

[32] A. Carballo, Y. Hara, H. Kawata, T. Yoshida, A. Ohya, and S. Yuta. Time synchronization between sokuiki sensor and host computer using timestamps. In *Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI '08)*, pages 261–266, Seoul, Korea, 2008.

[33] H Sommer, R Khanna, I Gilitschenski, Z Taylor, R Siegwart, and J Nieto. A low-cost system for high-rate, high-accuracy temporal calibration for lidars and cameras. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '17)*, pages –, Vancouver, Canada, 2017.

[34] A. English, P. Ross, D. Ball, B. Upcroft, and P. Corke. Triggersync: A time synchronization tool. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '15)*, pages 6220–6226, Seattle, USA, 2015.

[35] Jonathan Kelly and Gaurav S. Sukhatme. A general framework for temporal calibration of multiple proprioceptive and exteroceptive sensors. In Oussama Khatib, Vijay Kumar, and Gaurav S. Sukhatme, editors, *Experimental Robotics: The 12th*

International Symposium on Experimental Robotics, volume 79 of *Springer Tracts in Advanced Robotics*, pages 195–209. Springer, Berlin Heidelberg, 2014. doi: 10.1007/978-3-642-28572-1_14.

[36] Jonathan Kelly, Nicholas Roy, and Gaurav S. Sukhatme. Determining the time delay between inertial and visual sensor measurements. *IEEE Transactions on Robotics*, 30(6):1514–1523, December 2014.

[37] Fredy Tungadi and Lindsay Kleeman. Time synchronisation and calibration of odometry and range sensors for high-speed mobile robot mapping. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA '08)*, pages 1–8, Canberra, Australia, 2008.

[38] Fadri Furrer, Marius Fehr, Tonci Novkovic, Hannes Sommer, Igor Gilitschenski, and Roland Siegwart. Evaluation of combined time-offset estimation and hand-eye calibration on robotic datasets. In *Proceedings of Field and Service Robotics (FSR '17)*, pages 1–14, Zurich, Switzerland, 2017.

[39] Elmar Mair, Michael Fleps, Michael Suppa, and Darius Burschka. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, Phuket, Thailand.

[40] M Li and A I Mourikis. 3d motion estimation and online temporal calibration for camera-imu systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '13)*, pages 5709–5716, Karlsruhe, Germany, 2013.

[41] Joern Rehder, Paul Beardsley, Roland Siegwart, and Paul Furgale. Spatio-temporal laser to visual/inertial calibration with applications to hand-held, large scale scanning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'14)*, pages 459–465, Chicago, Illinois, USA, September 2014. doi: 10.1109/IROS.2014.6942599.

[42] Joern Rehder, Roland Siegwart, and Paul Furgale. A general approach to spatiotemporal calibration in multisensor systems. *IEEE Transactions on Robotics*, 32 (2):383–398, April 2016.

[43] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'13)*, pages 1280–1286, Tokyo, Japan, November 2013. doi: 10.1109/IROS.2013.6696514.

[44] T D Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017. ISBN 9781107159396. doi: 10.1017/9781316671528.

[45] S Anderson and T D Barfoot. Full steam ahead: Exactly sparse gaussian process regression for batch continuous-time trajectory estimation on se(3). In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '15)*, pages 157–164, Hamburg, Germany, 2015.

[46] S Anderson, T D Barfoot, C H Tong, and S Särkkä. Batch nonlinear continuous-time trajectory estimation as exactly sparse gaussian process regression. *Autonomous Robots*, 39(3):221–238, October 2015.

[47] S Anderson. *Batch Continuous-Time Trajectory Estimation*. PhD thesis.

[48] C.E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27(3):379–423, July 1948.

[49] P.A. Bromiley, N.A. Thacker, and E. Bouhova-Thacker. Shannon entropy, renyi entropy, and information. *Tina Vision Staistics and Segmentation Series*, 2008(1): 1–8, July 2010.

[50] J.N. Kapur. *Measures of Information and Their Applications*. Wiley.

[51] E Parzen. On estimation of a probability density function and mode. *Annals Mathematical Statistics*, 33(3):1065–1076, 1962.

[52] Raul Mur-Artal et al. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Trans. Robot.*, 31(5):1147–1163, Sep. 2015.

[53] D G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[54] H Bay, T Tuytelaars, and L Van Gool. Surf: Speeded up robust features. In *Proceedings of the European Conference on Computer Vision (ECCV '06)*, pages 404–417, Graz, Austria, 2006.

