Master's Thesis

Targetless Extrinsic Calibration of Camera and LiDAR from Motion and Geometric Structures

Srdjan Milojevic

Examiner: Prof. Dr. Wolfram Burgard
Advisers: Wera Winterhalter, Freya Fleckenstein

Albert-Ludwigs-University Freiburg

Faculty of Engineering

Department of Computer Science

Autonomous Intelligent Systems

September 7, 2020

Writing Period

04. 07. 2020 – 07. 09. 2020

Examiner

Prof. Dr. Wolfram Burgard

Second Examiner

Prof. Dr. Thomas Brox

Advisers

Wera Winterhalter, Freya Fleckenstein

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

Intelligent agents are often required to localize within their environment to perform certain tasks. To enable localization, they perceive their environment with sensors. Oftentimes, various types of sensors are used together to provide complementary information about specific aspects of the world. In order to effectively fuse the data acquired from different sensor modalities, an extrinsic calibration that defines the pose of the sensors relative to each other is required. The problem of estimating the extrinsic calibration parameters between a LiDAR and camera sensor is well-studied in the field of mobile robotics. Some approaches depend on modifying the environment with calibration objects that are easily identified by both sensors, e.g., a calibration checkerboard. Others require manual selection of corresponding points to perform the calibration. In this thesis, we present a novel approach that is based on estimating the trajectory of each sensor and matching detected geometric edges in the scene as feature correspondences. The presented method then constructs a hyper-graph from the ego-motions of camera and LiDAR to perform a graph-based optimization of the calibration parameters. In addition, we define a similarity measure between 2D image lines and 3D LiDAR edges to further constrain the hyper-graph. A qualitative and quantitative analysis of our method, manual calibration, checkerboard-based approach and our previous feature-based work show, that their calibration accuracy is comparable. Overall, the experiments show, that our method can be used for quick on-site calibration of LiDAR and camera because, contrary to state-of-the-art, our approach requires no calibration objects or modifications to the scene.

Zusammenfassung

Intelligente Agenten müssen sich häufig in ihrer Umgebung lokalisieren, um bestimmte Aufgaben auszuführen. Um die Lokalisierung zu ermöglichen, nehmen sie ihre Umgebung mit Sensoren wahr. Oft werden verschiedene Arten von Sensoren zusammen verwendet, um ergänzende Informationen über bestimmte Aspekte der Welt bereitzustellen. Um die von verschiedenen Sensormodalitäten erfassten Daten effektiv zusammenzuführen, ist eine externe Kalibrierung erforderlich, die die Position der Sensoren relativ zueinander definiert. Das Problem der extrinsischen Kalibrierungsparameterschätzung zwischen einem LiDAR und einem Kamerasensor ist auf dem Gebiet der mobilen Robotik gut untersucht. Einige Ansätze hängen von der Modifizierung der Umgebung mit Kalibrierungsobjekten ab, die von beiden Sensoren leicht identifiziert werden können, z.B. einem Kalibrierungsschachbrett. Andere erfordern die manuelle Auswahl entsprechender Punkte, um die Kalibrierung durchzuführen. In dieser Arbeit stellen wir einen neuartigen Ansatz vor, der auf der Schätzung der Flugbahn jedes Sensors und der Anpassung erkannter geometrischer Kanten in der Umgebung als Merkmalskorrespondenz basiert. Die vorgestellte Methode erstellt dann einen Hypergraphen aus den Ego-Bewegungen von Kamera und LiDAR, um eine graphbasierte Optimierung der Kalibrierungsparameter durchzuführen. Darüber hinaus definieren wir ein Ähnlichkeitsmaß zwischen 2D-Bildlinien und 3D-LiDAR-Kanten, um den Hypergraphen weiter einzuschränken. Eine qualitative und quantitative Analyse unserer Methode, der manuellen Kalibrierung, des Schachbrett-basierten Ansatzes und unserer früheren Merkmal-basierten Arbeit zeigt, dass ihre Kalibrierungsgenauigkeit vergleichbar ist. Insgesamt zeigen die Experimente, dass unsere Methode für die schnelle Kalibrierung von LiDAR und Kamera vor Ort verwendet werden kann, da unser Ansatz im Gegensatz zum Stand der Technik keine Kalibrierungsobjekte oder Änderungen an der Szene erfordert.

Contents

1 Introduction **1**

2 Related Work **5**

2.1 Calibration Using Calibration Targets 5

2.2 Calibration Using Features in the Environment 6

2.3 Calibration Using Sensor Motion 7

3 Background **9**

3.1 Checkerboard-Based Calibration 9

3.2 Visual Odometry 12

3.2.1 Local Feature Descriptors 12

3.2.2 The Essential Matrix 14

3.3 LiDAR Odometry and Mapping 16

3.4 2D Edge Detection 18

3.5 3D Edge Detection 22

4 Approach **25**

4.1 Problem Definition 25

4.2 Initial Motion-Based Calibration 27

4.3 Refined Calibration Using Odometries and Feature Matching 30

4.3.1 3D Edges Preprocessing 30

4.3.2 2D-3D Feature Matching 32

4.3.3 Graph Based Optimization 37

5 Experiments 43
5.1 Sensor Setups and Datasets 43
5.2 Validation 49
5.3 Applicability 53

6 Conclusion 59

7 Acknowledgments 61

Bibliography 68

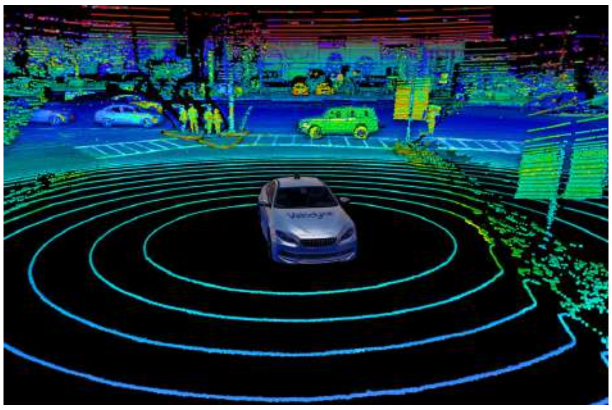
1 Introduction

An intelligent agent perceives its environment using sensors, and based on its percepts, it reacts to the surrounding. Just like humans, who have five senses, intelligent agents are often equipped with various types of sensor modalities that are used together to provide complementary information about specific aspects of the world. But to effectively fuse the data acquired from different sensors, an extrinsic calibration is required. It defines the relative orientation and translation between the sensors.

In recent years, Light Detection and Ranging (LiDAR) sensors (Figure 1(a)) have become popular in the mobile robotics field and in the intelligent vehicle context. They provide accurate range measurements with large angular fields (Figure 1(b)). In addition, they allow for high-frequency real-time data acquisition and their accuracy is independent of light conditions. On the other hand, monocular camera sensors produce high-resolution intensity images with rich information of colour, texture and appearance. Because of the complementary information they provide about the environment, LiDAR and camera sensors are often jointly used on intelligent vehicles (Figure 2).

Compared to using a single sensor modality, the superiority of multi-sensor setups lies in neutralizing weaknesses of each sensor by taking advantage of the strengths of other sensors and in fusing different kinds of perceptual information. In order to integrate LiDAR and camera information effectively, it is required to know the calibration parameters for each sensor. The calibration of these sensors can be

(a) Velodyne Alpha Prime VLS-128.


(b) Point cloud acquired from the top of a car.

Source: (a) https://velodynelidar.com/products/alpha-prime/ [accessed 9 August 2020] (b) https://www.i-micronews.com/how-lidar-is-getting-ready-for-the-automotive-mass-market-an-interview-with-velodyne/ [accessed 11 August 2020]

Figure 1: Velodyne Alpha Prime VLS-128 (a) is a modern LiDAR sensor with 128 beams covering 360° horizontal and 40° vertical field of view. It creates a high resolution (0.2° × 0.1°) dense point cloud (b).

decomposed into calibration of intrinsic and extrinsic parameters. The extrinsic calibration parameters consist of rotation and translation of the reference frame of each sensor to one common frame. In this thesis, we assume that intrinsic parameters, such as the camera calibration matrix [1], are known and focus on the extrinsic calibration.

Automating the calibration of camera and LiDAR sensors has been a research topic ever since LiDAR technology became popular after the DARPA Grand Challenge in 2004 [2]. A variety of approaches have been proposed for the calibration of the extrinsic parameters. Some depend on detecting calibration objects [3, 4, 5], others rely on shapes and features already present in the scene [6, 7]. A planar checkerboard pattern is the most common calibration object used by researches, as it is easy to extract from both camera and LiDAR data. Lately, a new approach that uses an ego-motion estimation of the sensors has been proposed. The method leverages the rigid coupling between the camera and the LiDAR sensor.

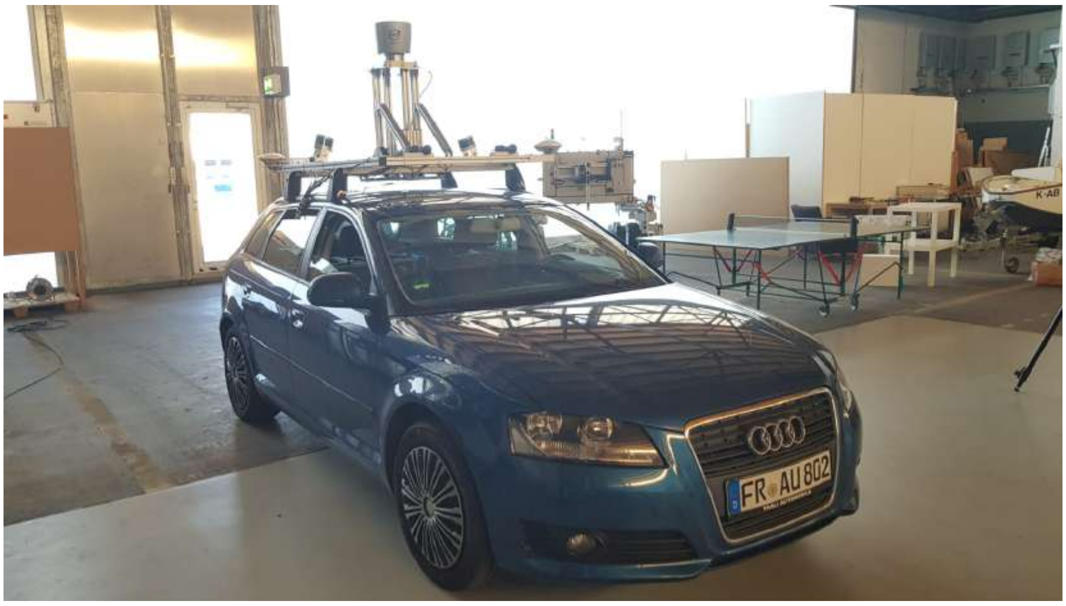


Figure 2: An example of an autonomous car equipped with multiple cameras and LiDARs covering different fields of view. The car relies on its sensors to perceive and react to the environment.

The goal of this thesis is to address the shortcomings of the current state-of-the-art calibration method that uses a checkerboard. Still, it was important that we keep its positive side, high precision. Therefore, we build our method on the motion-based approach and extend it by matching features between the camera and the LiDAR sensor. By combining these two techniques, we preserve the advantages they bring, such as ease of use and independence of calibration objects, as will be further discussed in Section 2. Furthermore, we resolve some of the issues that occur in these two methods, e.g., imprecision and lack of robustness. As a result, we create a novel approach to calibrate a monocular camera with a LiDAR sensor that can, with minor modifications, be extended to calibrate other sensor modalities, as well. Importantly, the results of our method are comparable to the state-of-the-art.

In Chapter 2, we review previous work related to our approach. The overview is divided with respect to the method used to estimate the calibration parameters.

In Chapter 3, we describe a state-of-the-art method for calibrating LiDAR and camera sensors with a checkerboard, that we use to compare the results of our thesis. Furthermore, we introduce the methods we use to estimate the ego-motion of the sensors and to detect features in both camera image and LiDAR point cloud.

Our approach is comprehensively explained in Chapter 4. It can be divided into two parts: initial calibration and refinement. Before the initial calibration, the ego-motion of both camera and LiDAR sensor has to be estimated. By matching the trajectories of rigidly mounted sensors, we obtain the initial calibration parameters. For the refinement step, we first detect 2D lines in images and 3D edges in point cloud data. Finally, we construct an optimization graph from estimated ego-motions and matched features from camera and LiDAR. We remove outliers iteratively and ensure the robustness of the proposed method.

In Chapter 5, we present experimental results obtained using real-world data from both indoor and outdoor scenes recorded by different integrated sensor setups. The results are assessed quantitatively by comparison to parameters obtained by human measurement, checkerboard calibration and our previous feature-based approach. Furthermore, we analyze qualitative results by overlaying a LiDAR point cloud to the image plane. To validate our method, we repeat the experiments on different LiDAR-camera sensor setups and diverse urban environments.

Finally, in Chapter 6 we summarize the thesis, highlight the contributions, discuss possible use cases and suggest potential future work.

2 Related Work

The extrinsic calibration of LiDAR and camera sensors is a well-studied problem in computer vision and robotics. In most cases, the calibration depends on finding correspondences (e.g., point-to-point or point-to-plane) between features seen simultaneously in both sensors. In other cases, the calibration depends on estimating the ego-motion of each sensor. We first cover the traditional methods which use a calibration object that can be easily detected in both sensors in Section 2.1. In Section 2.2, we reference feature-based methods that remove the need to modify the scene with a calibration target. Finally, in Section 2.3 we discuss existing motion-based approaches.

2.1 Calibration Using Calibration Targets

During the long study of the problem of calibrating camera and LiDAR sensor, various types of calibration objects have been proposed. They all share the same requirement - being easily detectable by both sensors. Also, all of these methods require modification of the scene by placing calibration targets such as circle-based objects [3], reflective materials [4] or bright spots [5].

A planar checkerboard pattern is the most common calibration object, as it is easy to extract from both camera and LiDAR data. It was first introduced by Zhang et al. [8] and various approaches have been proposed since. Some use multiple checkerboards placed in the scene [9], requiring just a single image-point cloud pair to calibrate. Others use a recording of a single moving [10] checkerboard to get more feature correspondences. Some even present methods that use a single static checkerboard [11]. However, these determine the checkerboard pattern in the LiDAR data by examining different intensities of the returning laser rays reflected from white and black parts of the checkerboard.

The advantage of these methods is their ease of use and high reliability. Because of the precise parameters they produce, we use a checkerboard-based calibration as described in Section 3.1 to compare to the results of our proposed method. The biggest weakness of these approaches is the dependency on calibration objects and the need to modify the scene with them. Another downside is that the features have to be observed at the same time, requiring a large percent of overlap in the field of view (FoV) of the sensors and thus limiting the use case of these methods. The following section presents methods that build upon the target-based approaches by utilizing objects and structures already present in the scene to perform the calibration.

2.2 Calibration Using Features in the Environment

While there are approaches that rely on manual selection of corresponding points from a single laser-camera acquisition of a natural scene [12], we are interested in the methods where feature correspondences are automatically extracted and matched. Two challenges arise: defining features that are detectable by both camera and LiDAR, and matching the correct features as correspondences. The most commonly used features are edges and corners. Targetless approaches often use a setup providing a high-density point cloud such as a Kinect range sensor [6] or a 2D laser range finder mounted on a nodding mechanism [7].

In previous work, we developed a targetless calibration method that operates on a sparse point cloud from a sixteen-beam LiDAR. Since in a sparse LiDAR point cloud only basic geometric shapes (e.g., cylinders, spheres, planes etc.) can be distinguished, we proposed geometric corners as features, since they are generally common in an urban environment. This characteristic point can be fully determined by a LiDAR as the intersection point of three non-coplanar planes. Also, a geometric corner can be recognised in the camera image as the intersection point of edges formed by three planes.

The advantage of methods based on environment features compared to the target-based ones lies in their ability to operate without modifying the scene with artificial markers or calibration objects. Still, the automatic feature extraction methods are generally not robust and require manual supervision in feature matching. The downside of target-based approaches of requiring overlapping FoV remains. Nonetheless, overlapping FoV are not a necessity for motion-based methods.

2.3 Calibration Using Sensor Motion

Recent studies show that it is possible to estimate calibration parameters using ego-motion of a camera and a LiDAR sensor [13, 14]. This is achieved by leveraging the rigid coupling between the sensors. Although they achieve rather satisfying results, the estimated monocular camera odometry is scaleless which leads to low accuracy of extrinsic translation parameters.

Motion-based approaches tackle the shortcomings of previously mentioned methods in the sense that they require no calibration objects, nor do the FoV need to overlap. However, one big shortcoming of this approach is that the extrinsic calibration parameters are estimated only up to a scale. Furthermore, as visual and LiDAR odometry is prone to drift and rotation errors, the precision of results of this method relies on the odometry estimation being as accurate as possible. Lastly, certain types of motion, such as translation only, can lead to degenerate results [15].

3 Background

In this chapter, we describe an approach that uses a checkerboard to determine the calibration parameters. This method will later be used in Section 5 to evaluate the quality of the results of this thesis.

Next, we analyze the building blocks of our approach. It is fundamental to estimate the ego-motion of sensors as a sequence of rigid body transformations in three-dimensional space. We introduce a mathematical background for estimating camera ego-motion and reference the method we utilize to obtain LiDAR odometry. Also, we give a short overview of the algorithms we use to detect 2D line segments in the camera image and 3D edges in the LiDAR point cloud.

In this thesis, we represent a point and a line in 3D with a bold uppercase letter, while a point and a line on the 2D image plane is marked with a lowercase bold letter. A line segment is described as a tuple of its start and end point.

