AUTOMATIC AND FEATURELESS Sim(3) CALIBRATION OF PLANAR
LIDARS TO EGMOTION SENSORS

by

Jacob Lambert

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

© Copyright 2017 by Jacob Lambert
Abstract

Automatic and Featureless Sim(3) Calibration of Planar Lidars to Egomotion Sensors

Jacob Lambert

Master of Applied Science
Graduate Department of Aerospace Science and Engineering
University of Toronto

2017

This thesis aims at generalizing the process of extrinsically calibrating two rigidly attached sensors on a mobile robot. Recent work in this area is revisited and its theory extended to the problem of recovering the Sim(3) transformation between a planar lidar and a monocular camera, where the scale of the camera trajectory is not known a priori. An efficient algorithm with only a single tuning parameter is implemented and studied. The robustness of the approach is tested on realistic simulations in multiple environments, as well as on data collected from a handheld sensor rig. Results show that, given a non-degenerate trajectory and a sufficient number of lidar measurements, the calibration procedure achieves millimetre-scale and sub-degree accuracy. Moreover, the method relaxes the need for specific scene geometry, fiducial markers, and overlapping sensor fields of view.
Acknowledgements

The author of this thesis would like to acknowledge his supervisor, Jonathan Kelly, for his invaluable guidance and support throughout this research. Special thanks also goes to Tim Barfoot for his review of this thesis. The members of the Space and Terrestrial Autonomous Robotics (STARS) Laboratory and of the Dynamic Systems Laboratory (DSL) also deserve mention for creating a pleasant and fulfilling work environment. From these laboratories, special mention goes to Lee Clement, Valentin Perethroukhin and Karime Pereida-Pérez for their support and friendship.
# Contents

1 Introduction ................................................. 1

2 Related Work .................................................. 4
   2.1 Target-based Calibration ................................. 4
   2.2 Calibration in the Wild ................................. 7

3 Theory .......................................................... 10
   3.1 Kinematics ............................................... 10
      3.1.1 SE(3) Transformation ............................... 10
      3.1.2 Sim(3) Transformation ............................. 13
   3.2 Entropy .................................................. 14
      3.2.1 Background ......................................... 14
      3.2.2 Point Cloud Entropy ............................... 17

4 Algorithm ....................................................... 20
   4.1 Entropy Computation Approximation .................... 20
   4.2 Camera Egomotion Estimation ........................... 24
      4.2.1 Asynchronous Sensor Streams ..................... 25
   4.3 Optimization ............................................ 26
      4.3.1 Gradient-based Optimization ...................... 26
      4.3.2 Gradient-free Optimization ....................... 28
   4.4 Implementation ......................................... 32

5 Simulations .................................................... 34
   5.1 Setup .................................................... 34
      5.1.1 Environment ....................................... 37
   5.2 Results .................................................. 40
      5.2.1 Cost Function Validation ......................... 40
      5.2.2 Global Optimization ............................ 42
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Simulation parameters: sensor design specification, sensor noise parameters and trajectory parameters. The trajectory is defined by 6 independent sinusoids, each with independent amplitude $A$ and frequency $\omega$.</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>Average absolute error and standard deviation, $\mu (\sigma)$, over ten unique trajectories, for the five simulation environments.</td>
<td>42</td>
</tr>
<tr>
<td>5.3</td>
<td>Average absolute error and standard deviation, $\mu (\sigma)$, on 10 temporally delayed data sets in the <em>Simple Room</em> environment, with and without temporal pre-calibration.</td>
<td>45</td>
</tr>
<tr>
<td>6.1</td>
<td>Calibration results for the <strong>overlapping</strong> case, for one data set segmented into four trial runs.</td>
<td>49</td>
</tr>
<tr>
<td>6.2</td>
<td>Calibration results for the <strong>non-overlapping</strong> case, for one data set segmented into three trial runs.</td>
<td>49</td>
</tr>
</tbody>
</table>
List of Figures

3.1 Diagram demonstrating the lidar, camera and global reference frames and the transformation between them .................................................. 11

4.1 Histogram of maximum eigenvalue for the covariance $\Sigma_i$ of each lidar point $x_i$ in a typical dataset. .............................................. 22

4.2 Entropy contribution, or negative of cost, as a function of Euclidean distance between a pair of lidar points, for three isotropic covariances $\Sigma$ with eigenvalue $\lambda$. .................................................. 22

4.3 Example of the linear relationship between the neighbourhood radius and parameter $k$, for isotropic covariances $\Sigma_i$ with eigenvalue $\lambda$. ............... 23

4.4 Flow chart of the ORB-SLAM algorithm ........................................... 24

4.5 One dimension of the numerical gradient of our cost function, estimated through forward differences, for multiple step sizes ...................... 27

4.6 Global optimization of the translational parameters through CRS and NM. 29

4.7 Global optimization of the rotational parameters through CRS and NM. 30

4.8 Global optimization of the scale parameters through CRS and NM (top) and progression of entropy minimization (bottom). The shift from CRS to NM occurs at the entropy spike around iteration 1750. ............. 31

5.1 Two views of a simple lidar simulation in a test environment to demonstrate the capabilities of the simulation. The FOV is larger than it appears as no ray (red) is drawn if no object is detected within the lidar maximum range of 10 meters .................................................. 35

5.2 The noisy, mean trajectory of our base sensor, with orientation shown every 100th pose ................................................................. 37

5.3 Simulation environments and base trajectory (blue) used to validate our method, with a cut away of the lidar measurements (red). .............. 39
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>Effect of variation of individual translational parameters on cost function, with others held constant. Offset error versus cost is displayed for each translational parameter.</td>
<td>40</td>
</tr>
<tr>
<td>5.5</td>
<td>Effect of variation of individual rotational parameters on cost function, with others held constant. Offset error versus cost is displayed for each rotational parameter.</td>
<td>41</td>
</tr>
<tr>
<td>5.6</td>
<td>Scale parameter versus cost with true value at scale $s = 1$ (in red), with SE(3) transform parameters held constant.</td>
<td>41</td>
</tr>
<tr>
<td>5.7</td>
<td>Sample result of the temporal pre-calibration approach, based on a trajectory generated in the <em>Simple Room</em> environment.</td>
<td>44</td>
</tr>
<tr>
<td>6.1</td>
<td>Hardware configurations used in experiments.</td>
<td>47</td>
</tr>
<tr>
<td>6.2</td>
<td>Cluttered office space in the MIT’s Stata Center used for data collection.</td>
<td>48</td>
</tr>
<tr>
<td>6.3</td>
<td>Visualization of a point cloud collected in the environment pictured in Figure 6.2, using initial values and mean values from Table 6.1. Note the smooth wall segments on the right side of Figure 6.3b.</td>
<td>51</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In both academia and industry, multisensor payloads have become the norm for mobile robotic systems. It is a great advantage to have sources of complementary information, primarily as it leads to a more detailed and accurate state estimate of the robot and its environment. Redundant information can also be desirable for long term autonomy, as individual sensors may become unreliable. Exteroceptive and interoceptive measurements from various sensors are therefore fused to accomplish tasks such as autonomous navigation[1], simultaneous localization and mapping[2, 3] and scene understanding [4], with high precision. The most commonly used interoceptive sensors include wheel odometry and inertial measurement units, while exteroceptive measurements are usually given by some type of camera system or laser rangefinders. Laser-based sensors are usually called lidars – an acronym for light detection and ranging. Some modern laser rangefinders utilize a single rotating laser to obtain slices of range measurements, called planar lidars or 2D lidars, while 3D lidars employ multiple rotating lasers at an angle from each other to provide three-dimensional point clouds.

Lidars are frequently used on multisensor platforms for their ability to estimate line-of-sight distance accurately. Visual sensors such as stereo cameras can recover this depth information using multiple view geometry, but they generally do not have the accuracy of lidars[5]. Inexpensive RGB-D cameras are also commonly used for building dense, coloured 3D maps, but they have limited range and are typically unreliable in natural, outdoor lighting. Conversely, modern 3D lidars are often employed to generate ground truth depth maps against which to evaluate the performance of visual depth estimation techniques[6]. The long range and lighting-invariance of the lidar sensor makes it suitable for both indoor and outdoor operation. For example, lidars have proven to be an unparalleled sensor choice for autonomous ground vehicles such as self-driving cars[7, 8]. A widespread choice is however to use both cameras and lidars as complementary sensors
Chapter 1. Introduction

on a robotics platform. This is because cameras provide dense visual information, but in the form of 2D images, which complicates the recovery of 3D information. Furthermore, in the case of monocular motion estimation, a scale ambiguity exists; the metric scale of the camera’s incremental motion cannot be recovered from monocular images alone. In contrast, lidars provide precise range information, ideal for creating depth maps and naturally complementing camera data in scene reconstruction\cite{9}.

The drawback to using multiple sensors is that to combine their respective information, knowledge of the coordinate transformations between the sensors is required. As an inaccurate transform necessarily leads to systematic error in the fused sensor data, this calibration process has to be performed carefully. Manually measuring the translational and rotational offset between sensors is far too inaccurate and also impractical, as the origins of the sensor coordinate systems are often embedded within the sensors themselves. Instead, it is preferable to use data-driven techniques, which leverage the data recovered by each sensor to estimate the coordinate transformation. However, an additional piece of information is then required to calibrate the sensors, usually in the form of some known feature in the environment. Conventional approaches to this problem will often require a specific calibration target, for example a checkerboard in the case of cameras, or textured planes for lidars. While largely successful at solving this problem, target-based approaches are impractical for long-term robotic deployments, during which the extrinsic calibration parameters may slowly drift or suddenly change; a calibration procedure applicable to an unknown environment and sensor configuration would then be necessary. State-of-the-art extrinsic calibration research attempts to tackle this problem of automatic calibration in the wild. Despite recent progress, a lack of generality persists in the literature, in the sense that techniques are too often designed for a specific sensor suite, require specialized scene structure, or placing restrictions on the sensors’ fields of view.

This thesis seeks to address these limitations by developing a generalized technique for the extrinsic calibration of a 2D lidar to a sensor capable of providing egomotion information. Our method builds directly upon previous work\cite{10,11} by representing the lidar point cloud as a Gaussian mixture model and minimizing its Rényi quadratic entropy in order to recover the coordinate transformation between the two sensors. In contrast to \cite{10,11}, which are concerned with estimating $SE(3)$ rigid body transformations between sensor pairs, we estimate $Sim(3)$ similarity transformations. This allows us to use a monocular camera as the egomotion sensor, despite the scale ambiguity in its trajectory reconstruction; the scale factor is estimated as part of the calibration process. The work presented in this thesis was in large part published in an academic conference\cite{12}.
The rest of this thesis is organized as follows: we first give an overview of the current literature addressing planar lidar to camera calibration in Chapter 2. We present the theory behind our approach in Chapter 3 and then discuss implementation details in Chapter 4. We provide validation for this method through simulation, presented in Chapter 5 and laboratory experiments, presented in Chapter 6. Finally, we discuss potential extensions to this work and conclude the thesis in Chapter 7.
Chapter 2

Related Work

In the following sections, we give an overview of the current state of extrinsic calibration research. We discuss primarily the literature addressing planar lidar to monocular camera calibration, as this is our focus. In Section 2.1, we examine methods which utilize specific targets for calibration, while in Section 2.2 we shift our attention towards methods that perform calibration in the wild.