[55] T D Barfoot, C H Tong, and S Särkkä. Batch continuous-time trajectory estimation as exactly sparse gaussian process regression. In *Proceedings of Robotics: Science and Systems (RSS '14)*, Berkeley, USA, July 2014.

[56] Vlad I. Morariu, Balaji Vasan Srinivasan, Vikas C. Raykar, Ramani Duraiswami, and Larry S. Davis. Automatic online tuning for fast gaussian summation. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[57] V. C. Raykar, C. Yang, R. Duraiswami, and N. Gumerov. Fast computation of sums of gaussians in high dimensions. Technical Report CS-TR-4767, Department of Computer Science, University of Maryland, CollegePark, 2005.

[58] Jonas Mockus. *Bayesian Approach to Global Optimization*, volume 37 of *Mathematics and Its Applications*. Kluwer Academic Publishers.

[59] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(13):455–492, June 1998.

[60] M Franey, P Ranjan, and H Chipman. Branch and bound algorithms for maxmizing expected imporvement functions. *Journal of Statistical Planning and Inference*, 141 (1):42–55, January 2011.

[61] Jose Luis Blanco and Pranjal Kumar Rai. nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) wih kd-trees. `https://github.com/jlblancoc/nanoflann`, 2014.

[62] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[63] Ruben Martinez-Cantin. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research*, 15:3915–3919, 2014. URL `http://jmlr.org/papers/v15/martinezcantin14a.html`.

[64] Valentin Peretroukhin, Lee Clement, and Jonathan Kelly. Get to the point: Active covariance scaling for feature tracking through motion blur. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'15) Workshop on Scaling Up Active Perception*, Seattle, Washington, USA, May 30 2015.

[65] O.J. Woodman. An introduction to inertial navigation.

Appendix A

Matrices

A.1 Rotation Matrix

The roll-pitch-yaw euler angles rotation matrix, following a 1-2-3 convention, is given by:

$$\mathbf{C_k}\left(\phi_k, \theta_k, \psi_k\right) = \begin{bmatrix} \cos(\psi_i)\cos(\theta_k) & \cos(\psi_k)\sin(\theta_k)\sin(\phi_k) - \sin(\psi_k)\cos(\phi_k) & \cos(\psi_k)\sin(\theta_k)\cos(\phi_k) + \sin(\psi_k)\sin(\phi_k) \\ \sin(\psi_k)\cos(\theta_k) & \sin(\psi_k)\sin(\theta_k)\sin(\phi_k) + \cos(\psi_k)\cos(\phi_k) & \sin(\psi_k)\sin(\theta_k)\cos(\phi_k) - \cos(\psi_k)\sin(\phi_k) \\ \sin(\theta_k) & \cos(\theta_k)\sin(\phi_k) & \cos(\theta_k)\cos(\phi_k) \end{bmatrix}. \tag{A.1}$$

A.2 SE(3) Jacobian

To develop the Jacobian for the sensor model in Section 3.1.1, let us consider what happens to a lidar point if we perturb the camera pose by a small perturbation $\delta\epsilon \in \mathfrak{se}(3)$,

$$\mathbf{p}_{G,n}^{(k)} = \exp(\delta\epsilon^{\wedge})\mathbf{T}_{G,C}(t_n + t_d)\mathbf{T}_{C,L}\mathbf{p}_{L_n}^{(k)}, \tag{A.2}$$

$$= \exp(\delta\epsilon^{\wedge})\hat{\mathbf{p}}_{G,n}^{(k)}, \tag{A.3}$$

where $\mathbf{p}_{G,n}^{(k)}$ and $\mathbf{p}_{L_n}^{(k)}$ are the lidar point in the global and lidar frames respectively, $\mathbf{T}_{G,C}(t_n + t_d)$ is our estimated interpolated camera pose, and $\mathbf{T}_{C,L}$ is the lidar to camera transform. A $\hat{}$ over a quantity denotes its nominal value, with no perturbation.