3.1 Checkerboard-Based Calibration

Being the most common technique, we chose a checkerboard based method [10] for comparing the performance of our approach and will refer to it as the state-of-the-art method for the rest of this thesis. This method relies on a single checkerboard pattern plane detected in several poses, as shown in Figure 3. The constraints are based on data captured simultaneously from the camera and LiDAR sensor.

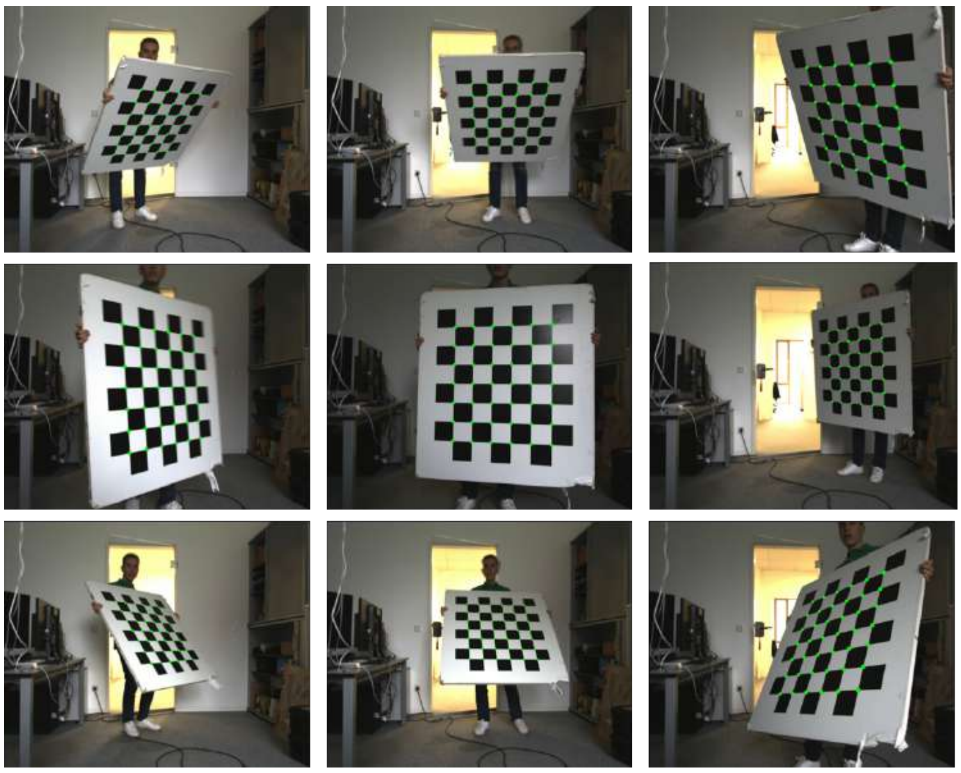


Figure 3: In theory a minimum of three noncoplanar orientations of the checkerboard are required to fully constrain the optimization problem for the checkerboard-based calibration. In practice, recording more views reduces the error of estimated calibration parameters and makes the approach robust to measurement errors.

A planar checkerboard pattern has become the classical calibration object because its pattern is easy to detect in a camera image and its planar structure can be simply extracted from a 3D point cloud. The state-of-the-art approach uses RANdom Sample Consensus (RANSAC) [16] to estimate the equation of the checkerboard plane in a point cloud. However, in order to detect the plane with RANSAC, the user has to tune the parameters such that a bounding box containing only a subsection of the point cloud that includes the calibration object is generated (Figure 4). The checkerboard plane is then detected as the best plane fit for all the points remaining in the filtered section.

The state-of-the-art approach is relatively easy to use and implement. Extracting

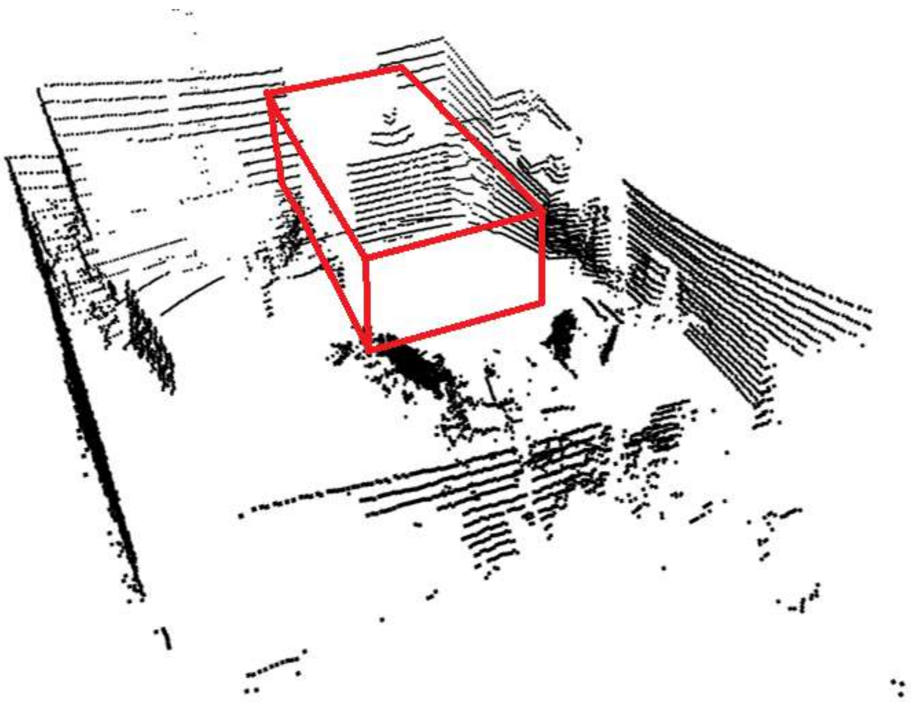


Figure 4: A point cloud obtained by a 16-beam LiDAR sensor. Marked in red is the section of the point cloud which needs to be filtered in order to correctly estimate the equation of the checkerboard plane.

checkerboard corners from an image is executed with a single function from the OpenCV library [17]. Afterwards, using precise measurements of the checkerboard size and the intrinsic calibration and distortion matrix of the camera, detected corners can be projected from the image plane to the reference frame of the camera by solving the Perspective-n-Point problem [18]. As a result, corner points in the image projected into 3D space form a plane. The LiDAR points detected on the checkerboard are then matched onto that plane using a point-to-plane correspondence. The cost function for this optimization problem is the distance of each LiDAR point on the checkerboard plane extracted from the LiDAR data to the plane defined by checkerboard corners from the camera image.

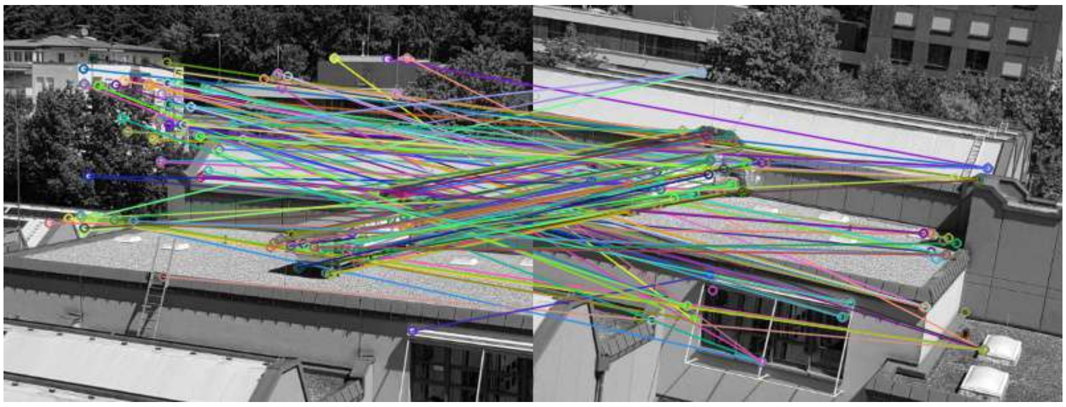
3.2 Visual Odometry

Given a set of captured images $\mathbb{I} = \{I_0, I_1, ..., I_N\}$, visual odometry estimates the camera ego-motion. We represent the camera odometry by a set of transformations $\mathbb{T}_{\text{cam}} = \{\mathbf{T}^1_{\text{cam}}, \mathbf{T}^2_{\text{cam}}, ..., \mathbf{T}^N_{\text{cam}}\}$, where $\mathbf{T}^i_{\text{cam}} = (\mathbf{R}^i_{\text{cam}}, \mathbf{t}^i_{\text{cam}})$ is transformation from camera origin when I_{i-1} was captured to the origin of I_i. Visual odometry determines these transformations by tracking pixel coordinates of some characteristic image points from one frame to the next, and evaluating how the camera origin could have moved to cause the change in the position of these pixels. Therefore, we break down the problem of visual odometry into finding local feature points that can be tracked and estimating the camera motion based on those.

3.2.1 Local Feature Descriptors

In visual odometry, we locate points that remain stable after camera motion. For example, an intuitive feature is a corner point in an image that can be pinpointed in the next camera frame. Many algorithms that rely on extracting corners by examining small local changes in pixel values were proposed, such as Harris corner [19], FAST corner [20], etc. However, these algorithms are not reliable for the needs of visual odometry. When a camera is rotated, the appearance of the corner changes, or when a camera approaches the feature, it may no longer resemble a corner. For that reason, in recent years more stable local image features have been designed, such as SIFT [21], SURF [22], ORB [23] and AKAZE [24]. In contrast to plain corners, these approaches are superior because they are invariant to scale, rotation, illumination or even some levels of distortion. They are differentiated by their computational efficiency, accuracy, repeatability and invariance. A comparative analysis of SIFT, SURF, AKAZE and ORB [25] demonstrates these differences.

Because of its great efficiency, the ORB feature has long been the go-to approach with regards to online visual odometry (e.g., ORB-SLAM [26]). Since we are not interested


(a) Matched keypoints based on Hamming distance.

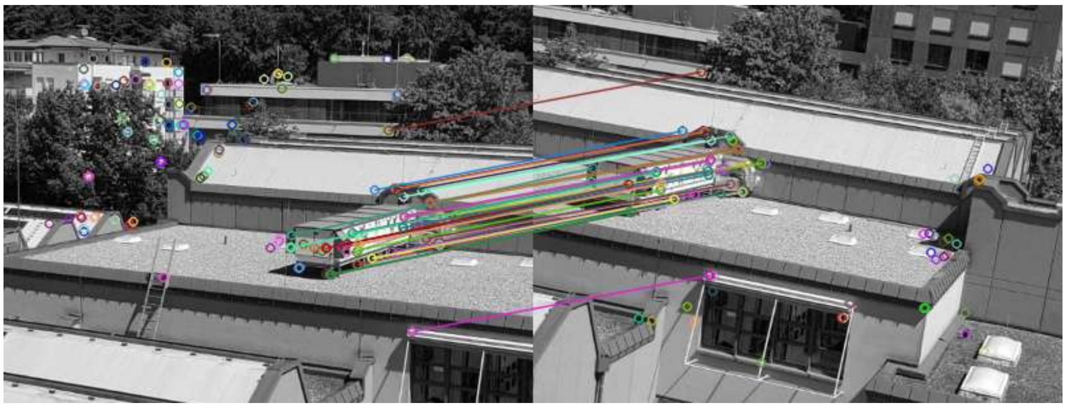

(b) Refined matches.

Figure 5: The results of matching AKAZE keypoints based on their descriptor distance. In (a) we match keypoints that have the smallest Hamming distance. To remove outlier matches, we only keep the pairs with Hamming distance less than twice the minimum distance (b).

in runtime complexity but put a high emphasis on accuracy, we use AKAZE features because of their trade-off between efficiency and precision.

The AKAZE features are composed of detectors and descriptors of keypoints. Descriptors represent local neighbouring pixels around the detected keypoint in a form of binary vector. The Hamming distance is often used as a measure of similarity between two descriptors. In Figure 5(a) we illustrate the results of the OpenCV [17] AKAZE detection and matching of keypoints based on Hamming distance. We see a large number of false matches, so we refine the results keeping only the pairs with a Hamming distance of less than twice the minimum distance. With this empirical fine-tuning, we filter the outliers in a substantial percentage, as shown in Figure 5(b).

3.2.2 The Essential Matrix

Given a set of matched feature points from two consecutive frames I_{i-1} and I_i, the goal pf visual odometry is to recover the camera motion $\mathbf{T}^i_{\text{cam}} = (\mathbf{R}^i_{\text{cam}}, \mathbf{t}^i_{\text{cam}})$ between the images. In this section we omit the superscript i for generality and to simplify notation.

A key component of visual odometry is the Essential matrix $\mathbf{E}$ [1]. It contains the information of the location of the second camera origin relative to the first camera origin, described with rotation $\mathbf{R}_{\text{cam}}$ and translation $\mathbf{t}_{\text{cam}}$. Since $\mathbf{E}$ has five degrees of freedom, it should be possible to determine it with only five points. However, the famous five-point algorithm [27] requires solving multiple non-linear equations to get the essential matrix $\mathbf{E}$. Therefore, we use another approach that requires a minimum of eight points [28]. Even though we removed most of the mismatched feature pairs, there might still be some outliers. To obtain more robust results, we use more than eight points with a RANSAC [16] based outlier removal.

In order to recover the camera motion from the essential matrix $\mathbf{E}$, we analyze its singular value decomposition (SVD). It is proven in [29] that the singular value of $\mathbf{E}$ is in the form of $[\sigma, \sigma, 0]^{\mathrm{T}}$, so we set the SVD of $\mathbf{E}$ to:

$$\mathbf{E} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}, \tag{1}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma, \sigma, 0)$ is the singular value matrix. For every $\mathbf{E}$, there are two possible $\mathbf{R}_{\text{cam}}, \mathbf{t}_{\text{cam}}$ that we recover:

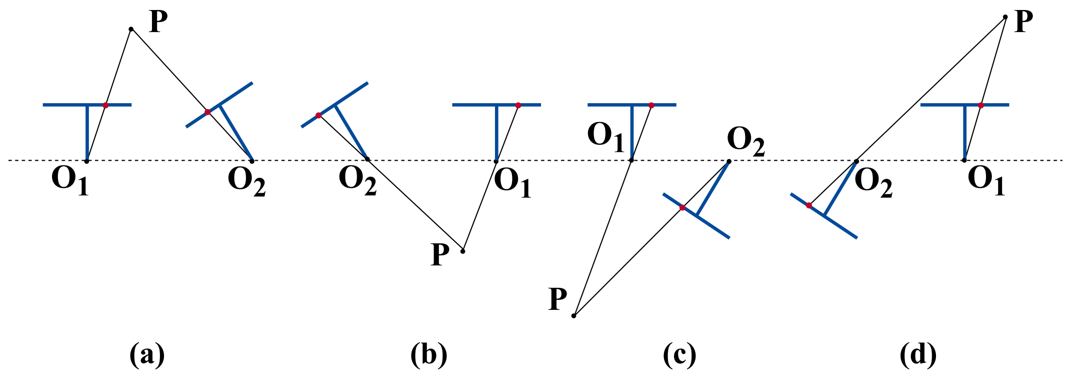


Figure 6: Four possible $\mathbf{R}_{\text{cam}}$ and $\mathbf{t}_{\text{cam}}$ recovered from $\mathbf{E}$. The blue "T" shape represents the focal length and image plane of a camera, while red dot defines the projection of point $\mathbf{P}$ to the image plane.

$$\begin{aligned} \mathbf{R}_{\text{cam}}^{1} &= \mathbf{U}\mathbf{R}_{Z}^{\text{T}}\left(\tfrac{\pi}{2}\right)\mathbf{V}^{\text{T}}, \quad \mathbf{t}_{\text{cam}}^{1\wedge} = \mathbf{U}\mathbf{R}_{Z}\left(\tfrac{\pi}{2}\right)\mathbf{\Sigma}\mathbf{U}^{\text{T}} \\ \mathbf{R}_{\text{cam}}^{2} &= \mathbf{U}\mathbf{R}_{Z}^{\text{T}}\left(-\tfrac{\pi}{2}\right)\mathbf{V}^{\text{T}}, \quad \mathbf{t}_{\text{cam}}^{2\wedge} = \mathbf{U}\mathbf{R}_{Z}\left(-\tfrac{\pi}{2}\right)\mathbf{\Sigma}\mathbf{U}^{\text{T}}, \end{aligned} \tag{2}$$

where $\mathbf{R}_{Z}\left(-\frac{\pi}{2}\right)$ is a matrix representing rotation of 90° along the z-axis. Also, since the polar constraint is equal to zero, using the negative of either translation possibilities will have the same results. Therefore, we get a total of four possible solutions for $\mathbf{R}_{\text{cam}}$ and $\mathbf{t}_{\text{cam}}$ from the essential matrix, as visualized in Figure 6.

Figure 6(a) and (b) correspond to one of the solutions for $\mathbf{R}_{\text{cam}}$ and $\mathbf{t}_{\text{cam}}$, with the difference in translation being inverted, and (c) and (d) represent the other solution, again with both positive and negative translation. It is obvious that only in one of these four scenarios, the point $\mathbf{P}$ is in front of both image planes and that is the only plausible scenario. It can be uniquely determined by triangulating the 3D point and comparing the depths from both camera origins. However, in order to minimize computational requirements, we simply take the smaller of the two rotations and its corresponding translation with a positive sign. This relies on the assumption that only small incremental movements of the sensor setup will be performed. As in monocular visual odometry the trajectory is scaled by an arbitrary factor, we mark the recovered translation with an overline $\overline{\mathbf{t}}_{\mathrm{cam}}$ to indicate its unknown scale. We show in Section 4.2 that our initial calibration solver can determine the correct sign and scale of the translation, so we delay deciding on the sign for now.

Repeating the described process for every set of two consecutive frames I_{i-1} and I_i, we obtain a set of transformations $\mathbb{T}_{\mathrm{cam}}$ that defines the camera odometry. Next, we determine the ego-motion of the LiDAR sensor, as presented in the following chapter.