2.1 Target-based Calibration

Extrinsic sensor calibration is an active area of research and a variety of approaches have been developed. Manual calibration is too unreliable to be used in practice, and instead, methods which use the data that the sensors collect are preferred. However, these so-called data-driven techniques imply fusing the sensor data \textit{without} knowing the coordinate transformation between them. As such, another piece of information must be used to assess the quality of the information fusion. The classic approach for extrinsic calibration involves a laboratory procedure making use of a known calibration object. The known environmental feature, in this case, is a calibration target of some kind, the choice of which highly depends on the sensors used. In this section, we discuss past research which addresses specifically planar lidar to camera calibration through the use of calibration targets.

Wasielewski and Strauss [13] were among the first to tackle this particular calibration problem. The target used in their work is composed of two planes at an angle of each other, like the pages of an open book. The key feature of this v-shaped target is that one side is white while the other is completely black; the method relies on detecting the line separating the two planes. In the camera frame, this represents the transition between black and white, an easily identifiable intensity gradient. Then, the known geometry
of the target makes it possible to find this line from the pattern of the lidar’s range measurements. To account for measurement noise, several of these line correspondences are used, from various viewpoints. Finally, a least-squares solution for the coordinate transformation relates the position of the line in each sensor frame. This line correspondence method is the basis of many calibration algorithms which use linear features, in the form of line or plane parameters. This specific approach and its v-shaped calibration target has been significantly improved over the years as new computer vision algorithms and robust optimization techniques have been developed. For example, Kwak et al. [14] utilized a novel technique to weight the numerous line correspondences to significantly increase accuracy and reduce computation time. An essentially equivalent algorithm, but with a modified target with several line features, was also found to yield good results[15].

It is now more common to see checkerboards as planar calibration targets. Recent work by Zhou [16] calibrates a 2D LIDAR with a camera by detecting the coplanar lines between three non-parallel checkerboards. This approach is closely related to previous work by Zhang and Pless [17], who were likely the first to use checkerboards for this specific extrinsic calibration problem. The algorithm’s ease-of-use and widely distributed toolbox implementation by Unnikrishnan and Hebert [18] have contributed to its popularity. Zhou [16] establishes a minimal solution to this line-to-plane correspondences problem, analysing both observability and numerical stability of their algorithm. They implement a variable permutation scheme to avoid singularities associated with the choice of rotation matrix parametrization. Their approach is one of the most recent implementations of this method and was shown to outperform its predecessors, [19], [20] and [21], while ensuring numerical stability. Subsequent work by Ying et al. [22] has further improved this approach by exploiting novel constraints on the rotation parameters and reworking the implementation to improve precision and computational efficiency. Many authors have also adapted this point-to-line correspondence approach for other sensors such as 3D lidars [23] and different camera configurations [24, 25, 26]. In particular, Geiger et al. [23] show that it is even possible, for most sensor configurations, to recover the coordinate transformation parameters in a single shot (one camera image and one lidar scan), if multiple targets are within line-of-sight.

Other notable methods use different calibration targets that utilizes the lidar’s accurate range measurements more directly. A checkerboard pattern with cut-out black squares used by García-Moreno et al. [27] or the sawtooth pattern used by Willis et al. [28] are examples of textured planes which create identifiable depth changes which can be used alongside edge detection algorithms. Another popular calibration target is a white, rectangular plane with a black line stretching horizontally from each end of the surface.
The location of the line feature on the target is trivial to compute from the camera perspective; the high contrast edge is easily detectable. In the lidar frame, this line can be detected through different methods. An approach proposed by Naroditsky et al. [29] is to use the reflectance values computed by modern lidars, though not all lidars provide this information. This approach was therefore adapted by Guo and Roumeliotis [30] to use only lidar range measurements and the geometric constraints created by the known target size.

The work presented thus far relies on detecting some linear feature in both sensor frames and matching it. They all make the same assumption: these features are visible simultaneously in both sensors’ field of view. This restricts the possible sensor configurations considerably, as their field of view must have a high degree of overlap. Overcoming this limitation is a significant challenge for these feature-based methods, but recent work by Bok et al. [31] attempts to relax this condition. A checkerboard is used as a planar target in the camera field of view, while either a plane or a line (intersection of two planes) must be visible to the lidar. The challenging assumption on the environment is that each plane, including the checkerboard, is be placed perpendicular or parallel to the others, such that each surface lies in the $xy$, $yz$ or $xz$ plane. This demonstrates how removing restrictions on sensor configuration forces other restrictions, this time on the environment. Nevertheless, with the right setup, they obtain robust performance in practice, but the issue becomes reproducibility. It is also worth noting that Bok et al.’s [31] approach is, to our knowledge, the only target-based approach which specifically deals with non-overlapping planar lidar to monocular camera calibration directly, without the need for an extra sensor.

Aside from typically requiring overlapping sensor fields of view, the target-based approaches discussed above are often extremely tedious to implement in practice due to poor generalization; these algorithms utilize specialized targets for the calibration of a unique sensor setup and any deviation from their specific procedure leads to problems. Perhaps the biggest drawback of target-based calibration is that it can only be performed in the lab. This is problematic for extended missions in autonomous robotics, where calibration parameters may change gradually or abruptly. For long term autonomy, it therefore is necessary to develop algorithms that calibrate sensors directly in the robot’s operational environment.
2.2 Calibration in the Wild

Calibration outside of the laboratory environment, or *in the wild*, is a challenging problem that has only recently been investigated. As a first step towards removing the need for specific calibration targets, several methods instead attempt to identify appropriate calibration geometry, or *features*, in the environment.

An obvious approach, which was found to be particularly viable in indoor environments by Yang et al. [32] and Moghadam et al. [33], is to pick out lines formed at the intersection of planar surfaces, again through depth discontinuities or edge detection algorithms. These approaches are reminiscent of the target-based feature-matching methods mentioned above. They also require the sensors to have overlapping fields of view, but recent work has attempted to relax this condition. Napier et al. [34] only assume that the same scene is *eventually* observed by each sensor. They first solve a view-matching problem, where lidar point clouds are projected to the camera frame, then compared directly with multiple camera images. When a point cloud and camera image are found to look at a scene with some degree of overlap, edges are again utilized for feature matching. Napier et al.'s [34] algorithm was developed specifically for the calibration of push-broom lidars (actuated planar lidars) to a multi-cameras system, and requires egomotion information. Furthermore, their optimization procedure is only robust with a good initial guess of the lidar-camera transform, as their cost function is only locally convex; this captures the difficulty of feature matching asynchronous data. Despite its specificity, Napier et al.'s [34] method was shown to be quite successful at solving the difficult problem of calibrating non-overlapping sensors and it inspired implementations for other sensor pairs [35, 36, 37].

Lines are not the only linear feature which can be utilized for *in-situ* calibration; planar surfaces can also be detected in the environment and used as targets. Rehder et al. [38], [39], examine entire camera and lidar dataset then extract and match planar surfaces. Their method requires egomotion information, but poses no restriction on sensor configuration and can recover a potential temporal offset between sensor data streams. Rehder et al.'s [39] work makes few assumptions about the environment and represents the state-of-the-art in terms of generalizability. While they implement and test their algorithm for a variety of sensor combinations, they do not directly deal with the planar lidar to monocular camera calibration problem; they require a second camera or an inertial measurement unit as intermediate for their accurate egomotion information.

Some methods deviate from these feature-based techniques and instead perform sensor-to-sensor calibration using the overall appearance of the environments. This has the advantage of putting fewer constraints on the calibration environment and the sensor
configuration. For example, Scott et al. [40] calibrate multiple cameras with push-broom planar lidars through mutual information, matching pixel intensity to lidar reflectance values. Overlapping fields of view is not required, but a calibrated stereo setup providing egomotion must be used to constrain the motion of the lidar prior to calibration. This approach was later improved by implementing an automatic scene selection scheme. Scott et al.'s [41] algorithm now selects a subset of scenes in a data set, which are deemed ideal for the purposes of maximizing mutual information. Scenes that contain simple structures such as planar surfaces are preferred over comparatively featureless or noisy settings. For autonomous ground vehicles this translates to urban settings being preferred over barren landscapes or noisy environments dominated by vegetation or foliage. Clever scene selection improves accuracy but more significantly the computational efficiency. Their method matches closely work by Pandey et al. [42], which also computes the mutual information of pixel intensities and lidar reflectance values, but to calibrate other sensor setups: a 3D lidar to an omnidirectional camera, a time-of-flight camera to a monocular camera or a 2D lidar to a monocular camera. Pandey et al.'s [42] research, while readily adaptable to a larger class of sensors, unfortunately requires overlapping sensor fields of view. Taylor and Nieto [43] modified this approach for application in mining, where hyperspectral cameras and lidars are used to determine ore concentration in quarries. The laser rangefinder used in this work is part of an immobile surveying station and cannot recover surface reflectance information, so they adapted the method to use range information only. The structure of these mining pits is such that surface normals can be reliably estimated from a dense point cloud. The surface normal at each point is used as pixel intensity for a projection of a lidar point cloud to a 2D image. The coordinate transformation which optimizes the normalized mutual information between the camera image and lidar image is then determined.

Appearance-based methods are also commonly used to calibrate two sensors through the intermediary of a base sensor. Zhao et al. [44] calibrate multiple lidars and cameras on a mobile platform by referring to a fiducial coordinate system. This same principle was applied by Maddern et al. [41] to calibrate planar lidars to 3D lidars by projecting their point cloud in the coordinate system of a GPS unit on the robot, used here as base sensor. Building on previous work by Sheehan et al. [45], [46], they develop a point cloud reconstruction technique through scene entropy minimization, a key component of the approach presented in this thesis which we discuss in detail in Section 3.2.

Also notable are ego-motion based approaches which can be applied in arbitrary settings. It was shown by Brookshire and Teller [47], [48] that for sensors capable of recovering their own incremental motion, it is possible to solve for the geometric trans-
formation between sensors without the need for specific environmental features. The calibration parameters are however only observable if the sensor trajectory spans $SE(3)$ space, meaning the robot translates and rotates about every axis of an arbitrary coordinate system, while varying its velocity. Their observability analysis provides insight into the type of trajectories that guarantee the observability of calibration parameters; their guidelines for developing generalized trajectories were adhered to in this work.
Chapter 3

Theory

In this chapter, we define an algorithm for the automatic calibration of a camera and a planar lidar, rigidly attached and mounted on a mobile platform. Formally, the goal of this algorithm is to estimate the set of transformation parameters from the lidar frame \( \mathcal{F}_L \) to the camera frame \( \mathcal{F}_C \). In 3D space, this represents an \( SE(3) \) rigid-body transformation between the sensor coordinate frames, but a scale ambiguity is introduced through the use of a monocular camera; a \( Sim(3) \) similarity transformation consists of an \( SE(3) \) transformation with an additional, uniform scaling as shown in Equation (3.16).

The follow sections discuss the mathematical foundations of the algorithm. First, we derive a chain of geometric transformations to project sensor data from several distinct reference frames to a single, global frame. In Section 3.1.1, we show how to do this in \( SE(3) \), then in Section 3.1.2, we demonstrate how to augment this method for \( Sim(3) \). In Section 3.2.1, we explain the intuition behind using entropy as a cost function in the search for optimal calibration parameters, then show in Section 3.2.2 a way to calculate the entropy of a lidar point cloud.