Since $\delta\epsilon$ is a small perturbation, we can make the following approximation,

$$\mathbf{p}_{G,n}^{(k)} = (\mathbf{I}_4 + \delta\epsilon^{\wedge})\hat{\mathbf{p}}_{G,n}^{(k)}. \tag{A.4}$$

We can expand this as follows [44],

$$\mathbf{p}_{G,n}^{(k)} = \hat{\mathbf{p}}_{G,n}^{(k)} + (\hat{\mathbf{p}}_{G,n}^{(k)})^{\odot}\delta\epsilon, \tag{A.5}$$

$$(\hat{\mathbf{p}}_{G,n}^{(k)})^{\odot} = \begin{bmatrix} \mathbf{I}_3 & -(\hat{\mathbf{x}}_{G,n}^{(k)})^{\wedge} \\ \mathbf{0}^T & \mathbf{0}^T \end{bmatrix}, \tag{A.6}$$

where $\hat{\mathbf{p}}_{G,n}^{(k)} = \begin{bmatrix} \hat{\mathbf{x}}_{G,n}^{(k)} \\ 1 \end{bmatrix}$ (i.e., $\hat{\mathbf{x}}_{G,n}^{(k)}$ contains the top 3 entries of the vector $\hat{\mathbf{p}}_{G,n}^{(k)}$).

Equation (A.5) and A.6 contain a 4×6 matrix, but only the top 3 rows of the matrix are relevant to the perturbation's effect on the x-y-z coordinates of the lidar point. Ultimately, the Jacobian has the following form,

$$\mathbf{J}_n^{(k)} = \begin{bmatrix} \mathbf{I}_3 & -(\hat{\mathbf{x}}_{G,n}^{(k)})^{\wedge} \end{bmatrix}. \tag{A.7}$$

A.3 Miscellaneous $\mathfrak{se}(3)$ and $\mathrm{SE}(3)$ Operators

The $^{\curlywedge}$ operator is used in the construction of adjoint matrices. Its form is as follows [44],

$$\begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\phi} \end{bmatrix}^{\curlywedge} = \begin{bmatrix} \boldsymbol{\phi}^{\wedge} & \boldsymbol{\rho}^{\wedge} \\ \mathbf{0} & \boldsymbol{\phi}^{\wedge} \end{bmatrix}, \tag{A.8}$$

$$\boldsymbol{\phi}^{\wedge} = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix},$$

where $\boldsymbol{\rho}, \boldsymbol{\phi} \in \mathbb{R}^3$.

$\boldsymbol{\mathcal{J}}()$ is the left Jacobian operator.

$$\boldsymbol{\mathcal{J}}(\begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\phi} \end{bmatrix}) = \begin{bmatrix} \mathbf{J} & \mathbf{Q} \\ \mathbf{0} & \mathbf{J} \end{bmatrix}, \tag{A.9}$$

$$\mathbf{J}(\boldsymbol{\phi}) = \frac{\sin(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|}\mathbf{I} + (1 - \frac{\sin(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|})\frac{\boldsymbol{\phi}}{\|\boldsymbol{\phi}\|}\frac{\boldsymbol{\phi}^T}{\|\boldsymbol{\phi}\|} + \frac{1 - \cos(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|}\frac{\boldsymbol{\phi}^{\wedge}}{\|\boldsymbol{\phi}\|},$$

$$\begin{aligned}\mathbf{Q}(\begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\phi} \end{bmatrix}) = \frac{1}{2}\boldsymbol{\rho}^{\wedge} + \frac{\|\boldsymbol{\phi}\| - \sin\|\boldsymbol{\phi}\|}{\|\boldsymbol{\phi}\|^3}(\boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge} + \boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge} + \boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge}) - \frac{1 - 0.5\|\boldsymbol{\phi}\|^2 - \cos(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|^4}(\boldsymbol{\phi}^{\wedge}\boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge} + \boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge}\boldsymbol{\phi}^{\wedge} - 3\boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge}) \\ - 0.5(\frac{1 - 0.5\|\boldsymbol{\phi}\|^2 - \cos(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|^4} - 3\frac{\|\boldsymbol{\phi}\| - \sin\|\boldsymbol{\phi}\| - \frac{1}{6}\|\boldsymbol{\phi}\|^3}{\|\boldsymbol{\phi}\|^5})(\boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge}\boldsymbol{\phi}^{\wedge} + \boldsymbol{\phi}^{\wedge}\boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge}).\end{aligned}$$

The adjoint operator $\mathrm{Ad}\,()$ maps from SE(3) to $\mathbb{R}^{6\times6}$, and is constructed as follows,

$$\mathrm{Ad}\left(\begin{bmatrix} \mathbf{C} & \mathbf{r} \\ \mathbf{0} & 1 \end{bmatrix}\right) = \begin{bmatrix} \mathbf{C} & \mathbf{r}^{\wedge}\mathbf{C} \\ \mathbf{0} & \mathbf{C} \end{bmatrix}. \tag{A.10}$$