3.3 LiDAR Odometry and Mapping

Just like we obtain the visual odometry from camera images, we compute the ego-motion of the LiDAR from a set of sparse LiDAR point clouds $\mathbb{Q} = \{\mathbf{Q}_0, \mathbf{Q}_1, ..., \mathbf{Q}_N\}$ acquired at each discrete timestamp. The difference is that we also reconstruct a dense 3D map $\mathbf{Q}^*$ for the traversed environment. We represent the LiDAR odometry as a set of transformations from one timestamp to the next with a set $\mathbb{T}_{\mathrm{lid}} = \{\mathbf{T}^1_{\mathrm{lid}}, \mathbf{T}^2_{\mathrm{lid}}, ..., \mathbf{T}^N_{\mathrm{lid}}\}$, where $\mathbf{T}^i_{\mathrm{lid}} = (\mathbf{R}^i_{\mathrm{lid}}, \mathbf{t}^i_{\mathrm{lid}})$ is the transformation from LiDAR origin when $\mathbf{Q}_{i-1}$ was scanned to the origin of $\mathbf{Q}_i$. The map $\mathbf{Q}^*$ will later be essential to detecting 3D features, as described in Section 3.5.

Traditional iterative closest points (ICP) methods [30] are reliable LiDAR odometers when the scan rate is high compared to the extrinsic motion. Since we have synchronized data acquisition of both sensors to the low camera frame rate of 5Hz, we utilize a more recent and precise approach. Being the current state-of-the-art method on KITTI benchmark [31], ‘‘LiDAR Odometry and Mapping in Real-time’’ [32] (LOAM) achieves a low drift error without the need of independent position estimation sensors, such as GPS or IMU.

The LOAM algorithm consists of odometry step and mapping step. In the former, it estimates the motion of LiDAR at high frequency but low fidelity, while the latter refines the odometry results and generates a 3D reconstruction of the traversed environment. In the following, we briefly explain the components of the LOAM algorithm.

First, LOAM registers every acquired point cloud by extracting the points of interest (sharp edges and planar surfaces). Since LOAM uses a setup with a 2D laser range finder mounted on a nodding mechanism, we modified it to suit our multi-beam LiDAR. For each point from the point cloud, the LiDAR driver infers which beam it belongs to. With that information, we determine the points of interest by evaluating the sharpness term of the local surface in each beam:

$$c_i = \frac{1}{|\mathcal{S}| \cdot \|\mathbf{X}_i\|} \sum_{j \in \mathcal{S}, j \neq i} \| (\mathbf{X}_i - \mathbf{X}_j) \|, \tag{3}$$

where $\mathcal{S}$ is a set of points from one laser beam and $\mathbf{X}_i$ and $\mathbf{X}_j$ are points from that scan. The points with the highest sharpness value are considered to belong to edges, while the points with the lowest value are considered to belong to planes.

Afterwards, the LOAM algorithm determines sharp and smooth point correspondences between two consecutive scans. For each edge point in the current scan line, LOAM finds the edge from the previous scan and computes point to line distance. On the other hand, for each planar point in the current LiDAR scan, the algorithm determines the closest planar patch from the previous scan and calculates the point to plane distance. These distances are then added to the optimization function. LiDAR motion is finally recovered by minimizing the sum of distances between the matched points.

The mapping step is performed in parallel with the odometry computation, as visualized in Figure 7. Let $\mathbb{T}^n_{\text{lid}}$ be the estimated LiDAR odometry and $\mathbf{Q}^*_n$ be the reconstructed map up to the timestamp n. By keeping all the sharp and flat points determined so far, a newly registered scan $\mathbf{Q}_{n+1}$ is matched with the map in a similar

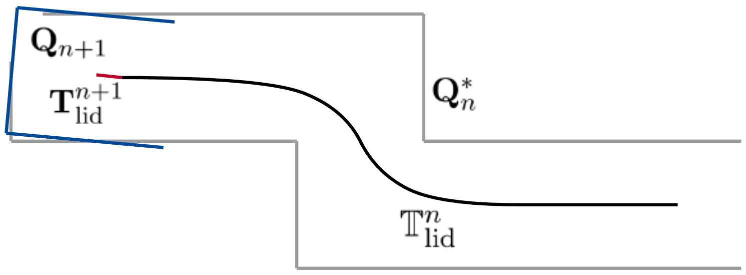


Figure 7: Illustration of LiDAR mapping process. The black curve represents the recovered LiDAR odometry $\mathbb{T}_{\text{lid}}^n$ up to the timestamp n. The red line indicates the transformation from time n to $n+1$ computed in the odometry step. The gray colored line segments illustrate the map $\mathbf{Q}_n^*$ reconstructed up to that point in time, while the blue lines denote the current LiDAR scan. By aligning the blue lines with the gray segments, mapping step refines the estimated transformation $\mathbf{T}_{\text{lid}}^{n+1}$.

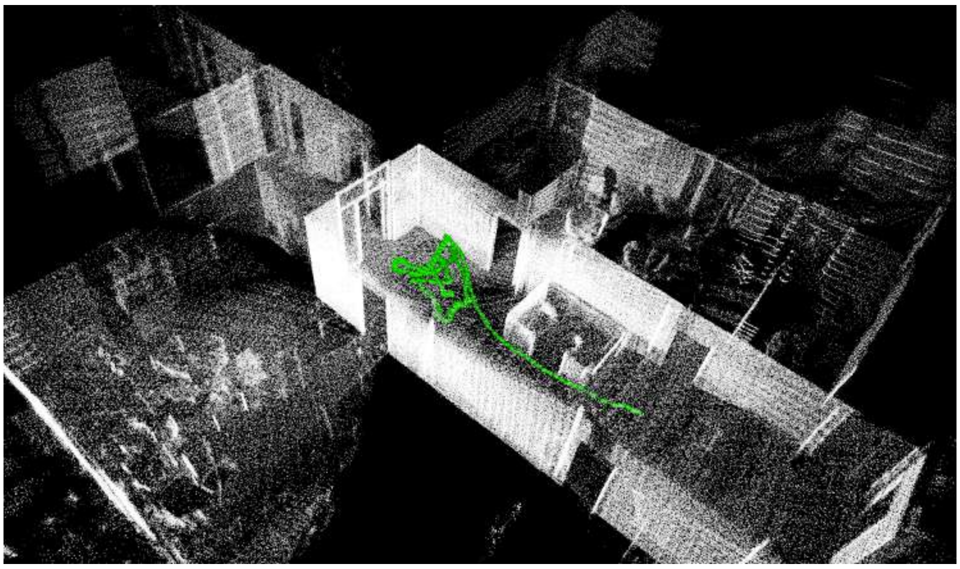
manner as in the odometry step. The difference is that this time LOAM compares sharp and planar points from the map with the current scan. Again, by minimizing the sum of distances between the points of interest, the current odometry estimation $\mathbf{T}_{\text{lid}}^{n+1}$ is refined.

In order to get a map with evenly distributed points, we downsample the point cloud by a voxel grid filter from Point Cloud Library [33] with a cubic voxel of 1 cm^3 in size. This will also be helpful for the 3D edge extraction algorithm. In Figure 8, we demonstrate the results of the extracted odometries and maps in both indoor and outdoor environments.

Having estimated both the camera and LiDAR odometry, in the next chapter we demonstrate how to obtain 2D and 3D features in the data of both sensors.

3.4 2D Edge Detection

Another important building block for our thesis is extracting 2D line segments in each captured camera image I_i, for $i = 0, ..., N$. In this chapter, we explain how we


(a) 3D reconstruction of a hallway in the Autonomous Intelligent Systems laboratory.

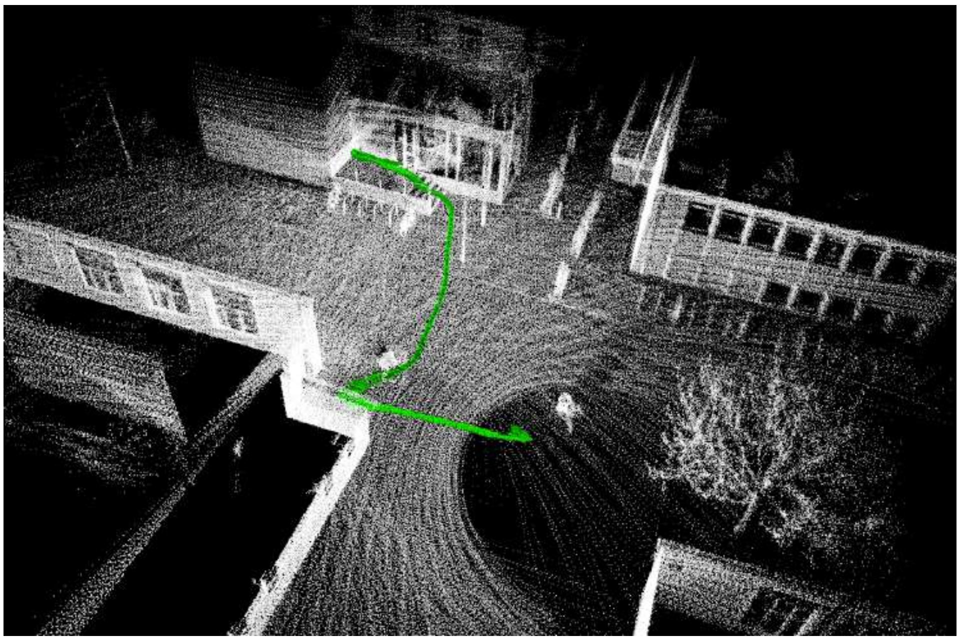

(b) 3D reconstruction of Faculty of Engineering campus, University of Freiburg.

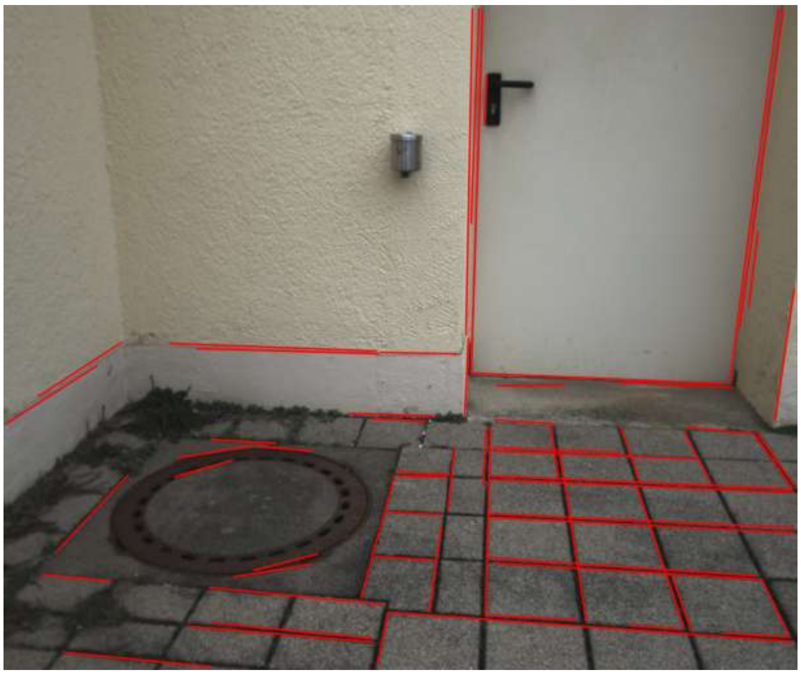
Figure 8: LiDAR odometry and mapping results in indoor (a) and outdoor (b) scene. Green arrows represent the LiDAR odometry.

find the set of features in the camera data $I_i \rightarrow \{\mathbf{l}_{\text{cam}}^{1,i}, \mathbf{l}_{\text{cam}}^{2,i}, ..., \mathbf{l}_{\text{cam}}^{M,i}\}$, where a line is represented by a tuple of its start and end point $\mathbf{l}_{\text{cam}} = (\mathbf{p}_{\text{cam}}^{\text{s}}, \mathbf{p}_{\text{cam}}^{\text{e}})$.

Detecting a line segment in an image is an important yet challenging task in computer vision. Solving this problem facilitates other complex tasks such as 3D reconstruction, image stitching, etc. Therefore, it is a well-studied research area with a collection of proposed methods. The classical way to obtain lines in an image in a parametric form is by the Hough transform [34]. However, it struggles in urban scenes because highly textured surfaces result in fragmented 2D line segments. Also, detecting edges on colour-homogeneous surfaces such as white walls is challenging.

The Fast Line Detector (FLD) is a recent approach introduced by Lee et al. [35]. With image preprocessing prior to FLD, some of the issues with the Hough transform can be reduced but the results are still unreliable, as shown in Figure 9(a). We see a great deal of false positive, but also false negative errors which would produce a drastic amount of outliers in 2D-3D matching. Also, some thick edges are detected more than once, as seen on the floor tiles.

A new trend in computer vision is to use deep learning to overcome the drawbacks of traditional solutions for line detection. This has dramatically improved the line segment detection performance. For our approach, we employ such a learning-based 2D edge extractor that predicts the attraction field map of the image [36]. An attraction field map is a set of vectors that assign each pixel from a line segment region to the line. This way, the learning-based approach turns line detection into a region colouring problem. The network is used to predict an attraction field map from an image followed by a squeeze module [37] that groups pixels belonging to the same line segment. The results are shown in Figure 9(b).


(a) 2D edges detected by FLD with prior image preprocessing.

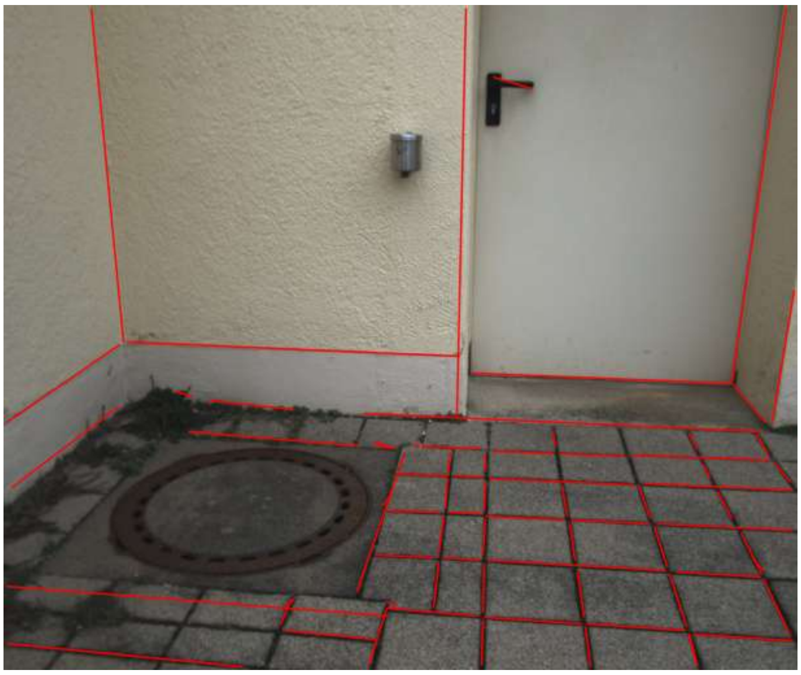

(b) Learning-based 2D edge detector results.

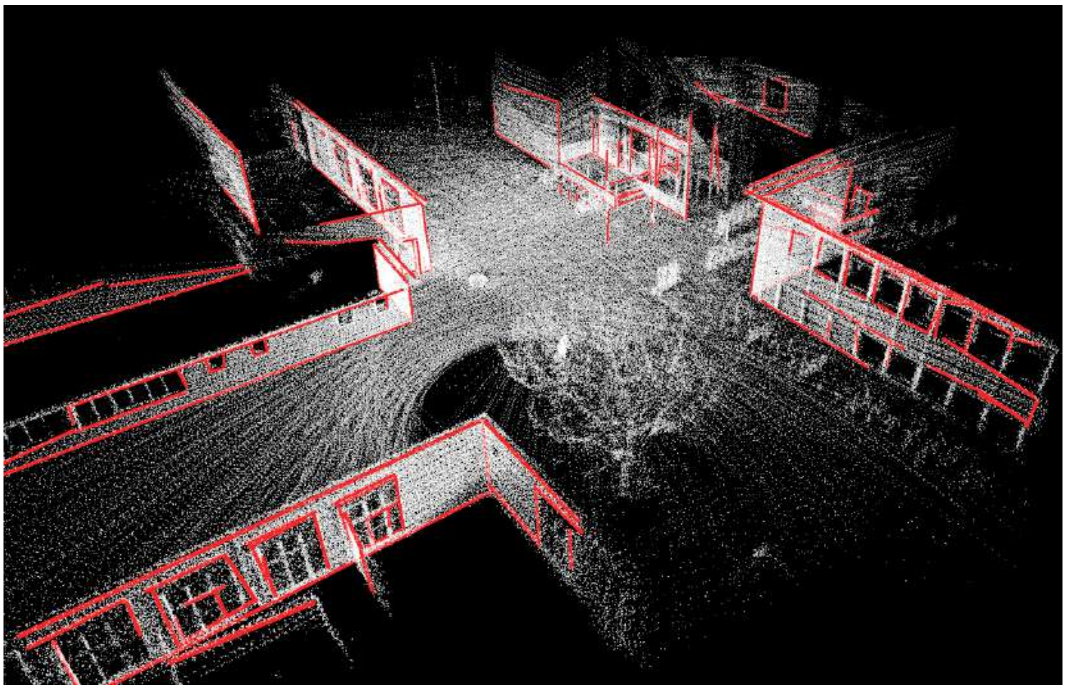
Figure 9: Comparison of 2D edge detectors. On (a) we see a lot of false positive (around the manhole) and false negative errors (corner on the beige wall). The results on (b) are more accurate.