3.1 Kinematics

3.1.1 SE(3) Transformation

The objective of the algorithm is to estimate a vector of calibration parameters. In 3D, there are 6 transformation parameters which define the change of reference frames:

\[
\Xi = [x_{C,L} \ y_{C,L} \ z_{C,L} \ \phi_{C,L} \ \theta_{C,L} \ \psi_{C,L}]^T,
\]  

(3.1)
where $[x_{C,L} \ y_{C,L} \ z_{C,L}]^T$ are the three translational parameters along the axes that define the reference frame, and $[\phi_{C,L} \ \theta_{C,L} \ \psi_{C,L}]^T$ which are the respective roll-pitch-yaw Euler angles and the rotational parameters of the transformation $T_{C,L}$, shown in Figure 3.1. This rotation parametrization was chosen for clarity and consistency with previous work\cite{11}, but is not a critical design choice; we will use eventually use a homogeneous pose representation and never revert back to Euler angles, thus avoiding the singularities associated with this parametrization.

Figure 3.1: Diagram demonstrating the three reference frames and the transformation between them. The lidar makes range measurements, capturing the distance between the lidar frame origin and points in the world frame (red dots). The camera takes images of the global frame, projecting the observation points (stars) through the camera frame and into the image frame. The location of the global frame features in lidar and camera frames are related by $T_{C,L}$.

Our sensor suite consists of a camera that collects 2D images of the environment, hence projecting data from our global frame, $F_G \rightarrow G$, to the camera frame, $F_C \rightarrow C$. In this work, we assume that from these images, we are able to extract the sensor’s egomotion, up to scale. For cameras, there are several algorithms capable of obtaining this information, one of which is discussed in Section 4.2. It is also true that any sensor capable of resolving its own motion could be used in place of a camera. This egomotion sensor, henceforth referred to as the base sensor, is assumed to provide a set of K, 6 degree-of-freedom
poses, \( Y \), where:

\[
Y = \{y_1, y_2, \ldots, y_K\},
\]

\[
y_k = [x_k \ y_k \ z_k \ \phi_k \ \theta_k \ \psi_k]^T.
\]

Each pose \( y_k \) has an associated timestamp \( t_k \) and covariance matrix \( Q_k \in \mathbb{R}^{6 \times 6} \). The covariance captures the uncertainty in the estimate of the camera pose, a direct consequence of obtaining this information from imperfect sensor measurements. The other sensor, the lidar, provides a set of \( K \times N \), 2D distance observations relative to the lidar frame, \( Z \), where:

\[
Z = \{z_1, z_2, \ldots, z_K\},
\]

\[
z_k = \{z_k^{(1)}, z_k^{(2)}, \ldots, z_k^{(N)}\},
\]

\[
z_k^{(n)} = \begin{bmatrix} x_k^{(n)} & y_k^{(n)} \end{bmatrix}^T,
\]

and each point \( z_k^{(n)} \) has associated timestamp \( t_k \); here we have assumed that the lidar and camera measurements are temporally aligned. Lidar measurements may also be given in polar coordinates, in the form of a range and bearing measurement, \( [r_k^{(n)}, b_k^{(n)}]^T \), in which case Equation 3.6 becomes:

\[
z_k^{(n)} = \begin{bmatrix} r_k^{(n)} \cos(b_k^{(n)}) & r_k^{(n)} \sin(b_k^{(n)}) \end{bmatrix}^T.
\]

Next, we adopt a homogeneous vector representation for the points in the lidar frame \( F_{-L} \) as:

\[
p_k^{(n)} = \begin{bmatrix} x_k^{(n)} & y_k^{(n)} & 0 & 1 \end{bmatrix}^T.
\]

We also express each camera pose, \( y_k \), as a homogeneous transformation from the camera frame \( F_{-C} \) to the fixed global frame \( F_{-G} \), given matrix \( T_{G,C_k} \in \mathbb{R}^{4 \times 4} \):

\[
T_{G,C_k} = \begin{bmatrix} C_k & t_k \\ 0^T & 1 \end{bmatrix},
\]

where \( t_k \) and \( C_k \) are the respective translation vector and rotation matrix associated with pose \( y_k \). The vector representing translation is given by:

\[
t_k = \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix}.
\]
The rotation matrix $C_k \in \mathbb{R}^{3 \times 3}$ follows a Roll-Pitch-Yaw ($1-2-3$) Euler angle convention with full form given in Appendix A.1. Similarly we use a homogeneous transformation matrix representation for our vector of calibration parameters, $T_{C,L} \leftarrow \Xi$, where:

$$T_{C,L} = \begin{bmatrix} C_{C,L} & t_{C,L} \\ 0^T & 1 \end{bmatrix}. \quad (3.11)$$

We are trying to find the true transformation matrix $T_{C,L}$ and assume we have a reasonable guess, which can be obtained by manual inspection of the sensor platform, for example. The kinematic chain with which we can estimate the position of a lidar point in the global frame through the camera pose and calibration parameter estimates is therefore given by function $h(p^{(n)}_{L_k}, y_k, \Xi)$:

$$\hat{p}^{(n)}_{G,k} = h(p^{(n)}_{L_k}, y_k, \Xi) = T_{G,C_k}T_{C_k,L_k}p^{(n)}_{L_k}. \quad (3.12)$$

We omit the homogeneous part such that for each point $\hat{p}^{(n)}_{G,k}$ we have a corresponding point $\hat{x}^{(n)}_{G,k}$:

$$\hat{x}^{(n)}_{G,k} = \begin{bmatrix} x^{(n)}_k \\ y^{(n)}_k \\ z^{(n)}_k \end{bmatrix} \leftarrow \begin{bmatrix} x^{(n)}_k \\ y^{(n)}_k \\ z^{(n)}_k \end{bmatrix} = \hat{p}^{(n)}_{G,k}. \quad (3.13)$$

We use the Jacobian of this model and the camera pose covariance to obtain a covariance matrix for each lidar point in the world frame, $\Sigma^{(n)}_k \in \mathbb{R}^{3 \times 3}$, where:

$$\Sigma^{(n)}_k = J^{(n)}_k Q_k J^{(n)}_k^T, \quad J^{(n)}_k = \frac{\partial h(x^{(n)}_{L_k}, y_k, \Xi)}{\partial y_k}. \quad (3.14)$$

The full form of the Jacobian, $J^{(n)}_k \in \mathbb{R}^{3 \times 6}$, is given in Appendix A.2. We obtain a set of $M$ 3D points $\hat{x}_m \in \hat{X}$ expressed in the global frame, each with an associated covariance $\Sigma^{(n)}_k \in \mathbb{R}^{3 \times 3}$ and timestamp $t^{(n)}_k$.

### 3.1.2 Sim(3) Transformation

For calibration with a monocular camera or other sensor whose egomotion information contains a scale ambiguity, we can add a scale factor $s$ to the vector of calibration parameters:

$$\Xi = [x_{C,L} \ y_{C,L} \ z_{C,L} \ \phi_{C,L} \ \theta_{C,L} \ \psi_{C,L} \ s]^T. \quad (3.15)$$
Chapter 3. Theory

The matrix which replace the transform in Equation (3.11) then becomes a similarity transformation under this set of parameters:

\[ T_{C,L} = \begin{bmatrix} C_{C,L} & t_{C,L} \\ 0^T & 1/s \end{bmatrix} \]  

(3.16)

This also has an impact on the jacobian \( J_k^{(n)} \) in Equation (3.14), as shown in Appendix [A.3].

### 3.2 Entropy

#### 3.2.1 Background

This algorithm attempts to reconstruct the lidar point cloud in the global frame by minimizing its entropy. We will see that entropy, as a property of a probability density, will become our metric to assess the crispness of our point cloud, which, we hypothesize, will be optimized when the calibration parameters are at their true values. In this section, we provide the intuition behind this choice through some mathematical and historical background on entropy.

Given finite and discrete probability density, \( P \) and \( Q \), each with \( \Omega \) possible events with respective probability \( p_i \) and \( q_i \) for \( i \in \{1, 2, ..., \Omega\} \), we denote \( P = (p_1, p_2, ..., p_\Omega) \) and \( Q = (q_1, q_2, ..., q_\Omega) \). In information theory, the entropy of \( P \) is written as \( H(P) \) and is said to quantify the uncertainty associated with this probability density. It was introduced by Shannon[49] as:

\[
H_{\text{SHANNON}}(P) = \sum_{i=1}^{\Omega} p_i \log_b \frac{1}{p_i}
\]

(3.17)

for some logarithm base \( b \), commonly 2 or 10. Due to the monotonic nature of the logarithm, the choice of base impacts little but the numerical output of this entropy formula; a choice of \( b = 10 \), as originally used by Hartley [50], produces an entropy value in Hartleys, whereas the unit associated with \( b = 2 \) is called the Shannon or the bit. This information measure was shown to satisfy several interesting postulates[51]. Shannon stated that to properly quantify the amount of information contained in probability density \( P \), an information measure, in this case entropy, should satisfy three properties[52]:

1. **Additivity**: The entropy of the direct product of two probability densities must
Chapter 3. Theory

be additive:

\[ H_{\text{SHANNON}}(P \ast Q) = H(P) + H(Q). \]  

(3.18)

2. **Continuity**: \( H(P) \) must be continuous in \( p_i \).

3. **Uniform property**: If all events are equally probably, \( p_i = 1/\Omega \ \forall \ i \in \{1, 2, ..., \Omega\} \), then \( H(P) \) must be a monotonically increasing function of \( \Omega \).

In the above, \( P \ast Q \) denotes the probability density consisting of all \( p_iq_j \) for \( i = 1, 2, ..., \Omega \) and \( j = 1, 2, ..., \Psi \). Equation (3.18) reflects the deep result that the entropy of two independent probability densities is simply equal to the sum of their entropies. In 1961, Rényi showed that a much larger family of functions satisfies almost all of these properties, with the exception of some special cases of additivity\(^\text{53}\). He defined this infinite set of functions as the **entropy of order** \( \alpha \), given by

\[ H_\alpha(P) = \frac{1}{1-\alpha} \log \left( \sum_{i=1}^{N} p_i^\alpha \right), \ \alpha > 0, \ \alpha \neq 1, \]  

(3.19)

which reduces to Shannon entropy in the limit where \( \alpha \to 1 \). The entropy of order \( \alpha \), as well as other later proposed entropy measures (Havrda-Charvat, Kapur) do not possess all the properties of an information measure as defined by Shannon, but they were shown to be equivalent from the perspective of optimization\(^\text{54}\). In other words, they are equally acceptable metrics for the purpose of maximizing or minimizing the entropy of a probability density.

The intuition behind entropy in information theory is that it quantifies the uncertainty associated with drawing a measurement from a probability distribution. For example, the entropy of a one-dimensional Gaussian distribution grows with its standard deviation; a Gaussian with small standard deviation implies that a random sample is very likely to fall within a small range of values. As the standard deviation increases, the less information we have about where a random sample from the distribution might be located. In the limit where the standard deviation goes to infinity, we obtain a uniform distribution, which maximizes the entropy of simple 1-dimensional distributions.

For multivariate distributions, it is more difficult to visualize this concept of entropy. There is however a physical intuition behind this notion that entropy is a measure of compactness or uniformity. After all, the concept of entropy first arose in physics, specifically in an attempt to quantify the notion of equilibrium. In that field, it is now often interpreted as the measure of disorder in a system, in that a system at equilibrium, or at maximal entropy, will have attained the highest degree of uniformity. This natural
law is embodied in the second law of thermodynamics which states that the entropy of a closed system cannot decrease. As an example of the physical manifestation of entropy maximization, consider placing a gas in an otherwise empty box. Regardless of the initial properties and spatial distribution of the gas, it is only natural that it will distribute itself uniformly, therefore normalizing the gas’ density and pressure throughout the box. It is standard practice in statistical mechanics to treat a set of particles, in particular a gas, as a probability density. As such, there is direct physical intuition relating how entropy characterizes a probability density.