3.5 3D Edge Detection

A line segment is a common structure of a natural scene, especially in urban environments with a lot of planar intersecting shapes. In Section 3.3, we showed how the LOAM algorithm reconstructs a dense 3D map $\mathbf{Q}^*$ of the traversed environment. In this section, we explain how we extract a set of 3D edges from the 3D map $\mathbf{Q}^* \rightarrow \{\mathbf{L}^1_{\text{lid}}, \mathbf{L}^2_{\text{lid}}, ..., \mathbf{L}^K_{\text{lid}}\}$, where a 3D line segment is represented as a tuple of its start and end point $\mathbf{L}_{\text{lid}} = (\mathbf{P}^{\text{s}}_{\text{lid}}, \mathbf{P}^{\text{e}}_{\text{lid}})$.

Contrary to detecting a line in a 2D image, which is a well-studied research topic with multiple efficient solutions, research on extracting a 3D edge from a point cloud is scarce. Some approaches determine points on sharp structures and then fit a 3D line segment [38]. The downside to these methods is in detecting these sharp points, which is not robust to sensor noise. Other approaches first identify all the planes present in the scene and estimate the edges as plane intersection points [39]. While this idea works in theory, in practice it struggles with highly structured scenes with frequent small planes.

Recently, a novel approach that overcomes the mentioned weaknesses has been published by Lu et al. [40]. It first segments the point cloud into 3D planes by region growing and region merging. Afterwards, it projects all points belonging to a 3D plane to their corresponding plane. By considering that 3D plane as a 2D image in 3D space, Lu et al. create a binary image with white pixels marking the projection points of the point cloud. The method then extracts contours in the image and returns 2D line segments. Lastly, the 2D lines are re-projected to 3D space. We run this 3D edge extraction algorithm on the map reconstructed during the LiDAR odometry estimation. In Figure 10, we show the results in both indoor and outdoor scenes. We see that the higher the density of the reconstructed point cloud, the finer the results. Therefore, while testing our proposed method, we perform slow movements to allow the LiDAR to record as many points of an observed area as possible.


(a) Detected 3D edges on a map of Faculty of Engineering campus, University of Freiburg.

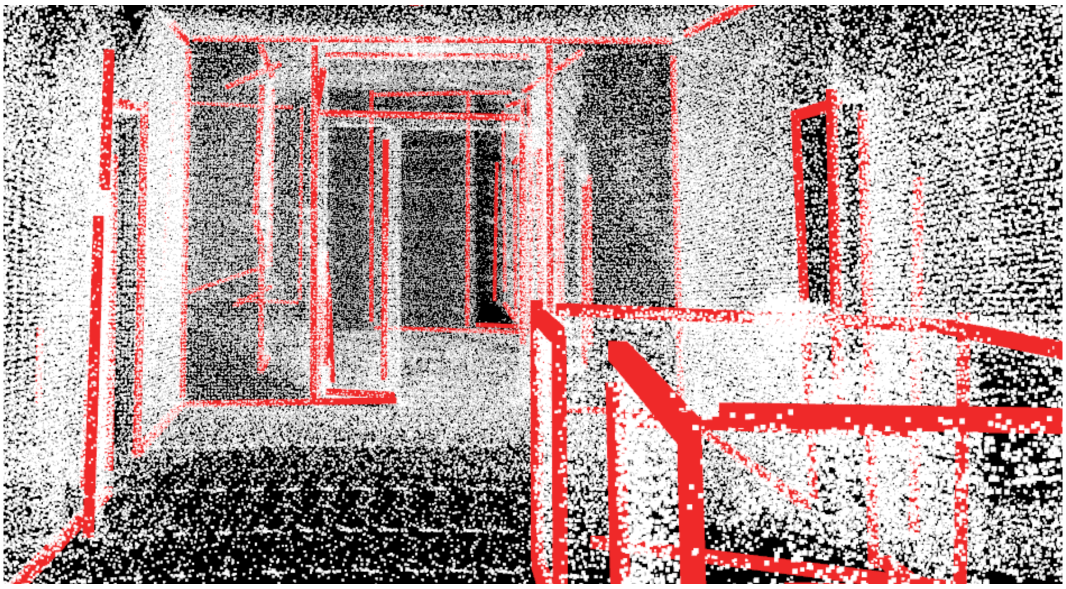

(b) A detail of the hallway in Autonomous Intelligent Systems laboratory.

Figure 10: The results of the 3D edge detection algorithm. The quality of results is dependent on the density of the point cloud. Edge detection on (a) is less accurate on the borders of the point cloud. A dense point cloud (b) provides credible results.

4 Approach

Just like people who have multiple senses, intelligent systems are often equipped with different sensor modalities that provide complementary information about the environment. In order to have a meaningful overlay of the perceived data by each of the sensors, the extrinsic calibration is required. In this work, we address the question of how to automate the process of calibrating a monocular camera and a LiDAR sensor.

4.1 Problem Definition

As previously discussed, motion-based extrinsic calibration between the camera and the LiDAR sensor is easy to use - the operator navigates the sensor setup through an unmodified natural scene and the calibration parameters are obtained. Unfortunately, this approach is sensitive to the ego-motion estimation and in case of a monocular camera, accurate only up to a scale. In this thesis, we address these weaknesses by refining the results of a motion-based method by extending it with a feature matching algorithm.

Our idea is to set up a modular approach for calibrating various sensor modalities. One constraint of the current state-of-the-art method is having an overlapping FoV between the sensors. Also, it depends on the calibration checkerboard. Our approach involves sensor modalities for which ego-motion can be estimated. Another key

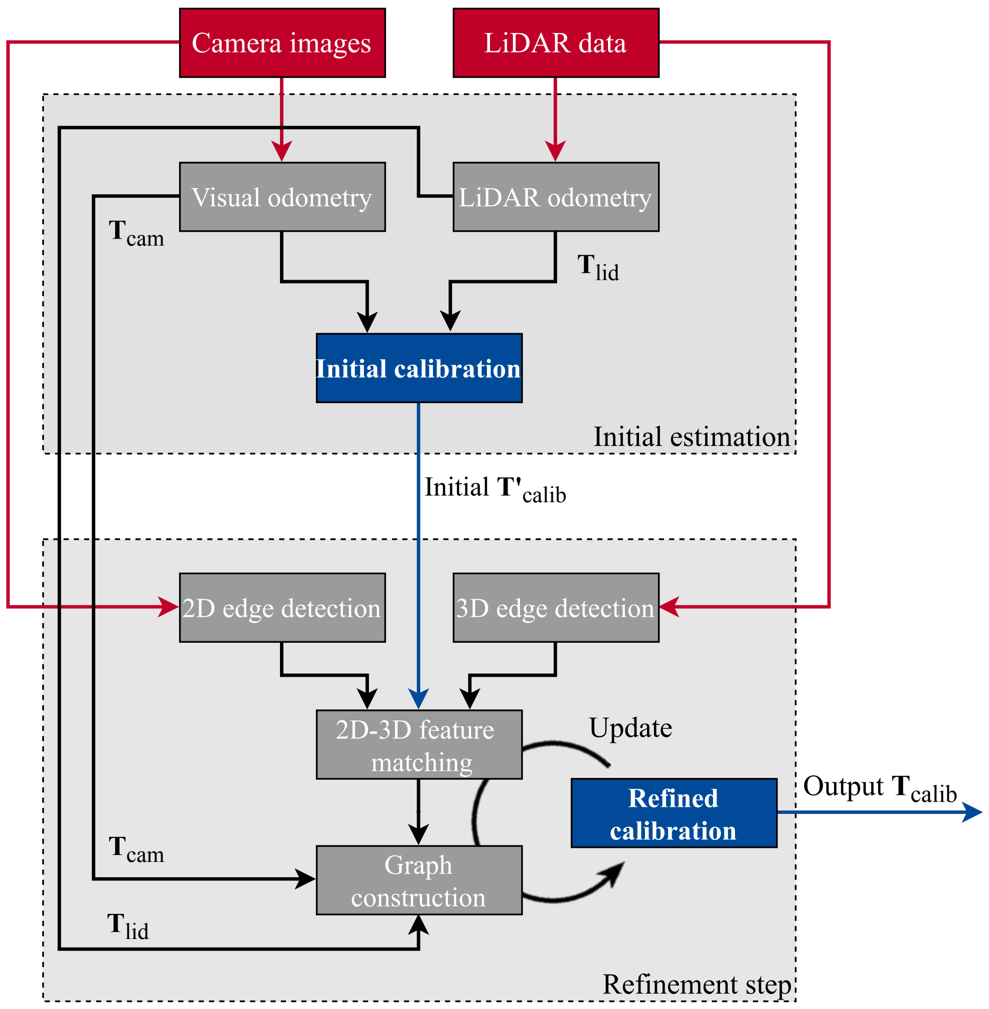


Figure 11: Overview of the proposed method. The red blocks represents the data captured by the sensors, while the building blocks of our workflow are marked in gray. The blue rectangles output the initial and refined calibration results.

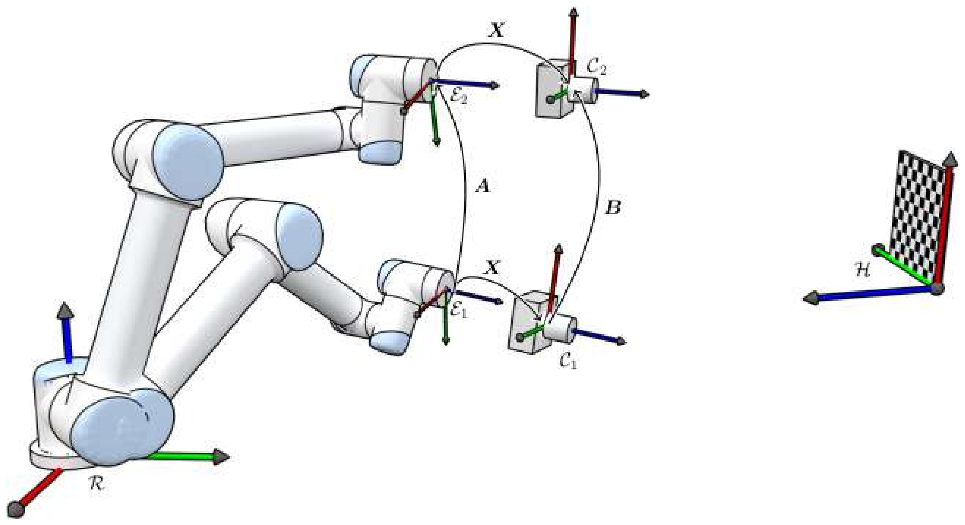
prerequisite of feature-based refinement is to define a feature that can be easily detected by both sensors. To calibrate a monocular camera and LiDAR sensor, we use one such characteristic landmark - an edge. The motivation to use an edge as a feature lies in the fact that it is frequently present in urban environments and can be detected in both camera and LiDAR sensor data. By overlaying the 3D edges extracted from the LiDAR point cloud to the camera pixel plane, we match them with the 2D edges on the image to get the feature correspondences.

In Figure 11, we illustrate the workflow of our approach. The input to our calibration algorithm is a set of images $\mathbb{I} = \{I_0, I_1, ..., I_N\}$ captured by the camera and a set of point clouds $\mathbb{Q} = \{\mathbf{Q}_0, \mathbf{Q}_1, ..., \mathbf{Q}_N\}$ acquired by the a LiDAR sensor. Our approach then calculates the transformation from camera reference frame to the reference frame of LiDAR. This is represented with a transform $\mathbf{T}_{\text{calib}} = (\mathbf{R}_{\text{calib}}, \mathbf{t}_{\text{calib}})$, consisting of rotation matrix $\mathbf{R}_{\text{calib}}$ and translation vector $\mathbf{t}_{\text{calib}}$.

To begin with, in the initial estimation phase we estimate the ego-motion of the camera and the LiDAR based on their recorded data, as explained in Section 3.2 and Section 3.3. Using the calculated $\mathbf{T}_{\text{cam}} = (\mathbf{R}_{\text{cam}}, \mathbf{t}_{\text{cam}})$ and $\mathbf{T}_{\text{lid}} = (\mathbf{R}_{\text{lid}}, \mathbf{t}_{\text{lid}})$, we get our initial calibration parameters $\mathbf{T'}_{\text{calib}}$, as presented in Section 4.2. Afterwards, we detect features in the recorded data - 2D edges in camera images and 3D edges in LiDAR point cloud. Based on the initial calibration, we then match these features with our similarity function, which we introduce in Section 4.3.2. Matched features, together with estimated ego-motions, are then used to construct a hyper-graph to optimize the calibration parameters. After several parameter updates, we obtain a refined calibration $\mathbf{T}_{\text{calib}}$. In the following sections, we describe how we utilize the results of the building blocks of our workflow to estimate extrinsic calibration parameters.

4.2 Initial Motion-Based Calibration

The *hand-eye calibration* problem [41] is a well-studied problem in robotics. The question is how to find a transformation between two rigidly coupled points based on a sequence of known poses of each point. The name originated from calibrating a camera ("eye") with the end effector of a robotic arm ("hand"). The hand-eye problem is illustrated in Figure 12. Its mathematical formulation resembles the problem of calibrating rigidly mounted sensors and is therefore used as the foundation in motion-based calibration. Similarly, given the position and orientation changes



Source: https://www.torsteinmyhre.name/snippets/robcam_calibration.html [accessed 11 August 2020]

Figure 12: Illustration of the hand-eye problem. When the robotic arm moves from position 1 to position 2, the transformation **A** of the end effector can be calculated by the changes in angles of joints. The transformation **B** of the camera can be estimated by visual odometry. With this, we recover the transformation **X** between the camera and the end effector using the relationship $\mathbf{AX} = \mathbf{XB}$.

observed by the camera $\mathbf{T}_{\text{cam}}$ and the LiDAR $\mathbf{T}_{\text{lid}}$, our goal is to determine their unknown relative transformation $\mathbf{T}_{\text{calib}}$. Solving $\mathbf{T}_{\text{cam}}\mathbf{T}_{\text{calib}} = \mathbf{T}_{\text{calib}}\mathbf{T}_{\text{lid}}$ for $\mathbf{T}_{\text{calib}}$ we get the extrinsic calibration parameters.

Having estimated odometry of the camera $\mathbb{T}_{\text{cam}} = \{\mathbf{T}^1_{\text{cam}}, \mathbf{T}^2_{\text{cam}}, ..., \mathbf{T}^N_{\text{cam}}\}$ from the captured set of images $\mathbb{I} = \{I_0, I_1, ..., I_N\}$ and the ego-motion of the LiDAR $\mathbb{T}_{\text{lid}} = \{\mathbf{T}^1_{\text{lid}}, \mathbf{T}^2_{\text{lid}}, ..., \mathbf{T}^N_{\text{lid}}\}$ from the set of acquired point clouds $\mathbb{Q} = \{\mathbf{Q}_0, \mathbf{Q}_1, ..., \mathbf{Q}_N\}$, we now recover the calibration parameters $\mathbf{T}_{\text{calib}} = (\mathbf{R}_{\text{calib}}, \mathbf{t}_{\text{calib}})$. Let $\mathbf{T}^i_{\text{lid}} = (\mathbf{R}^i_{\text{lid}}, \mathbf{t}^i_{\text{lid}})$ be the LiDAR motion and $\mathbf{T}^i_{\text{cam}} = (\mathbf{R}^i_{\text{cam}}, \bar{\mathbf{t}}^i_{\text{cam}})$ be the scaleless camera transformation at time i. Our algorithm determines both the scaled odometry of the camera $\mathbf{T}^i_{\text{cam}} = (\mathbf{R}^i_{\text{cam}}, \mathbf{t}^i_{\text{cam}})$ and the relative rotation and translation between the sensors. To achieve this, we extend the general hand-eye calibration problem to include the scale estimation for the camera translation. By decomposing the transformation into rotation and translation, we formulate:

$$\begin{gathered}\mathbf{R}^i_{\text{cam}} \cdot \mathbf{R}_{\text{calib}} = \mathbf{R}_{\text{calib}} \cdot \mathbf{R}^i_{\text{lid}}, \\ \mathbf{R}^i_{\text{cam}} \cdot \mathbf{t}_{\text{calib}} + s^i \cdot \overline{\mathbf{t}}^i_{\text{cam}} = \mathbf{R}_{\text{calib}} \cdot \mathbf{t}^i_{\text{lid}} + \mathbf{t}_{\text{calib}},\end{gathered} \tag{4}$$

where s^i is the scale factor of translation $\overline{\mathbf{t}}^i_{\text{cam}}$. As a result, obtaining the initial calibration parameters begins with solving the unconstrained optimization problem:

$$\mathbf{R}^*_{\text{calib}} = \underset{\mathbf{R}_{\text{calib}}}{\arg\min} \sum_i \left| \mathbf{R}^i_{\text{cam}} \mathbf{R}_{\text{calib}} - \mathbf{R}_{\text{calib}} \mathbf{R}^i_{\text{lid}} \right|, \tag{5}$$

where $-$ is the matrix subtraction and $|\cdot|$ is the matrix norm operator:

$$| \mathbf{A} | = \sqrt{\sum_i \sum_j |a_{ij}|^2}. \tag{6}$$

We obtain the solution with Levenberg–Marquardt algorithm [42] from the Google Ceres [43] optimization framework. For the purpose of calculating the scales s^i and translation calibration $\mathbf{t}$ we solve $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ for $\mathbf{x}$, with defined:

$$\mathbf{A} = \begin{bmatrix} \mathbf{R}^1_{\text{cam}} - \mathbf{I}_{3\times3} & \overline{\mathbf{t}}^1_{\text{cam}} & 0 & \cdots & 0 \\ \mathbf{R}^2_{\text{cam}} - \mathbf{I}_{3\times3} & 0 & \overline{\mathbf{t}}^2_{\text{cam}} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \mathbf{R}^N_{\text{cam}} - \mathbf{I}_{3\times3} & 0 & \cdots & 0 & \overline{\mathbf{t}}^N_{\text{cam}} \end{bmatrix}_{3N\times(N+3)}, \tag{7}$$

$$\mathbf{b} = \begin{bmatrix} \mathbf{R}_{\text{calib}} \cdot \mathbf{t}^1_{\text{lid}} \\ \mathbf{R}_{\text{calib}} \cdot \mathbf{t}^2_{\text{lid}} \\ \vdots \\ \mathbf{R}_{\text{calib}} \cdot \mathbf{t}^N_{\text{lid}} \end{bmatrix}_{3N\times1}, \tag{8}$$

$$\mathbf{x} = \begin{bmatrix} \mathbf{t}_{\text{calib}} \\ s^1 \\ \vdots \\ s^N \end{bmatrix}_{(3+N)\times 1} . \tag{9}$$

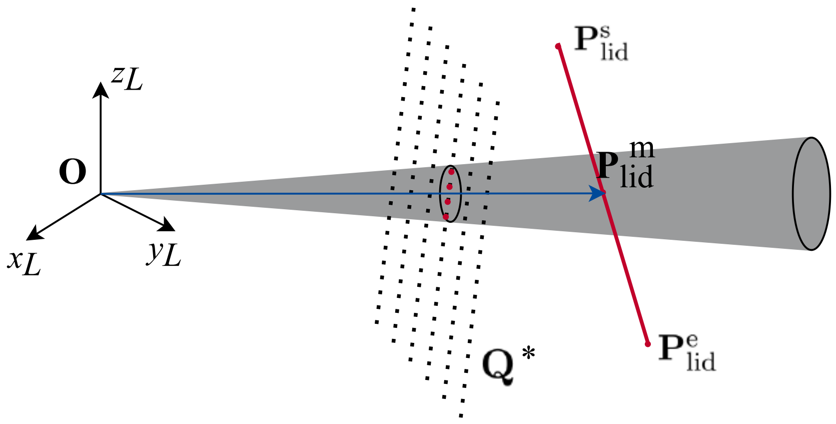
Solving $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ for $\mathbf{x}$, we finally get re-scaled camera translations by multiplying them with the newly obtained factors s^i, yielding $\mathbf{t}^i_{\text{cam}} = s^i \cdot \bar{\mathbf{t}}^i_{\text{cam}}$, and get the corrected camera odometry and initial calibration. This concludes the initialization step of our proposed method. We will re-use the estimated odometries while constructing the hyper-graph for our finer optimization of calibration parameters.

4.3 Refined Calibration Using Odometries and Feature Matching

The refinement step of our approach is based on matching the features extracted from the images and from the reconstructed 3D map of the environment. But prior to feature matching, the set of detected 3D LiDAR edges must be preprocessed. Also, the estimated odometries from the initial calibration step are utilized again in the refinement step during the construction of optimization hyper-graph.

4.3.1 3D Edges Preprocessing

The result of 3D edge extraction algorithm, explained in Section 3.5, is a set of 3D line segments $\{\mathbf{L}^1_{\text{lid}}, \mathbf{L}^2_{\text{lid}}, ..., \mathbf{L}^K_{\text{lid}}\}$. For a given LiDAR pose, we assess which of those edges are visible. Therefore, for each 3D line $\mathbf{L}_{\text{lid}} = \{\mathbf{P}^{\text{s}}_{\text{lid}}, \mathbf{P}^{\text{s}}_{\text{lid}}\}$, where $\mathbf{P}^{\text{s}}_{\text{lid}}$ and $\mathbf{P}^{\text{e}}_{\text{lid}}$ are start and end point, we perform an occlusion test. The output of the preprocessing step is a set of unoccluded 3D edges for a given LiDAR pose.



z_L
$\mathbf{O}$
x_L
y_L
$\mathbf{P}^{\mathrm{s}}_{\mathrm{lid}}$
$\mathbf{P}^{\mathrm{m}}_{\mathrm{lid}}$
$\mathbf{P}^{\mathrm{e}}_{\mathrm{lid}}$
$\mathbf{Q}^*$

Figure 13: Testing if a 3D edge is occluded. The point cloud of the reconstructed map is marked with $\mathbf{Q}^*$. The blue arrow indicates the direction of the viewing frustum which is represented with a grey cone. The cone marks the volume from which we abstract the points. Four red dots represent the points from 3D map within the viewing frustum. As their distance from LiDAR center is less than of the line midpoint $\mathbf{P}^{\mathrm{m}}_{\mathrm{lid}}$, we mark this line as occluded.

To begin with, we cannot perform the occlusion test using simple ray tracing because the dense point cloud is not a solid object. In certain scenarios, a ray can go through the planes represented by the point cloud without intersecting with any point. Also, checking if there is any point of the point cloud in small proximity to the ray is computationally intractable for relatively big point clouds consisting of millions of points.

Our solution is to portion the 3D space into a directed narrow FoV using frustum culling [33], as illustrated in Figure 13. We set its direction to $\overrightarrow{\mathbf{OP}^{\mathrm{m}}_{\mathrm{lid}}}$, where $\mathbf{P}^{\mathrm{m}}_{\mathrm{lid}} = (\mathbf{P}^{\mathrm{s}}_{\mathrm{lid}} + \mathbf{P}^{\mathrm{e}}_{\mathrm{lid}})/2$ marks the midpoint of the 3D line and $\mathbf{O}$ the center of LiDAR sensor. For every point in the viewing frustum within 15 cm from its centre, we calculate the distance from the LiDAR origin. If it is less then 80% of the depth of point $\mathbf{P}^{\mathrm{m}}_{\mathrm{lid}}$, then we consider 3D line $\mathbf{L}_{\mathrm{lid}}$ to be occluded by other objects viewed from $\mathbf{O}$. Instead of checking the viewing frustum in the direction of all the points along the 3D line segment, we reduce the computational complexity by approximating the occlusion test only for the midpoint. Also, checking the start and the end point was ineffective, because they are often not visible in sharp cornered structures.

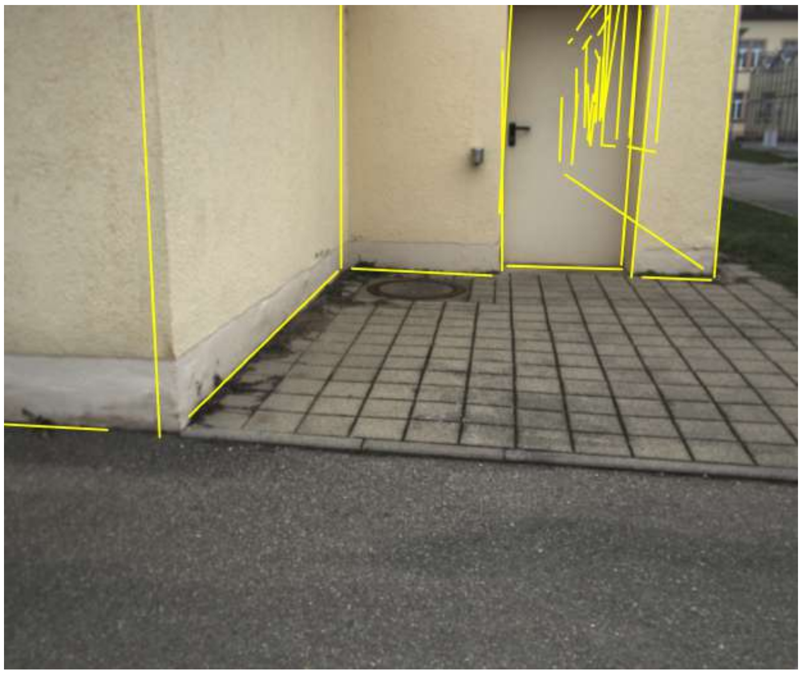
In Figure 14, we demonstrate the effect of running our occlusion test for one LiDAR pose. Results of projecting all the 3D lines extracted from the point cloud map to the image plane are shown in Figure 14(a). Noticeably, a substantial number of lines that are physically behind the building are present. After removing those lines we are left with the ones that are visible from the current position of the sensor (Figure 14(b)).

For the LiDAR pose from every timestamp i along the traversed trajectory, we evaluate which of the 3D edges extracted from $\mathbf{Q}^*$ are unoccluded. The 3D lines marked as occluded are removed from the set of candidate edges to be matched with the 2D edges from the image I_i. Doing so significantly reduces the search space in the 2D-3D matching process and makes the algorithm more efficient. Even further, it allows for higher robustness and fewer mismatches.

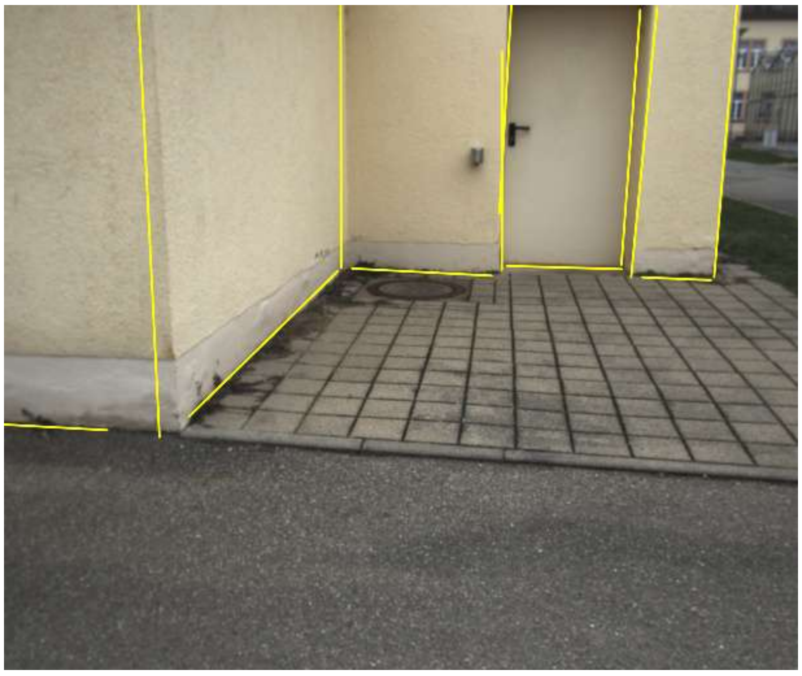
4.3.2 2D-3D Feature Matching

In every discrete timestamp i, we determine a set of 2D line segments extracted from the image I_i and a corresponding set of unoccluded edges calculated in the previous step. In this section, we show how we match 2D and 3D line segments, given a current estimate of calibration parameters, denoted with $\mathbf{R}_{\text{calib}}$ and $\mathbf{t}_{\text{calib}}$. As a result, we get a set of tuples consisting of the matched features $(\mathbf{l}_{\text{cam}}, \mathbf{L}_{\text{lid}})$.

In order to compare a 2D and a 3D line segment from two different reference frames, we first project the start and end point of each 3D line to the image plane using the current estimate of the calibration rotation and translation. This is formulated with:



(a) All detected 3D edges projected to image plane.



(b) Filtered 3D edges.

Figure 14: All the detected 3D edges (a) and the results of filtering based on the occlusion test (b).

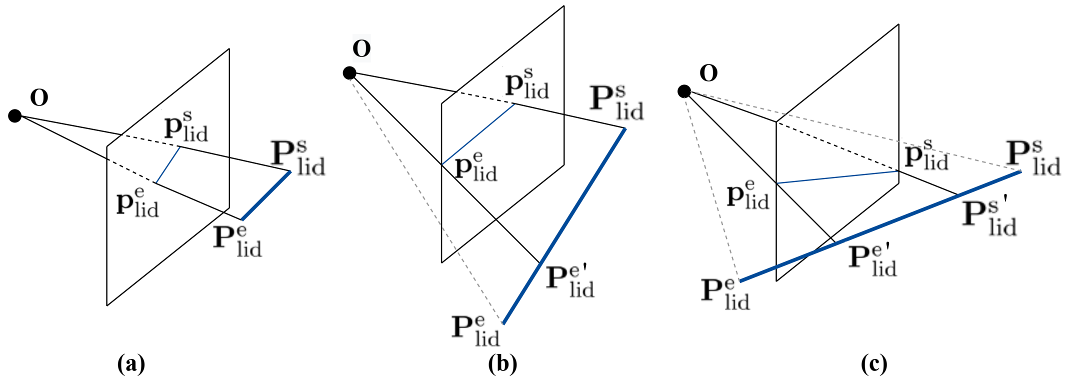


Figure 15: Projecting a 3D edge to the image plane.

$$\begin{aligned}\widehat{\mathbf{p}}^{s}_{lid} &= \begin{bmatrix} u^{s}_{lid} & v^{s}_{lid} & 1 \end{bmatrix}^{T} = \mathbf{K} \cdot (\mathbf{R}_{calib} \cdot \mathbf{P}^{s}_{lid} + \mathbf{t}_{calib}) \\ \widehat{\mathbf{p}}^{e}_{lid} &= \begin{bmatrix} u^{e}_{lid} & v^{e}_{lid} & 1 \end{bmatrix}^{T} = \mathbf{K} \cdot (\mathbf{R}_{calib} \cdot \mathbf{P}^{e}_{lid} + \mathbf{t}_{calib})\end{aligned} , \tag{10}$$

where $\mathbf{K}$ is the camera intrinsic matrix and the $\widehat{\cdot}$ indicates homogeneous pixel coordinates. Three cases are recognized, as visualized in Figure 15:

1. Both start and end point get projected to the image plane (Figure 15(a)). Therefore, we keep the whole 3D line segment as a 3D feature.

2. Only one endpoint gets projected to the image plane (Figure 15(b)). In this scenario, we must determine the furthest point $\mathbf{P}^{e}_{lid}{}'$ along the line segment that is also in camera FoV. For this, we calculate the intersection point of the projected 2D line segment with the image borders.

3. Neither start nor end point get projected to the plane. In Figure 15(c), we illustrate a degenerate case when even though both points are projected out of the image plane, a sub-segment is in the FoV. In these rare cases we clamp the points $\mathbf{P}^{s}_{lid}{}'$ and $\mathbf{P}^{e}_{lid}{}'$ to fit inside of the image borders.

Finally, we determine 2D-3D matches. Let $\mathbf{l}_{\text{cam}} = (\mathbf{p}^{\text{s}}_{\text{cam}}, \mathbf{p}^{\text{e}}_{\text{cam}})$ denote a detected 2D edge and $\mathbf{l}_{\text{lid}} = (\mathbf{p}^{\text{s}}_{\text{lid}}, \mathbf{p}^{\text{e}}_{\text{lid}})$ a two-dimensional projection of a 3D edge. For each feature pair, we define a similarity function between $\mathbf{l}_{\text{cam}}$ and $\mathbf{l}_{\text{lid}}$:

$$\operatorname{dist}(\mathbf{l}_{\text{cam}}, \mathbf{l}_{\text{lid}}) = \left(\theta, d, \frac{1}{l_{\text{overlap}}}\right), \tag{11}$$

where θ is the angle between the 2D lines, d is the average of point-to-line distances of both endpoints of $\mathbf{l}_{\text{lid}}$ from $\mathbf{l}_{\text{cam}}$ and l_{overlap} represents the length of their overlap.

The direction of a line segment with start point $\mathbf{p}^{\text{s}}$ and end point $\mathbf{p}^{\text{e}}$ is a vector $\mathbf{v} = (\mathbf{p}^{\text{e}} - \mathbf{p}^{\text{s}}) / \|\mathbf{p}^{\text{e}} - \mathbf{p}^{\text{s}}\|$. For both camera and LiDAR endpoints, we calculate the angle θ with the following formula:

$$\theta = \arccos\left(\mathbf{v}^{T}_{\text{lid}} \cdot \mathbf{v}_{\text{cam}}\right), \tag{12}$$

where $\mathbf{v}_{\text{lid}}$ and $\mathbf{v}_{\text{cam}}$ correspond to the LiDAR and the camera line segment vector, respectively. To calculate d, we first represent the camera line $\mathbf{l}_{\text{cam}}$ in its parametric form $ax + by + c = 0$. The coefficients are obtained with:

$$a = -(v^{\text{e}}_{\text{cam}} - v^{\text{s}}_{\text{cam}})/(u^{\text{e}}_{\text{cam}} - u^{\text{s}}_{\text{cam}}), \quad b = 1.0, \quad c = -a \cdot u^{\text{s}}_{\text{cam}} - v^{\text{s}}_{\text{cam}} \ . \tag{13}$$

The d is computed similarly to the distance proposed by Yu et al. [44] and represents the average point-to-line distances of both $\mathbf{p}^{\text{s}}_{\text{lid}}$ and $\mathbf{p}^{\text{e}}_{\text{lid}}$ to the 2D camera line:

$$d = \frac{|au^{\text{s}}_{\text{lid}} + bv^{\text{s}}_{\text{lid}} + c| + |au^{\text{e}}_{\text{lid}} + bv^{\text{e}}_{\text{lid}} + c|}{2\sqrt{a^2 + b^2}}. \tag{14}$$

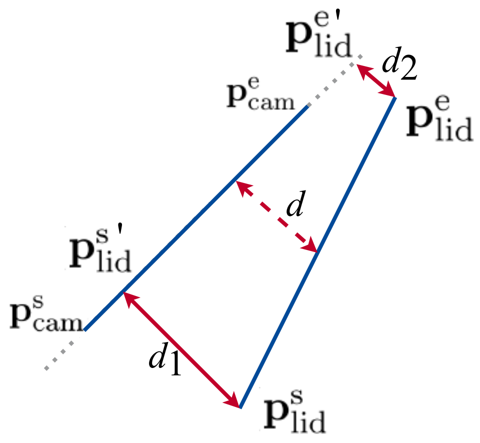

$\mathbf{p}_{\text{lid}}^{\text{e}\,'}$
d_2
$\mathbf{p}_{\text{cam}}^{\text{e}}$
$\mathbf{p}_{\text{lid}}^{\text{e}}$
d
$\mathbf{p}_{\text{lid}}^{\text{s}\,'}$
$\mathbf{p}_{\text{cam}}^{\text{s}}$
d_1
$\mathbf{p}_{\text{lid}}^{\text{s}}$

Figure 16: Our distance metric d between two lines represents the average of d_1 and d_2 point-to-line distances.