The relationship between thermodynamic entropy and information theoretic entropy is definitive when given a more rigorous statistical thermodynamics definition. Gibbs’ entropy is used to obtain the macroscopic properties of a system based on its distribution of particles, called microstates. If a closed system with \( \Omega \in \mathbb{Z}^+ \) microstates, each with probabilities given by \( \mathcal{P} = (p_1, p_2, ..., p_\Omega) \), then the entropy of the system, classically denoted \( S \), is given by:

\[
S_{\text{GIBBS}}(\mathcal{P}) = -k_B \sum_{i=1}^{\Omega} p_i \ln p_i. 
\] (3.20)

We can see this is simply Shannon entropy with Euler’s number as logarithmic base, and scaled by the Boltzmann constant, \( k_B \), giving thermodynamic entropy units of Joules per Kelvin. It is often fair to assume, especially for large systems, that each microstate is equally probable, which gives rise to Boltzmann’s equation for entropy:

\[
S_{\text{BOLTZMANN}}(\mathcal{P}) = k_B \ln \Omega. 
\] (3.21)

Maximizing entropy therefore usually implies maximizing \( \Omega \), the number of microstates available to your system. As such, the macroscopic properties of a system are those which maximize the number of possible particle configurations (microstates) that lead to that macrostate. Going back to our physical example, we can now see why a system with uniform properties and spatial distribution maximizes Equation (3.21); with all particles having the same properties, any two particle can be interchanged, and there are several microstates that result in the same macrostate. Conversely, states of small entropy represents a compact, highly structured arrangement of particles.

Equation (3.21) coincides very closely with Hartley entropy, one of the earliest entropy measures proposed in the context of information theory. Hartley used it to define the amount of information revealed by randomly picking a sample \( p_i \) from a uniform
probability density. Hartley entropy is defined as:

\[ H_{\text{HARTLEY}}(\mathcal{P}) = \log_b |\mathcal{P}| = \log_b \Omega. \]  

(3.22)

Boltzmann and Gibbs’ work in statistical mechanics preceded Hartley, Shannon and Rényi’s work in information theory, and while the latter lacks physical intuition, the relationship between the two was obvious enough to Shannon that he called his novel information measure entropy.

### 3.2.2 Point Cloud Entropy

In this context, we want to use entropy in order to maximize the compactness or crispness of the estimated point cloud \( \hat{\mathbf{X}} \). The intuition behind using this metric to estimate \( \mathbf{T}_{C,L} \) follows from the assumption that the surfaces in the environment are structured 2D manifolds as opposed to diffuse elements. If the lidar points are correctly transformed to the world frame, or in other words, we have an accurate estimate of \( \mathbf{T}_{C,L} \), they should all lie on the smooth surfaces that form the environment. Previous work\[^{[55, 11]}\] hypothesized this arrangement of lidar points represents a low state of entropy that is not randomly attainable, and that an incorrect transform can only increase entropy with respect to this configuration.

To calculate the entropy of our lidar point cloud in the global frame, we must first represent it as a probability density function. One way of doing this is to use Parzen window density estimation\[^{[56]}\], a data-interpolation technique used to estimate the probability density function associated with a set of events, \( \{ \hat{x}_1, \hat{x}_2, ..., \hat{x}_M \} \in \hat{\mathbf{X}} \). Given a kernel function, it estimates the value of the probability density function at \( x \) by looking at all the events \( \hat{x}_i \) that fall within a window of \( x \). Given some kernel function \( K(\cdot, \cdot) \) and given that \( x \) is \( D \)-dimensional, the Parzen window is defined by bandwidth parameter \( h \) and the probability density function is given by:

\[ p(x) = \frac{1}{M} \sum_{i=1}^{M} \frac{1}{h^D} K \left( \frac{x - \hat{x}_i}{h} \right). \]  

(3.23)

A suitable kernel for our application is the Gaussian kernel as we assume that the noise on both the camera poses and lidar measurements is normally distributed. From Equation (3.14), the noise on the camera pose associated with a \( \hat{x}_i \) is \( \Sigma_i \). Conventional lidars have small, normally distributed noise on the range measurement, but there are several other, less documented sources of noise\[^{[57]}\]. These are not usually modelled, and so a
reasonable approximation is to add isotropic noise, \( \sigma^2 \mathbf{I} \) to capture the range uncertainty. Our Parzen window probability density function essentially becomes a Gaussian mixture model with bandwidth defined by a covariance for which both sensors contribute:

\[
p(x) = \frac{1}{M} \sum_{i=1}^{M} \mathcal{N}(x - \hat{x}_i, \Sigma_i + \sigma^2 \mathbf{I}),
\]

where \( \mathcal{N}( \cdot ) \) denotes a multivariate Gaussian distribution. We can now substitute the Gaussian mixture model from Equation (3.24) in our formula for entropy, though it is necessary to use an integral instead of a sum, to account for the fact that \( x \) is a continuous variable. Using Shannon’s definition of entropy with this probability density function makes the integral impossible to solve analytically and numerical solutions are complex and computationally expensive. Fortunately, our aim is to minimize the entropy, and as we discussed, several other definitions of entropy are equivalent to Shannon’s in the sense of optimization\[54\]. Motived by previous work\[55, 46, 45, 11\], an appealing option is Rényi’s entropy of order \( \alpha \), given by Equation (3.19). Rényi quadratic entropy (RQE) corresponds to an \( \alpha = 2 \) and is given by:

\[
H(\hat{X}) = -\log \int p(x)^2 dx.
\]

(3.25)

The choice of \( \alpha = 2 \) is made to trivialize the evaluation of the indefinite integral which, for other values of \( \alpha \), is particularly difficult. Substituting Equation (3.24) in Equation (3.25) we obtain:

\[
H(\hat{X}) = -\log \int \left( \frac{1}{M} \sum_{i=1}^{M} \mathcal{N}(x - \hat{x}_i, \Sigma_i + \sigma^2 \mathbf{I}) \right)^2 dx
\]

(3.26)

\[
= -\log \left( \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \int \mathcal{N}(x - \hat{x}_i, \Sigma_i + \sigma^2 \mathbf{I}) \mathcal{N}(x - \hat{x}_j, \Sigma_j + \sigma^2 \mathbf{I}) dx \right)
\]

(3.27)

The integral on the last line is simply the convolution of two Gaussian distributions, which has a closed-form solution:

\[
\int \mathcal{N}(x - \mu_1, \sigma_1^2) \mathcal{N}(x - \mu_2, \sigma_2^2) dx = \mathcal{N}(\mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2) = \mathcal{N}(\mu_2 - \mu_1, \sigma_1^2 + \sigma_2^2)
\]

(3.28)
From this follows an analytic solution for the entropy of the point cloud:

\[
H(\hat{X}) = -\log \left( \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \mathcal{N}(\hat{x}_i - \hat{x}_j, \Sigma_i + \Sigma_j + 2\sigma^2 I) \right)
\] (3.29)

Therefore, to calculate the entropy of our probability density function, we must compute all pairwise entropy contributions; this amounts to the exponential of the squared Mahalanobis distance between pairs of points, scaled by the coefficient in front of the exponential in the Gaussian.
Chapter 4

Algorithm

The implementation details of the calibration algorithm presented in this thesis are discussed in the following sections. First, in Section 4.1 we develop a computationally efficient approximation for the cost function in Equation (3.29). Then, in Section 4.2 we discuss our choice of egomotion estimation algorithm for monocular cameras, ORBSLAM 2.0 [58,59]. In Section 4.3 we discuss the choice of optimization routines used to minimize our cost function. The implementation is then summarized in the pseudocodes of Section 4.4.

4.1 Entropy Computation Approximation

This section addresses the issue of the computational efficiency of the cost function derived in Section 3.2.2. For clarity, we restate that the entropy of a point cloud with $M$ points, $\hat{x}_i$, each with covariance, $\Sigma_i + \sigma^2 I$, is given by:

$$H(\hat{X}) = -\log \left( \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \mathcal{N}(\hat{x}_i - \hat{x}_j, \Sigma_i + \Sigma_j + 2\sigma^2 I) \right)$$  \hspace{1cm} (4.1)

It is immediately apparent by the double sum that the computation time scales as $O(M^2)$. With a typical lidar scanning at 40 hz with 720 beams per scan, the cloud accumulates nearly one million points in 30 seconds; the entropy computation could quickly become intractable for large data set. Following previous work using RQE [11], we make several improvements to the $M^2$ cost function above: first, notice that both sums run through all indices, yet we have:

$$\mathcal{N}(\hat{x}_i - \hat{x}_j, \Sigma_i + \Sigma_j + 2\sigma^2 I) = \mathcal{N}(\hat{x}_j - \hat{x}_i, \Sigma_j + \Sigma_i + 2\sigma^2 I),$$  \hspace{1cm} (4.2)
so we are effectively double-counting pairwise entropy contributions. We can simplify the cost function by changing the second sum appropriately:

\[
H(\hat{X}) = -\log \left( \frac{1}{M^2} \sum_{i=1}^{M} \sum_{j=1}^{M} \mathcal{N}(\hat{x}_i - \hat{x}_j, \Sigma_i + \Sigma_j + 2\sigma^2 I) \right) \quad (4.3)
\]

\[
= -\log \left( \frac{2}{M^2} \sum_{i=1}^{M} \sum_{j=i}^{M} \mathcal{N}(\hat{x}_i - \hat{x}_j, \Sigma_i + \Sigma_j + 2\sigma^2 I) \right). \quad (4.4)
\]

Then, we can remove the monotonic contribution of the coefficient and logarithm as this will not affect the optimization. Our cost, \( C(\hat{X}) \), an approximation of \( H(\hat{X}) \), is then:

\[
C(\hat{X}) = -\sum_{i=1}^{M} \sum_{j=i}^{M} \mathcal{N}(\hat{x}_i - \hat{x}_j, \Sigma_i + \Sigma_j + 2\sigma^2 I) \quad (4.5)
\]

A more meaningful improvement, proposed in [11], is made possible when we consider that the cost quickly becomes negligible as the distance between point pairs increases. As we are calculating Manahalobis distances, the cost also depends on the covariance associated with the point pair, and not only on the distance between them. In a typical dataset, the majority of points have a small covariance, and therefore small maximum eigenvalue. In Figure 4.1, we show how the majority of the point’s covariances have associated maximum eigenvalue of only a few millimeters. In this case, the resulting cost function decreases very quickly, as illustrated in Figure 4.2, where we show the value of the cost function as the distance between point pair grows. Since the cost function depends on the covariance, we plot the cost for a few isotropic covariances, where \( \Sigma = \lambda I \). Even for the largest values of \( \lambda \), the cost decays quickly; when points are farther than half a meter apart, they contribute negligible entropy. Figure 4.2 also shows that points that contribute the most entropy are those that have small covariance and are nearby. As such, we can drastically reduce the computational cost of the algorithm by only considering nearby points. By using a k-dimensional tree (k-d tree) to store our point cloud, we can efficiently query points within some radius of each other. The computational complexity of building this k-d tree is \( O(M \log M) \), which we only do once, while querying the data structure is, at worst, \( O(\log M) \), which is a considerable improvement over our \( O(M^2) \) cost function.

The choice of using a k-d tree makes it necessary to pick an appropriate neighbourhood size. Since the cost function decays faster as the covariance grows, it is sensible to define
Chapter 4. Algorithm

Figure 4.1: Histogram of maximum eigenvalue for the covariance $\Sigma_i$ of each lidar point $\hat{x}_i$ in a typical dataset.