The reason for using this distance measure between two lines instead of utilizing the closest line-to-line distance is illustrated on Figure 16. In the scenario where one of the endpoints of $\mathbf{l}_{\text{lid}}$ is close to the image line but the other is not, line-to-line distance would still rate these two features as a match. Our average point-to-line distance takes into account the proximity of both endpoints to the camera line segment and thus provides more information about the alignment between the two lines.

Lastly, prior to calculating the length of overlap l_{overlap}, we must determine the projection of $\mathbf{p}_{\text{lid}}^{\text{s}}$ and $\mathbf{p}_{\text{lid}}^{\text{e}}$ to the 2D camera line, denoted with $\mathbf{p}_{\text{lid}}^{\text{s}}\prime$ and $\mathbf{p}_{\text{lid}}^{\text{e}}\prime$. That is achieved with orthogonal intersection of the camera line with a line that includes a projected 3D LiDAR endpoint. By computing the intersection of the camera line segment with the projected LiDAR line segment, we calculate l_{overlap} as depicted in Figure 17.

A searching strategy is utilized to find the first 3D feature correspondence that satisfies the threshold values of d, θ and l_{overlap} for each detected 2D edge. For parameters $d = \frac{\text{image width}}{20}, \theta = 10°$ and $l_{\text{overlap}} = \frac{\text{image width}}{10}$, we obtain a set of 2D-3D feature matches used for further refinement of calibration parameters, illustrated in Figure 18.

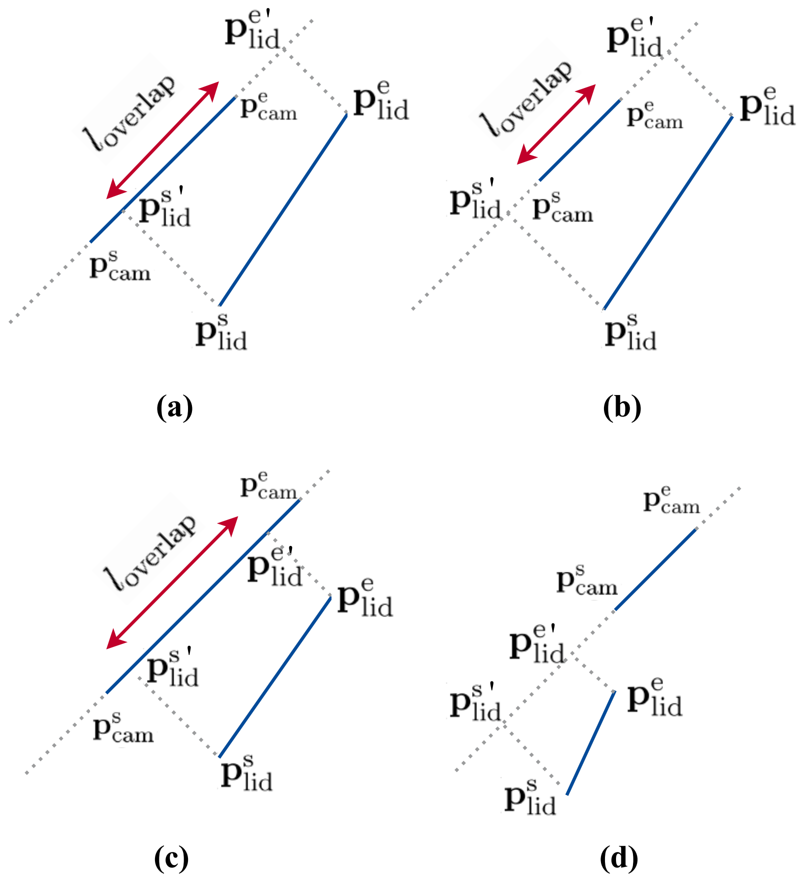
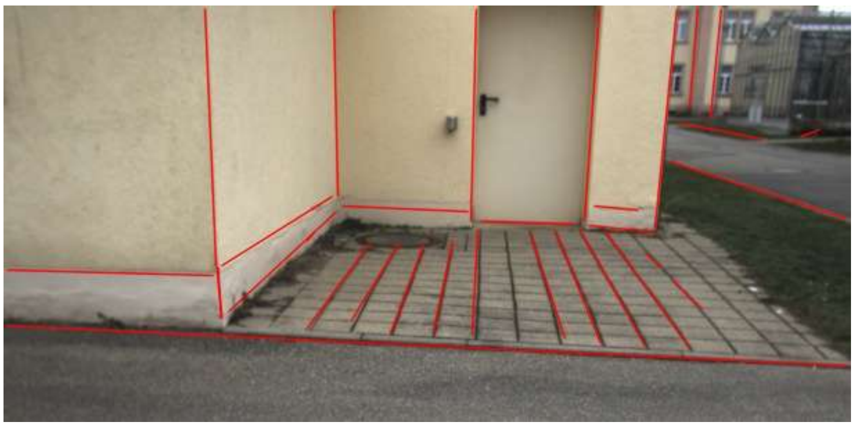


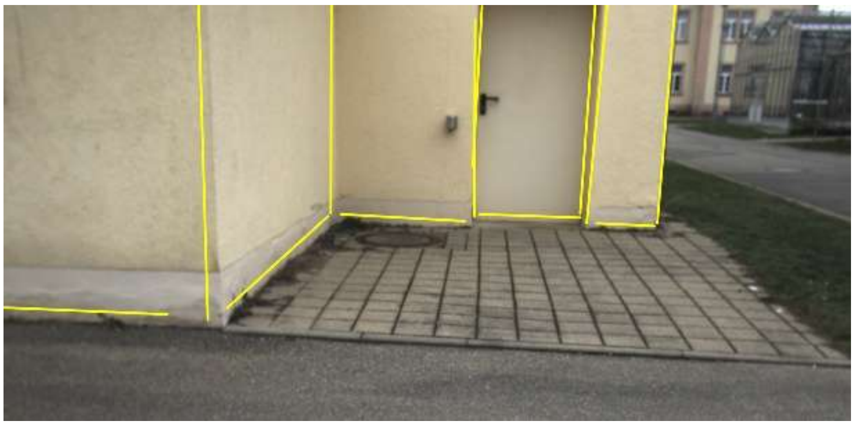
Figure 17: Various scenarios for calculating l_{overlap}. The blue line segments represent the 2D camera feature and the 2D projection of the LiDAR edge. The projection of the start and the end point of 2D LiDAR segment are denoted with $\mathbf{p}^{s}_{\text{lid}}\prime$ and $\mathbf{p}^{e}_{\text{lid}}\prime$. Degenerate cases where $l_{\text{overlap}} = 0$ (d) are also frequent and those 2D-3D pairs are no longer considered.

4.3.3 Graph Based Optimization

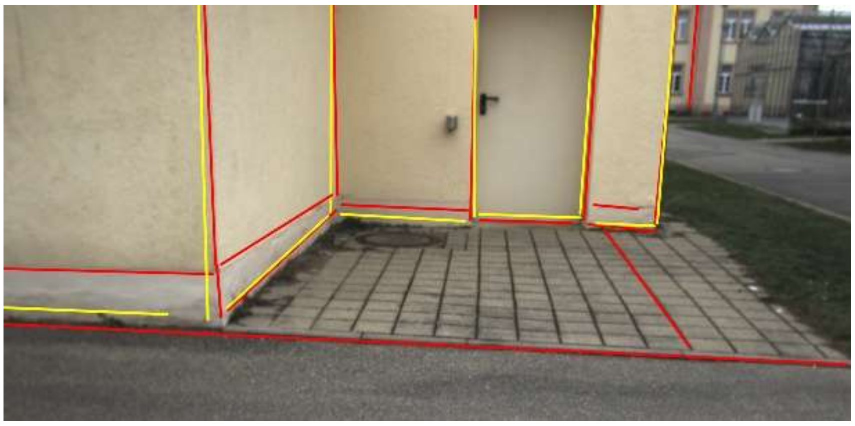
In the previous sections, we described how we determine camera and LiDAR ego-motion, use that estimate to obtain initial extrinsic calibration parameters and how we detect and match features observed in both sensors. The final step of our approach is to refine the calibration using all of the collected information.



(a) Detected edges in the camera image marked with red.



(b) 3D LiDAR edges projected to the pixel plane indicated in yellow.



(c) Matched 2D-3D edges.

Figure 18: The results of our 2D-3D feature matching algorithm based on the distance function. One can note a false match on the concrete tiles. We deal with outliers at a later step.

We model the problem as a hyper-graph, where each node represents either a sensor position or the calibration parameters. Edges connect nodes if there is a measurement between them. An edge is formed either to match a pair of 2D-3D observations or from odometry estimates for each sensor. A solution is a configuration of nodes that best complies with the measurements defined by the constraints.

Note that just like in the work of Kümmerle et al. [45], our approach is an extension from the classical graph-based SLAM algorithms. Since a measurement does not only depend on a pair of nodes but instead on a triplet (calibration parameters and nodes) it cannot be modelled by a graph but rather requires the use of a hyper-graph.

As the sensor setup moves, in each timestamp we add nodes representing camera and LiDAR poses to the graph. Let $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_n)^\top$ and $\mathbf{y} = (\mathbf{y}_0, \ldots, \mathbf{y}_n)^\top$ be vectors containing respectively camera and LiDAR positions, such that the following holds:

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{R}^i_{\text{cam}} \cdot \mathbf{x}_i + \mathbf{t}^i_{\text{cam}} \\ \mathbf{y}_{i+1} &= \mathbf{R}^i_{\text{lid}} \cdot \mathbf{y}_i + \mathbf{t}^i_{\text{lid}} \end{aligned}, \tag{15}$$

where $\mathbf{R}^i_{\text{cam}}, \mathbf{t}^i_{\text{cam}}$ and $\mathbf{R}^i_{\text{lid}}, \mathbf{t}^i_{\text{lid}}$ are estimated sensors movements from the timestamp i to $i+1$. For the sake of simplicity of notation, we use motion composition operators $\oplus$ and $\ominus$ introduced by Cox et al. [46] such that:

$$\mathbf{x}_{i+1} \ominus \mathbf{x}_i \overset{\text{def.}}{=} (\mathbf{x}_i)^{-1} \oplus \mathbf{x}_{i+1}. \tag{16}$$

Furthermore, let $\mathbf{k} = \{\mathbf{R}_{\text{calib}}, \mathbf{t}_{\text{calib}}\}$ be the pose of the camera relative to the reference frame of the LiDAR and let $\mathbf{l}_{\text{cam}}$ and $\mathbf{L}_{\text{lid}}$ be a 2D camera and 3D LiDAR feature, respectively.

We define the motion error function $\mathbf{e}^{\mathbf{m}}(\mathbf{x}_i, \mathbf{x}_{i+1}, \mathbf{y}_i, \mathbf{y}_{i+1}, \mathbf{k})$ to measure how well the ego-motion estimation from one timestamp to the next satisfies the calibration $\mathbf{k}$.

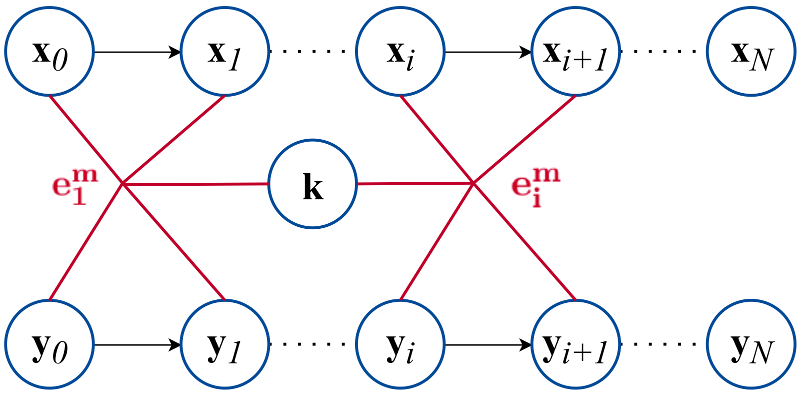


Figure 19: Illustration of the motion hyper-edge in the hyper-graph.

The smaller its value, the better its constraint is satisfied. We simplify the notation of error function in the following manner:

$$\mathbf{e}^{\mathbf{m}}\left(\mathbf{x}_i, \mathbf{x}_{i+1}, \mathbf{y}_i, \mathbf{y}_{i+1}, \mathbf{k}\right) \overset{\text{def.}}{=} \mathbf{e}_i^{\mathbf{m}}(\mathbf{x}, \mathbf{y}, \mathbf{k}). \tag{17}$$

Eventually, the motion error function $\mathbf{e}_i^{\mathbf{m}}(\mathbf{x}, \mathbf{y}, \mathbf{k})$, illustrated in Figure 19, is defined as:

$$\mathbf{e}_i^{\mathbf{m}}(\mathbf{x}, \mathbf{y}, \mathbf{k}) = (\mathbf{y}_{i+1} \ominus \mathbf{y}_i) \ominus ((\mathbf{x}_{i+1} \oplus \mathbf{k}) \ominus (\mathbf{x}_i \oplus \mathbf{k})). \tag{18}$$

Additionally, our observation error function $\mathbf{e}_i^{\mathbf{z}}(\mathbf{k})$ defines how well a 2D-3D feature match in timestamp i satisfies the calibration parameters between the sensors. We depict the observation hyper-edge in Figure 20.

We have introduced the transformation that projects a 3D line $\mathbf{L}_{\text{lid}}$ to a 2D line $\mathbf{l}_{\text{lid}}$ on the pixel plane using calibration parameters with Equation (10). Let an operator $\text{project}(\cdot, \mathbf{k})$ take a 3D line as an argument and calculate its 2D projection onto an image based on the calibration $\mathbf{k}$ and camera intrinsic matrix $\mathbf{K}$. If we mark

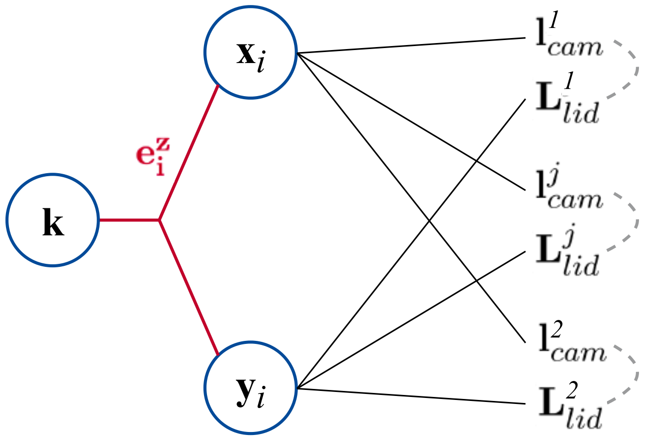


Figure 20: Illustration of the observation hyper-edge in the hyper-graph.

a 2D camera feature with $\mathbf{l}_{\text{cam}}$ and use the 2D-3D distance function outlined in Equation (11), we write the observation error function as:

$$\mathbf{e}_i^{\mathbf{z}}(\mathbf{k}) = \sum_{j \in J_i} \text{dist}\left(\mathbf{l}^j_{\text{cam}}, \text{project}(\mathbf{L}^j_{\text{lid}}, \mathbf{k})\right) \in \mathbb{R}^3, \tag{19}$$

where J_i represents the set of all 2D-3D features matches in timestamp i and $\sum$ is performed element-wise.

The goal of our maximum likelihood approach is to determine the configuration of camera poses $\mathbf{x}$, LiDAR poses $\mathbf{y}$ and calibration parameters $\mathbf{k}$ such that the negative log-likelihood F is minimized:

$$\mathbf{F}(\mathbf{x}, \mathbf{y}, \mathbf{k}) = \sum_{i}^{N} \mathbf{e}_i^{\mathbf{z}}(\mathbf{k})^\top \boldsymbol{\Omega}_i^{\mathbf{z}} \mathbf{e}_i^{\mathbf{z}}(\mathbf{k}) + \sum_{i}^{N-1} \mathbf{e}_i^{\mathbf{m}}(\mathbf{x}, \mathbf{y}, \mathbf{k})^\top \Omega_i^{\mathbf{m}} \mathbf{e}_i^{\mathbf{m}}(\mathbf{x}, \mathbf{y}, \mathbf{k}), \tag{20}$$

where N is the number of recorded poses and $\boldsymbol{\Omega}_\mathbf{i}^{\text{z}}$ and $\boldsymbol{\Omega}_\mathbf{i}^{\text{m}}$ the information matrix of the observations and motion estimations, respectively.

To solve the described optimization problem we use the graph optimization framework g2o [47]. After a few optimization iterations, the calibration parameters might change to the extent that some 2D-3D feature matches no longer satisfy the matching criteria. Also, new matches could be found with the newly determined calibration. Therefore, after every ten iterations, we re-run the 2D-3D feature matching algorithm, but with more strict thresholds. As a result of repeated optimization and feature association, we finally get the refined extrinsic calibration parameters.

5 Experiments

In this chapter, we evaluate the results of the presented approach and compare them to different calibration methods. We begin with introducing the datasets and sensor setups that we use in our experiments. In our first experiment, we compare the precision of our approach to the manual calibration, the state-of-the-art method and our previous feature-based work. We validate that our method performs comparably accurate to the state-of-the-art calibration technique that uses a checkerboard. Afterwards, we analyze the effects of varying levels of overlap in the FoV of the sensors. We show that our approach does not require sensors to have a large overlap in the FoV.

5.1 Sensor Setups and Datasets

We evaluate our approach on two different sensor setups. The first is a real-life integrated sensor setup, developed in the Autonomous Intelligent Systems laboratory of the Faculty of Engineering at the University of Freiburg. Additionally, we construct a custom modular sensor setup that facilitates various orientations between the sensors.

The real-life integrated sensor system, depicted in Figure 21, consists of a Blackfly S GigE monocular camera that captures RGB images with a 2448×2048 pixel resolution at a refresh rate of 5 Hz. A Velodyne Puck VLP-16 LiDAR with sixteen beams

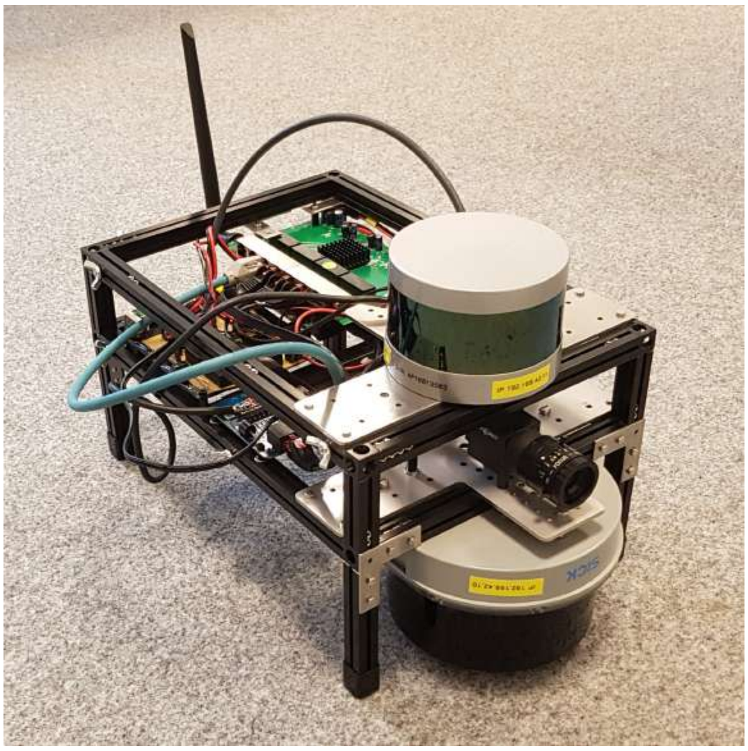
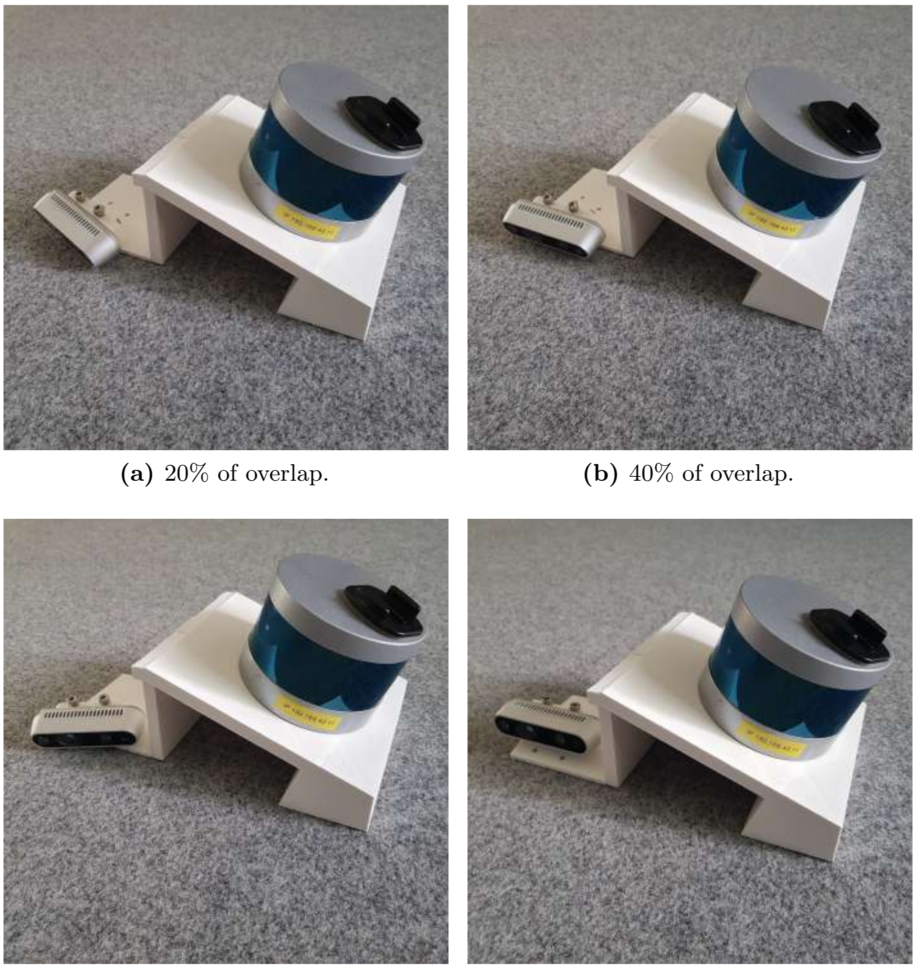


Figure 21: Experimental multi-sensor system used for extrinsic calibration consisting of a Velodyne Puck VLP-16 LiDAR, a SICK MRS1000 LiDAR and a Blackfly S GigE monocular camera.

producing horizontally 360° and vertically 30° FoV is mounted above. Below the camera, a SICK MRS1000 LiDAR is installed. All of our sensors are fixed rigidly. Because of the richer information it provides, only Velodyne LiDAR will be used in the evaluation.

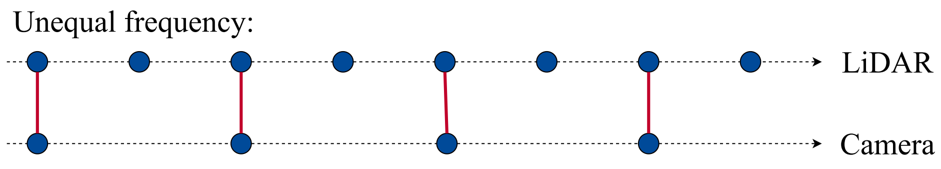
Furthermore, we have constructed a custom modular sensor setup (illustrated in Figure 22) that we use to assess the calibration results for varying levels of overlap between the camera and the LiDAR FoV. It includes the same Velodyne Puck VLP-16 LiDAR sensor and an Intel RealSense RGB-D camera which can be mounted in different orientations. It is important to note that even though the RealSense camera is a stereo camera system, we use only the RGB information from a single monocular camera. The sensors are connected directly to a laptop that records the sensor data.



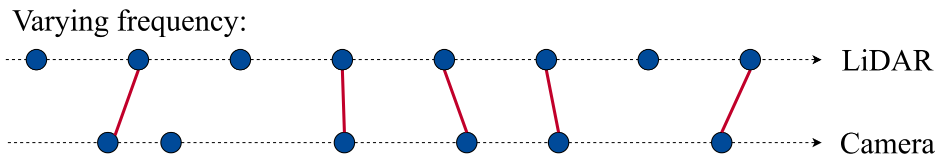
(a) 20% of overlap. **(b)** 40% of overlap. **(c)** 60% of overlap. **(d)** 100% of overlap.

Figure 22: Custom sensor setups with different camera placements yielding different levels of overlapping FoV.

We utilize the Robotic Operating System (ROS) [48] to record the data from sensors into a single file. Each received data frame from a sensor includes a timestamp, which we use to get synchronized data from the camera and the LiDAR sensor. However, the paired timestamps are only approximately synchronized, which brings one source of error to our results, independent of the approach. The approximate time synchronization is illustrated in Figure 23. The sensors of the real-life integrated sensor system have stable but unequal frequencies (Figure 23(a)). By setting the



(a) Synchronized data is acquired at the same time.

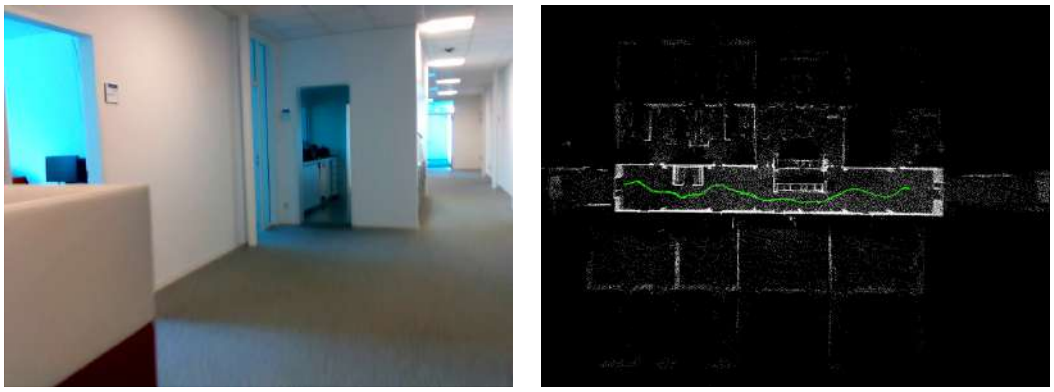


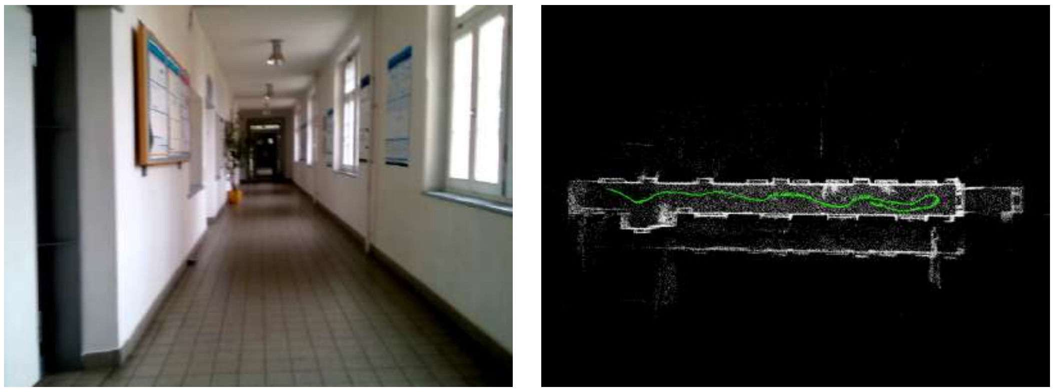
(b) Approximate synchronization of the data.

Figure 23: Approximate time synchronization of the recorded data from two sensors. The dotted line represents a stream of data from a single sensor, while blue points mark the time data was acquired. The red line connects a pair of data from approximately the same time.

frequency of the camera to be a factor of the refresh rate of the LiDAR we minimize the effects of this error. Contrary to the industrial camera on the real-life integrated sensor, our custom sensor setup is equipped with a camera that captures images at an unstable varying frequenct (Figure 23(b)). Therefore, in this sensor setup, the effects of time synchronization error are more pronounced.

We record a collection of datasets to evaluate the performance of our approach in urban environments. In Figures 24 and 25, we show sample images and reconstructed 3D map of each scene from our indoor and outdoor recordings, respectively. We also indicate the trajectory and its duration and length. For all four camera orientations that our custom modular sensor setup supports, we traverse these six scenes in a similar trajectory. In total, we created 24 datasets that we name by the compound of the scene label and the percentage of overlap in the FoV, i.e., "AiS hallway - 20", "AiS hallway - 40", "AiS hallway - 60", "AiS hallway - 100", "Narrow corridor - 20", "Narrow corridor - 40", etc. The scenes incorporate different types of structures typically present in an urban environment (walls, stairs, doors, benches, etc.). By testing on all these environments, we prove the validity of our approach on any real-life urban surrounding.


(a) "AiS hallway" scene. Recording duration ≈ 60 seconds. Distance traveled ≈ 30 m.


(b) "Narrow corridor" scene. Recording duration ≈ 60 seconds. Distance traveled ≈ 20 m.

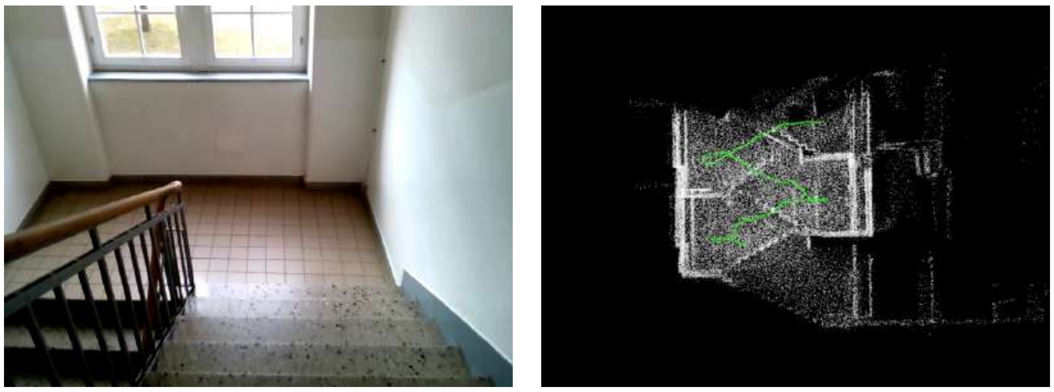
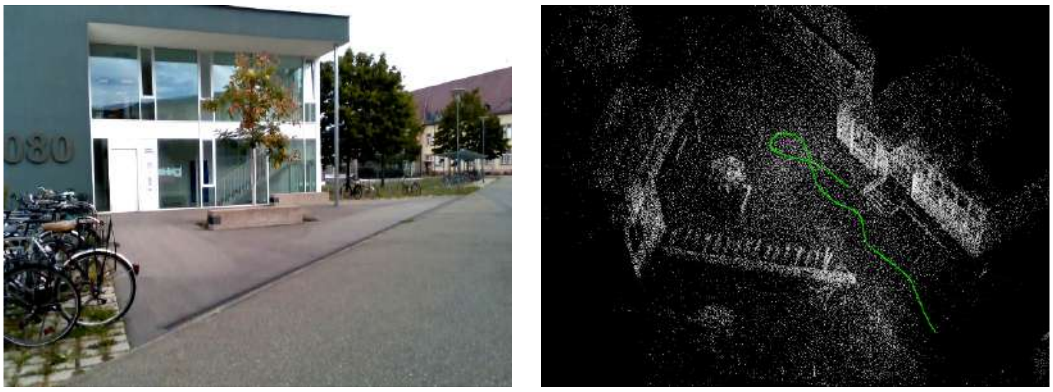
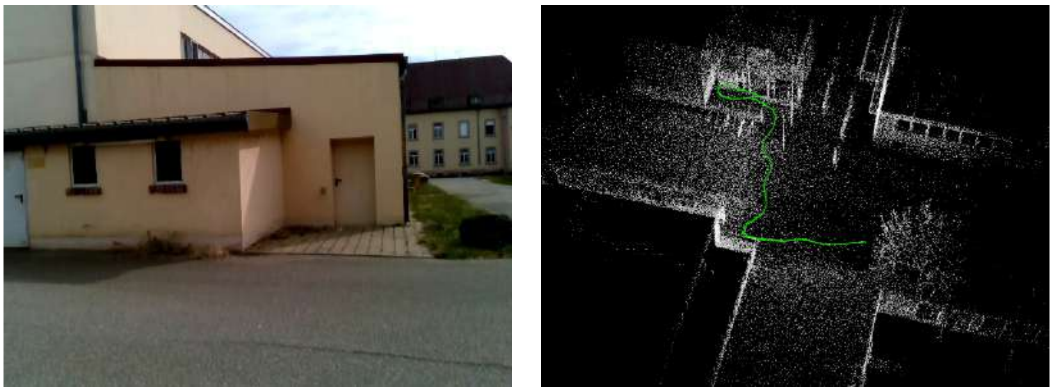

(c) "Stairs" scene. Recording duration ≈ 60 seconds. Distance traveled ≈ 25 m.

Figure 24: Sample footage (left) and reconstructed 3D map with the traversed trajectory marked in green (right) for three indoor scenes. For each dataset, we indicate an approximate duration and distance travelled.


(a) "AiS entrance" scene. Recording duration ≈ 60 seconds. Distance traveled ≈ 25 m.


(b) "Campus buildings" scene. Recording duration ≈ 60 seconds. Distance traveled ≈ 15 m.

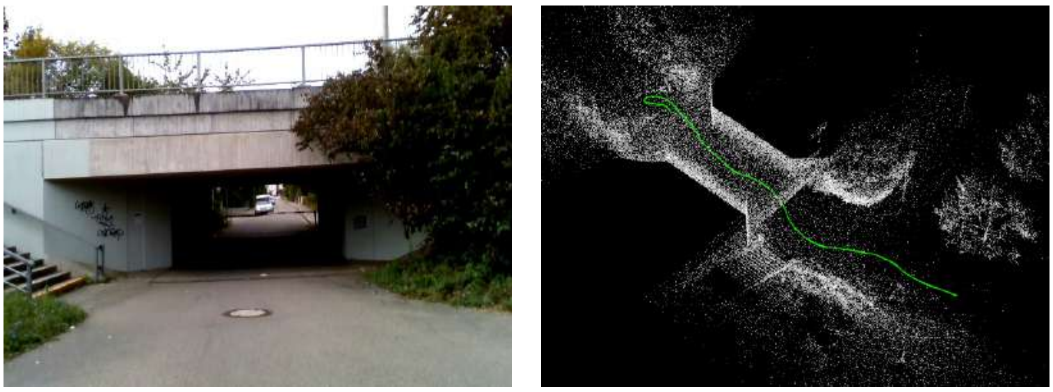

(c) "Tunnel" scene. Recording duration ≈ 50 seconds. Distance traveled ≈ 30 m.