Figure 4.2: Entropy contribution, or negative of cost, as a function of Euclidean distance between a pair of lidar points, for three isotropic covariances $\Sigma$ with eigenvalue $\lambda$. 

Chapter 4. Algorithm

Figure 4.3: Example of the linear relationship between the neighbourhood radius and parameter $k$, for isotropic covariances $\Sigma_i$ with eigenvalue $\lambda$.

The radius based on this covariance. Given a pair of points, $\hat{x}_i$ and $\hat{x}_j$, we can say:

$$N(\hat{x}_i - \hat{x}_j, \Sigma_i + \Sigma_j + 2\sigma^2 I) \approx 0 \text{ if } \|\hat{x}_i - \hat{x}_j\| \leq k(\lambda_{\text{max}}(\Sigma_i) + \lambda_{\text{max}}(\Sigma_j) + 2\sigma^2),$$

where $\lambda_{\text{max}}(\Sigma)$ denotes the largest eigenvalue of $\Sigma$, and $k$ is a tuning parameter that allows us a conservative approximation. The neighbourhood radius would then be given by:

$$R = k(\lambda_{\text{max}}(\Sigma_i) + \lambda_{\text{max}}(\Sigma_j) + 2\sigma^2),$$

for some value of $k$ which is selected to make the inequality in Equation (4.6) hold. Figure 4.3 plots this radius as a function of $k$ for the same values as those in Figure 4.2 allowing for easy comparison. By inspection, we can see that $k = 2$ is an appropriate choice as it cuts off only the tail end of the cost function.

A conservative and practical adjustment to the neighbourhood radius would be to define it using only the maximum eigenvalue of both covariances:

$$R = 2k(\max(\lambda_{\text{max}}(\Sigma_i), \lambda_{\text{max}}(\Sigma_j)) + \sigma^2).$$

As outlined in Section 4.4, this allows us to efficiently avoid double counting and limits our k-d tree queries, which is the most computationally expensive operation in the algorithm, after building the k-d tree.
4.2 Camera Egomotion Estimation

It is an assumption of the algorithm that the base sensor can determine its own egomotion in the form of relative, incremental poses accurate up to scale. The algorithm can therefore accommodate several sensors, but this complicates our calibration pipeline as we must now be concerned with the implementation of an appropriate algorithm to retrieve egomotion information. As we are primarily concerned with monocular cameras in this work, we briefly discuss ORB-SLAM below, which was used for motion estimation to produce the results presented in Chapter 6.

ORB-SLAM is an open-source, feature-based SLAM system\textsuperscript{[58]}. It was recently improved and extended to include stereo or depth information when available\textsuperscript{[59]}. This particular algorithm was chosen as it is the current state-of-the-art in the field of monocular motion estimation, shown to outperform its predecessors LSD-SLAM\textsuperscript{[60]} and PTAM\textsuperscript{[61]}.

In Figure 4.4 the authors of \textsuperscript{[58]} give an overview of their system. The algorithm extracts unique ORB features to perform frame-to-frame bundle adjustment, as well as place recognition within a global map for relocalization and loop closure. They maintain a local map and a corrected global keyframe map and use both to obtain reliable egomotion information. Real-time use is made possible on large data set by avoiding redundancy of information in the map. The ORB features are fast to compute and show good viewpoint invariance.

![Flow chart of the ORB-SLAM algorithm](image)

Figure 4.4: Flow chart of the ORB-SLAM algorithm\textsuperscript{[58]}.

The details of the ORB-SLAM implementation are not particularly relevant to this work. The precision of the algorithm’s output is however critical, as poor estimates of the
camera pose $T_{G,C_k}$ leads to a noisy global point cloud as shown by Equation \((3.12)\). The output of ORB-SLAM is the keyframe map and incremental poses in the form of $T_{C_k,G}$. No covariance information is given, implying we must infer precision of incremental poses from their experimental results. In their tests, ORB-SLAM is shown to perform well on a variety of data sets and environments. In our experience, it performs robustly in well-lit indoor environments, when the camera motion is slow enough to avoid motion blur. As such, we followed these guidelines when generating camera trajectories. The authors of ORB-SLAM show peek performance when looking at the same scene from different viewpoints, but we found this criteria to be too restrictive when it comes to in-situ calibration. The RMSE error on all keyframes is reported in their paper, which makes it difficult to estimate a covariance on an individual pose. Upon further examination of their results, we find that a standard deviation of 0.01 meters and 0.025 radians (approximately 1.5 degrees) on the each camera pose sufficiently reflects ORB-SLAM’s precision.

### 4.2.1 Asynchronous Sensor Streams

We have previously assumed that sensors gather data simultaneously, such that we have a lidar scan and a camera pose at each time stamp. In reality, there are many factors which can cause asynchronous data streams, such as sensors operating at different frame rates. This can be addressed fairly easily, but a more prevalent issue is that the internal clocks of the two sensors may be uncalibrated, creating a bias in the published time stamps. There exist software to calibrate internal clocks relative to a central clock (e.g. a computer clock), but misalignments on the order of milliseconds are difficult to detect as data transfer rates are also on order of milliseconds. This offset is often ignored as it is negligible for platforms with slow, smooth dynamics, but for other applications it must be accounted for. As previously discussed, some chose to include a temporal offset in their calibration parameter vector to address this issue, but this complicates the calibration procedure significantly.

In this work, we attempt to use the nature of the sensors to minimize temporal offset between sensors. Planar lidars have relatively slow scan rate, for example 0.025 ms (40 hz) for the Hokuyo UTM-30LX, which sets a upper bound on the data acquisition rate. On the other hand, the camera data can be collected at much higher frequency; the Flea3 Point Grey camera can record at 200 frames per second. This suggest the delay between the camera internal clock and the central clock is on the order of milliseconds at most, which we assume is negligible. Using ROS timestamps as central clock, we calibrated
the temporal delay of the Hokuyo UTM-30LX lidar internal clock with respect to the central clock using a built-in procedure. We then paired lidar and camera data according to their ROS timestamps, resulting in an approximately synchronized sensor stream at 30 hz. The high rate of the camera ensures that the error between lidar and camera timestamps is no more than 2.5 milliseconds.

In this research, we make the assumption that the motion of the sensor platform is slow and smooth, making our approximate synchronization scheme sufficient. It is however difficult to comment on the scale of the temporal offset of the internal sensor clocks with respect to the central clock. The built-in ROS procedure to temporally calibrate the lidar may be inaccurate, in which case a significant temporal offset may exist despite our efforts. In Section 5.2.3 we examine how temporal offsets in the sensor clocks affect our cost function, and propose an alternative temporal calibration scheme.

4.3 Optimization

We have so far focused our attention on the derivation of a computationally efficient cost function, but to estimate the optimal calibration parameters we must solve a different problem:

\[
\Xi^* = \arg\min_{\Xi} C(\hat{X}) = \arg\min_{\Xi} \left( -\sum_{i=1}^{M} \sum_{j=i}^{M} N(\hat{x}_i - \hat{x}_j, \Sigma_i + \Sigma_j + 2\sigma^2I) \right). \tag{4.9}
\]

There exist many optimization algorithms suitable for this task, some requiring the derivative and possibly the Hessian matrix with respect to \( T_{C,L} \). In Section 4.3.1 we discuss the difficulty with using gradient-based methods for our choice of cost function. Then, we show in Section 4.3.2 that we instead adopt a gradient-free approach to optimization.

4.3.1 Gradient-based Optimization

To use gradient-based optimization methods, it is necessary to find the analytical or numerical derivative of our cost function with respect to \( T_{C,L} \). Differentiating our cost function with respect to a similarity transformation is a challenging problem, largely due to how nested \( T_{C,L} \) is in the cost function. The matrix is used to obtain lidar points in
Figure 4.5: One dimension of the numerical gradient of our cost function, estimated through forward differences, for multiple step sizes.

the global frame $\hat{x}_i$ and its covariance $\Sigma_i$:

$$C(\vec{x}) = \arg\min_{\vec{x}} \left( -\sum_{i=1}^{M} \sum_{j=i}^{M} \mathcal{N}(\hat{x}_i - \hat{x}_j, \Sigma_i + \Sigma_j + 2\sigma^2 I) \right),$$

(4.10)

$$\hat{x}_i \leftarrow \hat{p}_{G,k}^{(n)} = T_{G,Ck} T_{Ck,Lk} p_{Lk}^{(n)},$$

(4.11)

$$\Sigma_i = J_k^{(n)} Q_k J_k^{(n)T},$$

(4.12)

$$J_k^{(n)} = \frac{\partial \hat{p}_{G,k}^{(n)}}{\partial T_{G,Ck}}.$$  

(4.13)

It is exactly the dependence through the derivative of the SE(3) transform $T_{G,Ck}$ that makes the analytical derivative challenging to compute.

Instead, we can try to use numerical methods to obtain the derivative of the cost function. Unfortunately, we find in practice that the cost function lacks smoothness on a small scale. Figure 4.5 plots the gradient of the cost function in the $x$ direction, as we vary this parameter. Even for reasonably large step size, we cannot use the gradient for optimization.
4.3.2 Gradient-free Optimization

An alternate approach to this problem is the use of gradient-free optimizers. For example, the Nelder-Mead simplex method [62] is a popular algorithm for gradient-free optimization of multivariate functions. For a $d$-dimensional function, the Nelder-Mead method forms a simplex using $d + 1$ points, the vertices of the simplex. The worst vertex, in terms of optimization, is rejected and a heuristic criteria is adopted to determine a new vertex and simplex. The process is repeated until some convergence criteria is met.

In practice, we found this approach is susceptible to local optima; when a poor initial guess was given, with translation error larger than 0.05 meters and angular error larger than 10 degrees, the Nelder-Mead approach often failed to converge to the calibration true values. Instead the algorithm would converge to a local minimum, a point with entropy value larger than at the true value. As it is a requirement of the algorithm to be robust to a large range of initial conditions, Nelder-Mead was to be found inadequate for our application.

Instead, we attempt to use a variation of the Nelder-Mead algorithm, modified for global optimization. Controlled random search with local mutations [63] injects some randomness in the vertex computation of the simplex. This variation of the classic control random search algorithm [64] adopts an improved interpolation scheme for new trial points, while remaining robust to local optima. We found this to be true in practice, though a fine global optimization procedure was deemed too computationally expensive to be practical; the classic Nelder-Mead approach was found to converge much faster, when it did.

As we found the Nelder-Mead (NM) approach adequate with a good initial guess, we adopt the following optimization strategy: first, perform a coarse global optimization through controlled random search with local mutations (CRS) to get close to the global minimum. Then, we perform a fine local optimization through NM, using the output of CRS as input. While still requiring many iterations, this optimization scheme was found to be much faster than plain CRS and just as reliable. In Figure 4.6, 4.7 and 4.8 we demonstrate the optimization of each parameter using this approach, where the switch is made from CRS to NM at about iteration 1750.
Figure 4.6: Global optimization of the translational parameters through CRS and NM.
Figure 4.7: Global optimization of the rotational parameters through CRS and NM.
Figure 4.8: Global optimization of the scale parameters through CRS and NM (top) and progression of entropy minimization (bottom). The shift from CRS to NM occurs at the entropy spike around iteration 1750.
4.4 Implementation

The algorithm was implemented in C++ with a few supporting libraries. The nanoflann k-d tree implementation was used for storage and query of the global point cloud, alongside the Eigen3 library for matrix operations. For optimization, the NLOPT library’s implementations of the CRS and NM algorithms were used.

Algorithm 1 provides the details of the global optimization procedure, summarized as follows: given a set of images and lidar points, we first do an approximate temporal sync at 30 hz, discarding the camera images that do not have an associated lidar scan. Then, we use ORB-SLAM2 to obtain the camera’s incremental motion, up to scale. With an initial guess for the calibration parameters, $\Xi_{\text{init}}$, we can begin the global optimization procedure. We first attempt to minimize our cost function, the approximate entropy of our point cloud as computed by Algorithm 2, using the CRS algorithm with a large tolerance. When this tolerance criteria is met, we use the final estimate as the initial guess of the NM algorithm. With a much smaller tolerance, the estimate is then refined.

Algorithm 1 GlobalOptimization

1: Data In: ${\{\text{Images}\}}, Z$
2: ${\{\text{Images}\}} \leftarrow \text{APPROXSYNC}({\{\text{Images}\}}, Z), 30\text{hz}$
3: $Y = \text{ORBSLAM}({\{\text{Images}\}})$
4: Set: $\Xi_{\text{init}}$
5: function OptimizationRoutine($\Xi_{\text{init}}$)
6: CostFunction = PointCloudRQEApprox
7: Set: $\text{tol}_{\text{CRS}} = 10^{-2}$, $\text{tol}_{\text{NM}} = 10^{-5}$
8: while $(\|\Xi_i - \Xi_{i-1}\| > \text{tol}_{\text{CRS}})$ do
9: $\Xi_i = \text{CRS::MINIMIZE}($CostFunction$, \Xi_{i-1})$
10: end while
11: $\Xi_i = \Xi_{\text{init}}$
12: while $(\|\Xi_i - \Xi_{i-1}\| > \text{tol}_{\text{NM}})$ do
13: $\Xi_i = \text{NM::MINIMIZE}($CostFunction$, \Xi_{i-1})$
14: end while
15: end function

The procedure to efficiently calculate the approximate entropy of a lidar point cloud is outlined in Algorithm 2. Given temporally synced lidar scans and camera poses, as well as a guess of the calibration parameters, we first transform the point cloud to the global frame and compute the covariance of each point. Then, we calculate the maximum eigenvalue of each covariance matrix and sort all points by decreasing eigenvalue. Finally, we calculate the entropy contribution of each relevant pair of points. For each point, we define the neighbourhood radius, then search for points within that distance. To
avoid double counting, we then check the eigenvalue of each neighbouring points, only calculating the entropy contribution of the neighbour if its eigenvalue is smaller than the centroid. After repeating this process, using each point as neighbourhood centroid, we output the negative sum of all entropy contributions.

Algorithm 2 PointCloudRQEApprox

1: Data In: $Y, Z, \hat{T}_{C_k,L_k}$
2: Parameters: $k = 2, \sigma = 0.005$
3: Set: $H = 0$
4: function TransformPointCloud($Y, Z, \hat{T}_{C_k,L_k}$)
5:   for $k = 1 : K$ do
6:     $\hat{T}_{G,C_k} \leftarrow y_k$
7:     for $n = 1 : N$ do
8:         $\hat{X} \leftarrow \hat{p}_{G,k} = T_{G,C_k}T_{C_k,L_k}p_{L_k}^{(n)}$
9:         $J_k^{(n)} = \frac{\partial h^{-1}(x_{L_k}^{(n)}|y_k, \Xi)}{\partial y_k}$
10:        $\Sigma_k^{(n)} = J_k^{(n)}Q_kJ_k^{(n)^T}$
11:        $\lambda_k^{(n)} = \text{MaxEigenvalue}(\Sigma_k^{(n)})$
12:        $\hat{X}, \Sigma, \lambda = \text{SortDescending}(\lambda)$
13:    end for
14: end for
15: return $\hat{X}, \lambda$
16: end function

17: function ApproximateRQE($\hat{X}, \Sigma, \lambda, k, \sigma$)
18:   for $i = 1 : M$ do
19:     $R = 2k(\lambda_m + \sigma^2)$
20:     for all $\{x_j \in \hat{X} | \|x_i - x_j\| \leq R\}$ do
21:        if $\lambda_i > \lambda_j$ then
22:           $H \leftarrow H + N(x_i - x_j, \Sigma_i + \Sigma_m + 2\sigma^2I)$
23:        end if
24:     end for
25: end for
26: return $H$
27: end function
Chapter 5

Simulations

We now seek to validate our calibration approach, first through simulations. As the choice of cost function is largely based on intuition, it is imperative to have a controlled environment to test the approach, where all variables are known, from the sensor trajectory to the noise parameters. To this end, we implemented a MATLAB simulation of a lidar attached to an egomotion sensor, described in Section 5.1. Datasets were generated in the environments described in Section 5.1.1 then tested using the global optimization procedure outlined in Algorithm 1. An analysis of the cost function is presented in Section 5.2.1 and finally the results of the global optimization are presented in Section 5.2.2.

5.1 Setup

We simulated a 2D lidar rigidly attached to a base sensor, with the base sensor following a known trajectory in a simulation environment. Each environment was formed using planar surfaces (triangular or quadrilateral), quadratic surfaces or an affine transformation of those shapes. This limited list of shapes was chosen to facilitate ray tracing. Since the intersection of a ray with any of these shapes has an analytical solution, we do not have to use any occupancy grid-based techniques; we simply calculate the intersection point between each shape and the ray if it exists, then pick the intersection point closest to the lidar. Using this technique, we can quickly generate lidar datasets, with range and bearing output in CSV files, ready to be used in the C++ algorithm. Figure 5.1 demonstrates an example environment along with a few traced lidar rays.

To mimic realistic egomotion measurements, uncorrelated zero-mean Gaussian noise in the translational and rotational parameters of each pose of the base sensor was added. Zero-mean Gaussian noise was also added each lidar range measurement, but no noise on the bearing, as these are accurate in practice. The simulated lidar operated at 40 Hz, with
Figure 5.1: Two views of a simple lidar simulation in a test environment to demonstrate the capabilities of the simulation. The FOV is larger than it appears as no ray (red) is drawn if no object is detected within the lidar maximum range of 10 meters.

a field of view of $240^\circ$ and angular resolution of $0.25^\circ$ degrees per beam. The simulated datasets were 50 seconds in duration and contained up to 1.9 million lidar points. The operational parameters of the simulated sensors and the noise in their measurements were chosen to mimic real hardware, as summarized by Table 5.1.

The base sensor trajectories used in each dataset were generated from sinusoids with unique frequencies and amplitudes for each translational and rotational parameter. A geometrically plausible trajectory was first determined and used as mean, then new trajectories were produced by randomly varying the frequency and amplitude of each sinusoidal component about this mean, as shown in Table 5.1. From work on egomotion-based calibration\cite{48}, we know to avoid a few conditions in the trajectory that could cause degeneracy in the rotation parameters:

1. Constant velocity: the base sensor does not accelerate in any direction.

2. Translation only: the base sensor does not rotate about any axis.

3. Planar motion: the base sensor’s motion is restricted to a plane.

In each of the above cases, the direction of the rotation axis of the base sensor remains constant, causing this degeneracy. Due to the random nature of the trajectories, these degenerate motions would sometimes occur, which would often lead to divergence of the global optimization; we therefore inspected each datasets to make sure we had a
Table 5.1: Simulation parameters: sensor design specification, sensor noise parameters and trajectory parameters. The trajectory is defined by 6 independent sinusoids, each with independent amplitude $A$ and frequency $\omega$.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensor parameters</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lidar FOV</td>
<td>–</td>
<td>240°</td>
</tr>
<tr>
<td>Lidar angular resolution</td>
<td>–</td>
<td>0.25°/beam</td>
</tr>
<tr>
<td>Lidar beam per scan</td>
<td>–</td>
<td>960</td>
</tr>
<tr>
<td>Lidar scan rate</td>
<td>–</td>
<td>40 hz</td>
</tr>
<tr>
<td>Lidar maximum range</td>
<td>–</td>
<td>10 m</td>
</tr>
<tr>
<td>Position noise</td>
<td>$\sigma_{pos}$</td>
<td>0.005 m</td>
</tr>
<tr>
<td>Orientation noise</td>
<td>$\sigma_{ori}$</td>
<td>1°</td>
</tr>
<tr>
<td>Lidar range noise</td>
<td>$\sigma$</td>
<td>0.005 m</td>
</tr>
<tr>
<td><strong>Trajectory (sinusoid) parameters</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dataset duration</td>
<td>–</td>
<td>50 seconds</td>
</tr>
<tr>
<td>Amplitude in $x$ direction</td>
<td>$A_x$</td>
<td>2.2 m</td>
</tr>
<tr>
<td>Amplitude in $y$ direction</td>
<td>$A_y$</td>
<td>1.5 m</td>
</tr>
<tr>
<td>Amplitude in $z$ direction</td>
<td>$A_z$</td>
<td>1.2 m</td>
</tr>
<tr>
<td>Amplitude in $\phi$ (roll) direction</td>
<td>$A_\phi$</td>
<td>$\pi/3$ rad</td>
</tr>
<tr>
<td>Amplitude in $\theta$ (pitch) direction</td>
<td>$A_\theta$</td>
<td>$\pi/4$ rad</td>
</tr>
<tr>
<td>Amplitude in $\psi$ (yaw) direction</td>
<td>$A_\psi$</td>
<td>$\pi/2$ rad</td>
</tr>
<tr>
<td>Frequency in $x$ direction</td>
<td>$\omega_x$</td>
<td>0.30 hz</td>
</tr>
<tr>
<td>Frequency in $y$ direction</td>
<td>$\omega_y$</td>
<td>0.19 hz</td>
</tr>
<tr>
<td>Frequency in $z$ direction</td>
<td>$\omega_z$</td>
<td>0.12 hz</td>
</tr>
<tr>
<td>Frequency in $\phi$ (roll) direction</td>
<td>$\omega_\phi$</td>
<td>0.27 hz</td>
</tr>
<tr>
<td>Frequency in $\theta$ (pitch) direction</td>
<td>$\omega_\theta$</td>
<td>0.20 hz</td>
</tr>
<tr>
<td>Frequency in $\psi$ (yaw) direction</td>
<td>$\omega_\psi$</td>
<td>0.28 hz</td>
</tr>
</tbody>
</table>
Figure 5.2: The noisy, mean trajectory of our base sensor, with orientation shown every 100th pose.

non-degenerate trajectory. The mean trajectory used in the data sets is shown, with associated random noise, in Figure 5.2

5.1.1 Environment

We created five distinct simulation environments, designed to be of increasing difficulty for the algorithm:

1. **Simple Room** consists only of orthogonal planar surfaces, in which the sensors move inside an enclosed rectangular ‘room’ (Figure 5.3a);

2. **Underground Parking Lot** adds several pillars (cylinders) to the *Simple Room* environment (Figure 5.3b);

3. **Plane City** contains planes of varying size, some occluding certain parts of the scene (Figure 5.3c);

4. **Quadratic Forest** is an open environment with spheres mounted on top of cylinders, and where the only planar surface is the ground (Figure 5.3d); and

5. **Triangle Array** is an open environment filled with non-intersecting triangles of various sizes (Figure 5.3e).
The first environment, the *simple room*, was used as initial test environment. From other calibration approaches, both with targets[16] and in the wild[38], there is this notion that three orthogonal plane is a sufficient requirement on the environment to ensure proper calibration, along with a non-degenerate trajectory[48]. We tested the algorithm’s robustness to non-planar surfaces by adding cylinders and sphere, similar to pillars found in an *underground parking lot*. Then, we created a planar but heavily obstructed scenes, with several smaller intersecting planes and occlusions; we designed the *plane city* environment to assess if this approach could handle smaller, localized structures.

After successful calibration in these closed environments, we designed two open environments with more complex structures to test the limits of the algorithm. In the *quadratic forest*, only the ground is a planar surfaces and most lidar hits fall on the amalgam of cylinders and spheres that make up the environment. The last environment is composed of several floating triangles of different size and orientation, none of which intersect. With this arrangement, we wanted to confirm that localized, independent structures were sufficient for the algorithm to converge.
Figure 5.3: Simulation environments and base trajectory (blue) used to validate our method, with a cut away of the lidar measurements (red).
5.2 Results

5.2.1 Cost Function Validation

We have discussed how entropy minimization is an intuitive way to quantify point cloud crispness and therefore extrinsic calibration accuracy, but the measured entropy is dependent on the sensor trajectory and the environment, so it is difficult to derive convergence criteria for this approach. Instead, we provide experimental validation. While we cannot directly visualize this high-dimensional optimization problem, we can gain some insight by varying each parameter individually while holding the others constant. In Figure 5.4, Figure 5.5 and Figure 5.6, we see that even with noise, the cost function is minimized very close to the calibration parameters’ true values, but there occasionally remains a small bias.

Outside of the region near the true calibration values, we can observe that the cost function is not particularly smooth. While this method of visualizing the cost function shows us there is a strong minimum at the true values, we expect the existence of several local minima, increasing the difficulty of the global optimization task. This also provides no guarantee the global minimum is at the true parameter values, but we will see experimentally that this is usually the case.

Figure 5.4: Effect of variation of individual translational parameters on cost function, with others held constant. Offset error versus cost is displayed for each translational parameter.
Figure 5.5: Effect of variation of individual rotational parameters on cost function, with others held constant. Offset error versus cost is displayed for each rotational parameter.

Figure 5.6: Scale parameter versus cost with true value at scale $s = 1$ (in red), with SE(3) transform parameters held constant.
Table 5.2: Average absolute error and standard deviation, $\mu \ (\sigma)$, over ten unique trajectories, for the five simulation environments.

<table>
<thead>
<tr>
<th>Environment</th>
<th>$x \ [\text{mm}]$</th>
<th>$y \ [\text{mm}]$</th>
<th>$z \ [\text{mm}]$</th>
<th>$\phi \ [\text{deg}]$</th>
<th>$\theta \ [\text{deg}]$</th>
<th>$\psi \ [\text{deg}]$</th>
<th>Scale $\times 10^{-3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Room</td>
<td>2.8 (2.5)</td>
<td>3.1 (2.5)</td>
<td>5.2 (3.5)</td>
<td>0.22 (0.12)</td>
<td>0.051 (0.043)</td>
<td>0.24 (0.15)</td>
<td>0.33 (0.30)</td>
</tr>
<tr>
<td>Underground Parking Lot</td>
<td>4.5 (4.4)</td>
<td>4.8 (4.1)</td>
<td>5.2 (4.5)</td>
<td>0.37 (0.22)</td>
<td>0.11 (0.11)</td>
<td>0.37 (0.17)</td>
<td>1.2 (0.9)</td>
</tr>
<tr>
<td>Plane City</td>
<td>4.1 (2.1)</td>
<td>5.2 (4.2)</td>
<td>4.0 (3.9)</td>
<td>0.38 (0.23)</td>
<td>0.18 (0.06)</td>
<td>0.35 (0.23)</td>
<td>0.69 (0.55)</td>
</tr>
<tr>
<td>Quadratic Forest</td>
<td>4.6 (2.6)</td>
<td>3.9 (1.6)</td>
<td>2.9 (3.3)</td>
<td>0.32 (0.20)</td>
<td>0.074 (0.54)</td>
<td>0.35 (0.27)</td>
<td>0.73 (0.32)</td>
</tr>
<tr>
<td>Triangle Array</td>
<td>3.0 (1.9)</td>
<td>2.9 (1.6)</td>
<td>4.6 (3.5)</td>
<td>0.64 (0.60)</td>
<td>0.10 (0.07)</td>
<td>0.61 (0.58)</td>
<td>0.47 (0.26)</td>
</tr>
</tbody>
</table>

5.2.2 Global Optimization

Table 5.2 summarizes calibration accuracy for each parameter in the simulation environments by presenting the average absolute error over 10 randomly generated trajectories. Our method achieves millimetre translational and sub-degree rotational accuracy in all environments, as well as precise scale measurements.

In practice, we found that the base sensor trajectory determined algorithm convergence far more than the environment itself. For example, some individual runs for the more challenging environments, Quadratic Forest and Triangle Array, produced better results than a typical run in the Simple Room environment. However, it was more challenging to find a suitable trajectories in the those two environments. When generating base sensor trajectories, we found avoiding the degenerate cases that were previously discussed did not guarantee convergence. Furthermore, we found that some trajectories were suitable for some environments but not others. Generally, we can say that the algorithm converges reliably if the lidar repeatedly scans the same surfaces from several different viewpoints, but the exact requirements on the trajectory remain unclear.

Despite our computational enhancements, computation time was slow; it took between 2000 and 3000 iterations of CRS and NM to find the global minimum and between 1 or 2 seconds per iteration, depending on the dataset and environments. The implementation would therefore have to be significantly revisited for online use.

5.2.3 Temporal Calibration

In this section, we examine the consequences of an assumption on the synchronicity of the data streams. We later discuss, in Section 6.2 how a temporal delay between sensors severely impacted the robustness of the algorithm. As such, we pursued the possibility of adapting our approach for temporal calibration of the sensor data streams, through simulations. We generated 10 datasets in the simple room environment, identical to the ones used in Table 5.2 except with a 20 ms delay to the lidar scans with respect to
the associated camera poses. This 20ms delay represents a worst case scenario based on a state-of-the-art temporal calibration algorithm for this sensor combination[39]. We first examined the result of our global optimization scheme on this temporally delayed dataset, as showed in the temporally uncalibrated column of Table 5.3. With an added temporal delay, our approach performs significantly worse, and the results are heavily biased.

Rather than performing spatial and temporal calibration jointly, we follow [11] and instead attempt to perform temporal calibration prior to spatial calibration, again through entropy minimization (instead of using, for example, the TICSync library [65]). We locked the Sim(3) transform parameters at nominal values and varied the time delay $t_d$ between the camera and lidar data streams, performing a simple linear interpolation between camera poses. Once the optimal value of $t_d$ was found, we then held that value constant as we carried out a global optimization over the Sim(3) parameters. A sample result using the true spatial calibration parameters is shown in Figure 5.7a, demonstrating that for a noisy trajectory in the Simple Room environment, it is possible to recover the temporal offset exactly.

In realistic scenarios, however, the spatial transform will not be known exactly. In Table 5.3, we show the average time offset error after temporal calibration, when the spatial transform parameters are held away from their true values. The parameters were seeded $\pm 30$ mm and approximately $\pm 5^\circ$ away from ground truth as this represents an initial guess accuracy that should be easily attainable in practice. Figure 5.7b shows how, given a poor initial guess, the estimated time delay between sensors can be inaccurate.

The results presented in Table 5.3 indicate how difficult point cloud reconstruction becomes when a temporal offset is unaccounted for. Pre-calibrating the time offset is shown to substantially improve the accuracy of the Sim(3) parameter estimates, but the errors remain considerably larger than those in Table 5.2, especially for the angular values. Given the result illustrated by Figure 5.7a, we hypothesize that simultaneously estimation of the temporal and spatial offset between sensors through entropy minimization may prove to be a reliable approach, but this requires significant modifications to the original algorithm and is outside the scope of this thesis.
(a) Error in temporal offset between sensors when the Sim(3) transform parameters are held constant at their true values as shown in Table 5.3.

(b) Error in temporal offset between sensors when the Sim(3) transform parameters are held constant at the seed values shown in Table 5.3.

Figure 5.7: Sample result of the temporal pre-calibration approach, based on a trajectory generated in the Simple Room environment.
Table 5.3: Average absolute error and standard deviation, $\mu$ ($\sigma$), on 10 temporally delayed data sets in the *Simple Room* environment, with and without temporal pre-calibration.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Initial</th>
<th>True</th>
<th>Temporal uncalibrated$^1$</th>
<th>Temporally pre-calibrated$^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_d$ [ms]</td>
<td>-20.0</td>
<td>20.0</td>
<td>-</td>
<td>5.6</td>
</tr>
<tr>
<td>$x$ [mm]</td>
<td>-230</td>
<td>-200</td>
<td>10.7 (7.8)</td>
<td>3.81 (1.89)</td>
</tr>
<tr>
<td>$y$ [mm]</td>
<td>80.0</td>
<td>50.0</td>
<td>7.1 (6.5)</td>
<td>3.0 (2.2)</td>
</tr>
<tr>
<td>$z$ [mm]</td>
<td>330</td>
<td>300</td>
<td>18.2 (10.2)</td>
<td>4.9 (3.5)</td>
</tr>
<tr>
<td>$\phi$ [deg]</td>
<td>9.74</td>
<td>14.3</td>
<td>0.67 (0.59)</td>
<td>0.39 (0.42)</td>
</tr>
<tr>
<td>$\theta$ [deg]</td>
<td>91.7</td>
<td>97.4</td>
<td>0.15 (0.13)</td>
<td>0.10 (0.06)</td>
</tr>
<tr>
<td>$\psi$ [deg]</td>
<td>63.0</td>
<td>57.3</td>
<td>0.79 (0.55)</td>
<td>0.47 (0.40)</td>
</tr>
<tr>
<td>Scale</td>
<td>1.2</td>
<td>1.0</td>
<td>$2.0 \times 10^{-3}$</td>
<td>$0.81 \times 10^{-3}$</td>
</tr>
</tbody>
</table>
Chapter 6

Experiments

Following simulations, we performed hardware experiments to further validate the approach. While we attempted to mimic sensors as realistically as possible when generating synthetic data, there are known sources of noise\textsuperscript{57} that were unaccounted for in our simulations. Real environments also tend to be more challenging, as they can be more cluttered and also may contain dynamic objects. Other problematic effects include lighting conditions, which can affect the quality of camera information significantly, while the reflectivity of surfaces can affect the quality of lidar measurements. Experimental validation therefore poses additional challenges which the algorithm may not be robust to. In the following sections, we present our experimental setup and results.

6.1 Experimental Setup

Experiments were conducted with a hand-held sensor rig consisting of a Hokuyo UTM-30LX planar lidar and a Point Grey Flea3 monocular camera. The sensors were rigidly fixed on the two mounts displayed in Figure 6.1(a) and Figure 6.1(b). Data were collected via USB on a laptop configured with ROS\textsuperscript{66}. Laser data were collected at 40 hz, with a 210 degree field of view and 0.25 degree angular resolution, resulting in 960 lidar beams per scan, as in simulations. The camera images were collected at 200 frames-per-second, with a resolution of $640 \times 512$ pixels per frame. As mentioned in Section 4.2.1, we paired the camera and lidar messages according to their timestamps to obtain an approximately synchronized 30 hz data stream. Given the approximately synchronized data, we estimated the trajectory of the camera (up to an unknown scale factor) using ORB-SLAM2\textsuperscript{59}.

Several data sets were collected in the office space at MIT’s Stata Center, shown in Figure 6.2. The sensor rig was moved manually, taking care not to capture images or
Chapter 6. Experiments

(a) Configuration 1: with overlapping sensor FOV.

(b) Configuration 2: non-overlapping sensor FOV.

Figure 6.1: Hardware configurations used in experiments.
laser measurements of the operator’s body. Given the sensor configuration and wide FOV of the lidar, some self-hits definitely occurred. It is also likely that some dynamic objects were captured during data collection, especially in the lidar frame. While the majority of the data set is of a cluttered but static environment, we identified several sources of outliers, making this a realistically challenging test environment.

A data set was collected for each of the sensor configurations shown in Figure 6.1. Individual data sets contain 2-4 minutes of lidar and camera measurements. To keep the optimization tractable, we split the data sets into multiple segments of similar length.

### 6.2 Results

Calibration results are shown in Table 6.1. For each data set, we initialized the calibration parameters simply by visual inspection of the sensor configuration. The scale parameter was initialized by roughly estimating a bounding box around the trajectory during data collection, then comparing with the ORB-SLAM2 trajectory. The scale factor is set based on the choice of keyframes used during ORB-SLAM2 initialization, and hence varies per data set. Since we split data sets in segments, it is possible to compare the scale result of the algorithm between runs from the same set.

Initial testing was successful on only very few data sets due to poor temporal calibration. We then used the built-in ROS software for the synchronization of the lidar clock and ROS clock. This, alongside with slow and smooth trajectories, minimized the impact of the temporal offset between sensors. After making these modifications, the
Table 6.1: Calibration results for the **overlapping** case, for one data set segmented into four trial runs.

<table>
<thead>
<tr>
<th>Calibration results</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$ [mm]</td>
</tr>
<tr>
<td>Initial Guess</td>
</tr>
<tr>
<td>Trial I</td>
</tr>
<tr>
<td>Trial II</td>
</tr>
<tr>
<td>Trial III</td>
</tr>
<tr>
<td>Trial IV</td>
</tr>
</tbody>
</table>

$\mu$ ($\sigma$) 180.3 (5.0) -3.6 (0.8) -49.7 (3.2) -90.03 (0.41) 0.06 (0.15) -90.29 (0.27) 0.508 (0.002)

Table 6.2: Calibration results for the **non-overlapping** case, for one data set segmented into three trial runs.

<table>
<thead>
<tr>
<th>Calibration results</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$ [mm]</td>
</tr>
<tr>
<td>Initial Guess</td>
</tr>
<tr>
<td>Trial I</td>
</tr>
<tr>
<td>Trial II</td>
</tr>
<tr>
<td>Trial III</td>
</tr>
</tbody>
</table>

$\mu$ ($\sigma$) 43.8 (1.2) -0.2 (0.8) 203.0 (0.85) 180.43 (0.24) -1.39 (0.28) -88.90 (0.02) 0.2156 (0.0005)

algorithm performed more reliably, but it is clear that our approximate synchronization of data streams dramatically impacts the accuracy of our approach. This conclusion is what led us to attempt the temporal pre-calibration approach of in Section 5.2.3.

After making modifications to reduce the impact of poor temporal calibration, we found that the Sim(3) transform parameters recovered in our experiments are consistent between runs, with only a few outliers. However, since the data sets in Table 6.1 and 6.2 feature smooth, slow motion, they had to be lengthy were needed to fulfill the requirement that the lidar observe the same surfaces from different viewpoints. As the current optimization procedure remains computationally expensive for larger data set, we were forced reduce the $k$ parameter in Equation (4.7) to $k = 1$, and consequently decrease the accuracy of the cost function. An alternative optimization procedures that would make the computation more tractable would allow us to increase the $k$ parameter, and thus the accuracy of the cost function, which we believe would lead to more accurate results.

In Figure 6.3, we show a comparison of the point cloud in the global frame before and after the calibration procedure for the overlapping FOV case presented in Table 6.1. Figure 6.3a shows the point cloud in global frame before calibration, rotated such that the $+z$ axis corresponds roughly to $up$ in the environment. The initial guess for $T_{C,L}$ which was used to generate Figure 6.3a is not far off, at most a few centimeters, but
error in translational values may be accentuated due to error in the scale parameter. The initial guess for rotational parameters was also very accurate; it was simple to manually estimate rotational parameters for our setup, as the mount made certain the angular offset between sensors could only be in increments of 90 degrees. We found that misalignments due to either internal configurations of the sensor or manufacturing defects in the mount, to cause at most 1 degree of additional offset. Even with a good initial guess, the consequences on the point cloud were significant, potentially accentuated due to a poor scale estimate. Some structure is immediately apparent in Figure 6.3a, but the ground plane is at an angle from the $xy$ plane. The intersecting walls on the right of the plot are also clearly misaligned. After calibration, in Figure 6.3b, the ground is more properly aligned, as well as the walls on the left and right extremes of the plot. The point cloud remains very noisy; there are patches of points distributed at seemingly random locations in the point cloud. These may correspond to occlusions, dynamic object or perhaps self-hits during data collection. By comparing the environment with the point cloud, we also believe some surfaces did not have sufficient reflectivity to be captured by the lidar, as some walls have odd patches lacking lidar data. It is a success of the algorithm that, despite these significant sources of noise which we did not account for, the calibration algorithm still performs consistently.
Figure 6.3: Visualization of a point cloud collected in the environment pictured in Figure 6.2, using initial values and mean values from Table 6.1. Note the smooth wall segments on the right side of Figure 6.3b.
Chapter 7

Conclusion

The automatic extrinsic calibration method presented in this thesis represents a novel approach for a planar lidar to monocular camera setup. While we focus our attention to Sim(3) calibration, the technique could just as effectively be used to calibrate a 2D or even 3D lidar to any motion estimation system. GNSS platforms, visual-inertial systems, stereo cameras, and 3D lidars are just a few examples of systems that could make use of the proposed approach. With some modifications, it may even be possible to simultaneously calibrate the internal parameters of sensors like cameras and inertial measurement units, or even calibrate point clouds obtained through other means than lidars (e.g. depth maps created by multiple cameras or RGB-D cameras). The distinct value of this calibration procedure is that it can be performed in virtually any environment, with no limitations on sensor configuration. The method is also simple to implement and there is little preprocessing required on the data. In this regards, we have fulfilled the objective of this thesis: taking a step towards a generalized calibration method, applicable to many sensor pairs and environment.

We have found in practice that the mis-estimation of the temporal offset between the sensor data streams can severely degrade the quality of the Sim(3) transform parameter estimates. Our temporal pre-calibration approach has limited accuracy, but the results presented in this work motivate the use of RQE minimization in general spatiotemporal calibration. For many applications, data synchronization is a must, so an implementation that simultaneously estimates temporal and SE(3) or Sim(3) calibration parameters is a natural extension to this work.

On the other hand, before further increasing the size of our calibration parameter vector, a more thorough study of the method would be valuable. The algorithm did fail to converge from time to time, for no immediately obvious reason; the trajectory seemed adequate, the lidar point clouds wholly captured the environment, and yet the
optimization procedure reached a heavily biased minimum. A better understanding of the convergence criteria of the algorithm would necessarily lead to more reliable performance in practice. The issue remains that a direct observability analysis is difficult; the cost function depends on the environment, trajectory and sensor models used. An experimental analysis, starting with exceedingly simple environment and trajectories could provide insight into this.

Finally, we recognize an issue of computational tractability for our algorithm, which severely limited the amount of experimentation that was possible. There are a few possible venues for improvement: first, it is possible that we can greatly reduce the size of point clouds used for calibration. Perhaps key sections of the point cloud have all the necessary information, for example at the intersection of multiple surfaces. Alternatively, we could thin out the point cloud by using a smaller lidar field of view, or larger angular resolution. Finally, we recognize the highly parallelizable nature of the cost function. Indeed, with our particular implementation, the entropy contribution of centroid-neighbour pairs could be computed simultaneously. It could be possible to exploit modern GPU processing to massively parallelize cost function computations. As this would improve the computational efficiency by a very large margin, we could increase neighbourhood size and even achieve potentially more accurate calibration.
Bibliography


Appendix A

Matrices

A.1 Rotation Matrix

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

\[
C_k(\phi_k, \theta_k, \psi_k) = \\
\begin{bmatrix}
\cos(\psi_k) \cos(\theta_k) & \cos(\psi_k) \sin(\theta_k) & -\sin(\psi_k) \\
\sin(\psi_k) \cos(\theta_k) & \sin(\psi_k) \sin(\theta_k) & \cos(\psi_k) \\
\sin(\theta_k) & -\cos(\theta_k) & 0
\end{bmatrix}
\]  

(A.1)

A.2 SE(3) Jacobian

The Jacobians for the sensor model in Section 3.1.1 is presented below, broken up in blocks:

\[
J_k^{(n)}(0 : 2, 0 : 2) = \\
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]  

(A.2)
Appendix A. Matrices

\[ J^{(n)}(0 : 2, 3) = \begin{pmatrix}
 y (\sin(\phi_C) \sin(\psi_C) + \cos(\phi_C) \cos(\psi_C) \sin(\theta_C)) + z (\cos(\phi_C) \sin(\psi_C) - \cos(\psi_C) \sin(\phi_C) \sin(\theta_C)) \\
 -y (\cos(\psi_C) \sin(\phi_C) - \cos(\phi_C) \sin(\psi_C) \sin(\theta_C)) - z (\cos(\phi_C) \cos(\psi_C) + \sin(\phi_C) \sin(\psi_C) \sin(\theta_C)) \\
 \cos(\theta_C) (y \cos(\phi_C) - z \sin(\phi_C)) \\
 0
\end{pmatrix} \]  
(A.3)

\[ J^{(n)}(0 : 2, 4) = \begin{pmatrix}
 y (\sin(\phi_C) \sin(\psi_C) + \cos(\phi_C) \cos(\psi_C) \sin(\theta_C)) + z (\cos(\phi_C) \sin(\psi_C) - \cos(\psi_C) \sin(\phi_C) \sin(\theta_C)) \\
 -y (\cos(\psi_C) \sin(\phi_C) - \cos(\phi_C) \sin(\psi_C) \sin(\theta_C)) - z (\cos(\phi_C) \cos(\psi_C) + \sin(\phi_C) \sin(\psi_C) \sin(\theta_C)) \\
 \cos(\theta_C) (y \cos(\phi_C) - z \sin(\phi_C)) \\
 0
\end{pmatrix} \]  
(A.4)

\[ J^{(n)}(0 : 2, 5) = \begin{pmatrix}
 z (\cos(\phi_C) \cos(\psi_C) + \sin(\phi_C) \sin(\psi_C) \sin(\theta_C)) - x \cos(\theta_C) \sin(\psi_C) \\
 z (\sin(\phi_C) \sin(\psi_C) + \cos(\phi_C) \cos(\psi_C) \sin(\theta_C)) - y \cos(\phi_C) \sin(\psi_C) - \cos(\psi_C) \sin(\phi_C) \sin(\theta_C)) \\
 x \cos(\phi_C) \cos(\theta_C)
\end{pmatrix} \]  
(A.5)

A.3 Sim(3) Jacobian

For the sensor model in Section 3.1.2, the first bloc of the Jacobian shown in Equation (A.2) is modified to

\[ J^{(n)}(0 : 2, 0 : 2) = \begin{pmatrix}
 s & 0 & 0 \\
 0 & s & 0 \\
 0 & 0 & s
\end{pmatrix} \]  
(A.6)