Figure 25: Sample footage (left) and reconstructed 3D map with the traversed trajectory marked in green (right) for three outdoor scenes. For each dataset, we indicate an approximate duration and distance travelled.

5.2 Validation

In order to evaluate the precision of our approach, we compare our results to the parameters obtained by the manual calibration, the state-of-the-art method and our previous feature-based work. With all four approaches, we calibrate the camera and the LiDAR sensor on the real-world integrated sensor setup from Figure 21. It is important to emphasise that neither state-of-the-art, our previous feature-based work nor this thesis require an initial guess of the calibration parameters.

The most straightforward way to calibrate a camera and a LiDAR sensor is by manually measuring their relative orientation and translation. With reference to the datasheets of the sensors, the translation parameters can be determined accurately. However, measuring the rotation parameters by hand is error prone and imprecise due to the lack of highly-accurate angular measuring tools. Therefore, in this experiment, we use the manually measured translation parameters as ground truth translation.

The state-of-the-art approach, introduced in Section 3.1, uses a checkerboard as calibration object. Downsides of this approach are that it is dependent on the calibration object and requires observation of the checkerboard by both sensors at the same time. This means that the camera FoV needs to overlap with the FoV of the LiDAR, preventing the use of this method on some sensor setups. We consider the rotation parameters obtained with this method to be the ground truth values.

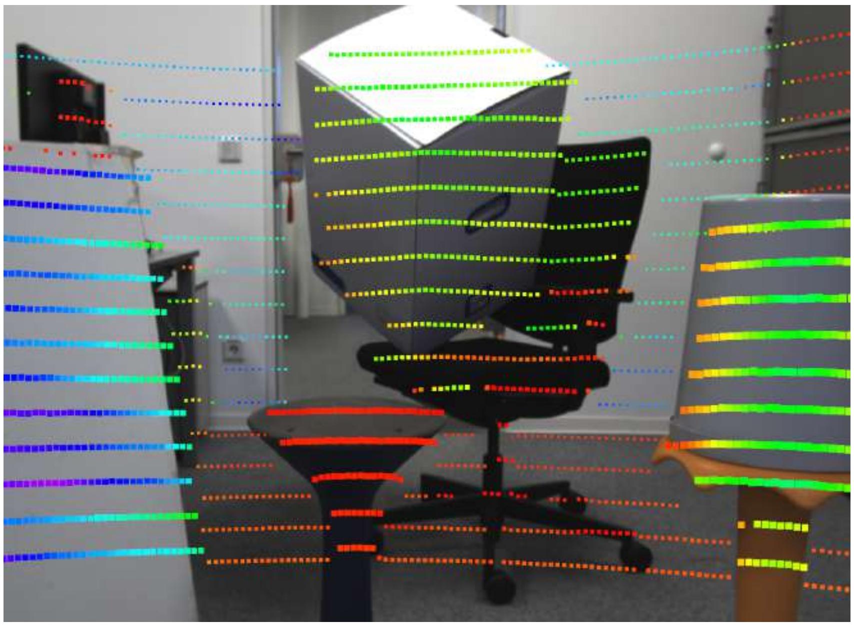
In our previous work, we developed a feature-based calibration algorithm that detects cornered structures from the scene. This way, our previous feature-based approach is independent on calibration objects, but the features still had to be simultaneously observed by both sensors. Therefore, the problem of overlapping FoV remains. Still, we include its results as a comparison of our approach to another targetless method.

In this thesis, we present a novel approach that does not require any calibration object nor a large overlap in the FoV of the sensors. We compare the calibration parameters obtained by all of these techniques and demonstrate how our approach surpasses the restrictions of other methods, while providing comparable performance. The values of roll, pitch and yaw angles and the translation on x, y and z axis obtained by manual measurements, the state-of-the-art approach, our previous feature-based work and this thesis are depicted in Table 1. Because of their easier interpretation, we show results in Euler angles. In essence, the difference of the angular values obtained by state-of-the-art, our feature-based approach and this thesis from the ground truth is small. The manually estimated angles are slightly off, as expected. The translation values differ a little further. Especially for the x and y axis, our method estimates results closer to the ground truth then the state-of-the-art and our previous feature-based work.

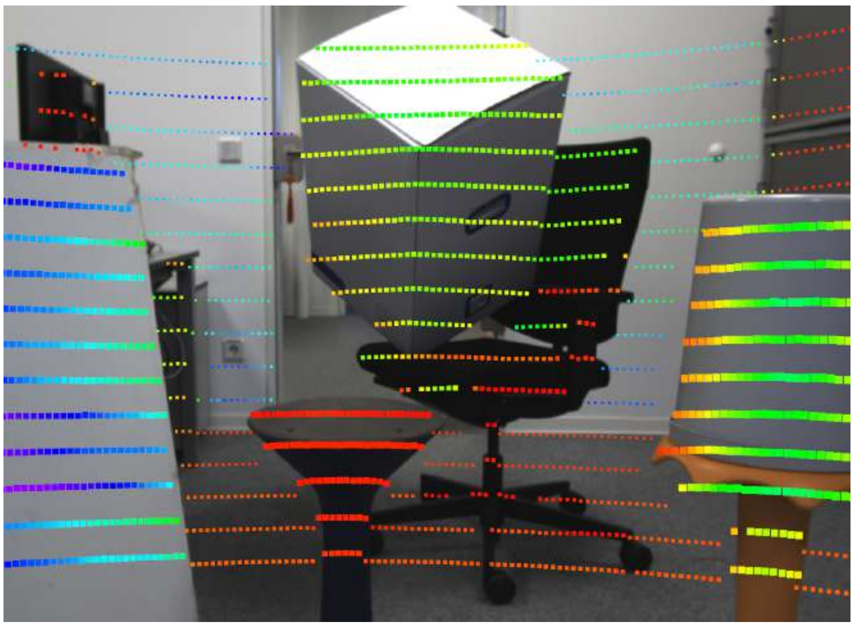
Table 1: The calibration parameters obtained by manual measurements, state-of-the-art, our previous work and this thesis. Marked in bold are the values we consider as ground truth.

	Manual	**SOTA**	**Previous Work**	**Thesis**
Roll (ϕ) [°]	90.0	**90.40**	90.60	90.34
Pitch (θ) [°]	0.0	**0.16**	-0.02	0.30
Yaw (ψ) [°]	90.0	**90.78**	90.72	90.52
t_x $[mm]$	**0.0**	-2.36	0.57	-0.80
t_y $[mm]$	**-78.3**	-81.55	-65.30	-78.0689
t_z $[mm]$	**-45.0**	-65.42	-62.27	-35.30

With this, we validate that our method performs comparably accurate to the state-of-the-art and estimates results close to the ground truth values. We also present qualitative results that better visualize the differences between the calibration parameters obtained by the four algorithms. Using the extrinsic calibration parameters, we fuse a captured camera image and a point cloud obtained by the 3D LiDAR to get RGB-D data for a local 3D scene. For performing the qualitative test of our results, we have set up a scene with objects of various shapes placed at different distances from the sensors, as shown in Figure 26.

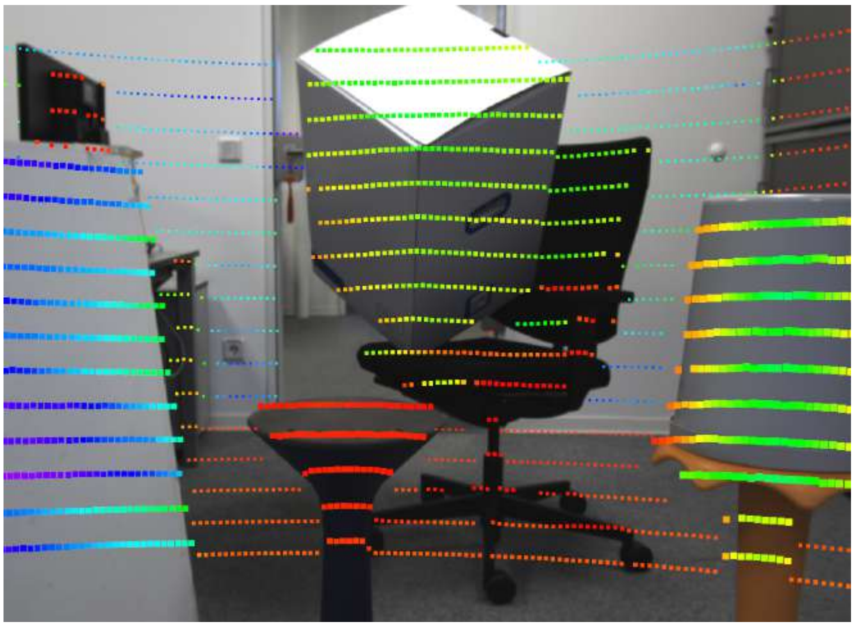


(a) Manual calibration results.



(b) State-of-the-art checkerboard calibration results.

Figure 26: Overlaying a simultaneously captured image and point cloud provides us with qualitative comparison of calibration methods. Manual calibration (a) shows apparent mismatch of the overlay. The state-of-the-art (b), our previous feature-based (c) and this thesis (c) calibration results provide comparable, accurate results.


(c) Results of our previous feature-based approach.

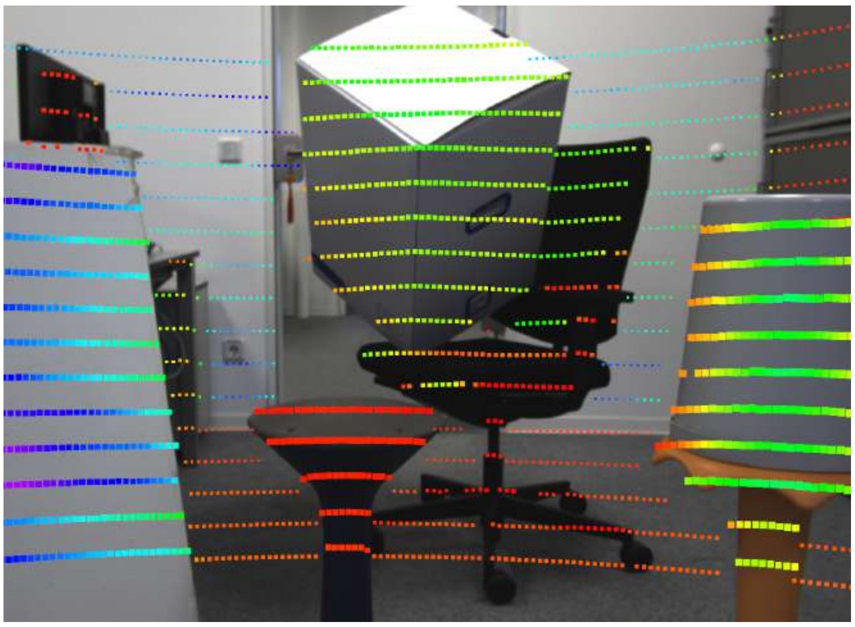

(d) Results of the proposed calibration method.

Figure 26: Overlaying a simultaneously captured image and point cloud provides us with qualitative comparison of calibration methods. Manual calibration (a) shows apparent mismatch of the overlay. The state-of-the-art (b), our previous feature-based (c) and this thesis (c) calibration results provide comparable, accurate results.

Looking at the borders of the objects, it is observable that the manual calibration (Figure 26(a)) is error prone. Small angular offsets are hard to estimate by hand but highly impact the quality of the results. However, the state-of-the-art and our previous feature-based work demonstrate accurate calibration results in Figure 26(b) and Figure 26(c), respectively. Finally, in Figure 26(d), we show that our results are accurate and comparable to the checkerboard and feature-based calibration, while the manual calibration is inferior.

5.3 Applicability

In this thesis, we present a novel approach that integrates the perceived data over time and thus no longer requires a large overlap in the FoV of the sensors. We provide experimental results that support this claim.

In this experiment, we test four different sensor orientations using our custom modular sensor setup from Figure 22 in three indoor and three outdoor scenes (Figures 24 and 25). This way, we prove that our approach can be used even on setups where sensors have only a small overlap in the FoV.

Knowing the exact model of our 3D printed sensor holder, we manually calculate the calibration parameters and consider those values as the ground truth. Still, we expect small deviation in the values of the rotation parameter induced during the mounting of the sensors. Contrary to the previous experiment, where we provided a detailed comparison for each calibration parameters, in this experiment we measure overall performance. Therefore, we present calibration errors more compactly. To compute the translation error, we use the Euclidean distance:

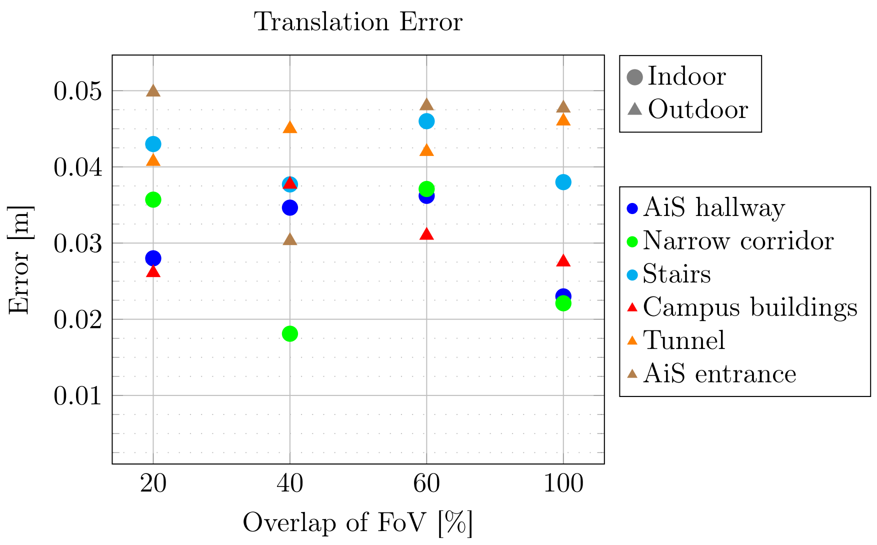
$$\text{error}_{\mathbf{t}} = \sqrt{(t_x - t_x^M)^2 + (t_y - t_y^M)^2 + (t_z - t_z^M)^2}, \tag{21}$$

where $\mathbf{t}^{\mathbf{M}}$ represents the manually estimated translation and $\mathbf{t}$ is the translation result of our approach. For the purpose of calculating the rotation error, we use quaternions. Let p and q be the rotation results of our approach and manual rotation parameters, respectively. The distance between these two rotations is the angle of the difference rotation represented by the unit quaternion $r = pq^*$, where * denotes the quaternion conjugate. We extract the angle from r with the following formula:

$$\theta = 2 \arccos(|p_1 q_1 + p_2 q_2 + p_3 q_3 + p_4 q_4|) = \text{error}_{\mathbf{R}}. \tag{22}$$

In Figure 27, we display the results of our method when the FoV of the camera and the LiDAR overlap by 20, 40, 60 and 100 percent. As previously mentioned, our custom sensor setup does not have an internal clock to synchronize data acquired by the camera and the LiDAR, but uses the approximate time synchronization. As a result, the translation errors in this experiment are slightly larger than with the real-life integrated sensor setup from the previous experiment. However, the rotation estimation offset from the manual calibration is consistent for all the setups. As we expect the manually measured rotation parameters to have a small offset, our approach actually provides more accurate rotational results. We demonstrate this in the following where we project the point cloud to the image plane using both manual and our estimated calibration.

We repeat the same qualitative experiment on the custom sensor setup that has the least overlapping FoV of the sensors (i.e. 20%) and the setup that has the highest rotational error (i.e. 40% overlap of the FoV of the sensors). The process of the previous analysis, where we projected a simultaneously acquired LiDAR point cloud to an image, cannot be used with these setup because of their small overlap of the FoVs. Therefore, we use the dense 3D map that our method creates by integrating LiDAR data over time. Results of projecting the dense 3D map to the camera image using manual calibration and the calibration results of our method are presented in



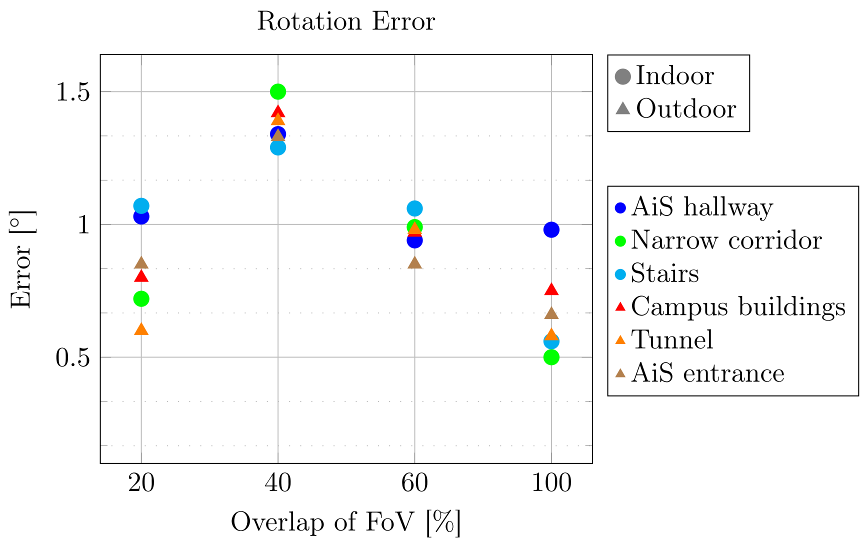
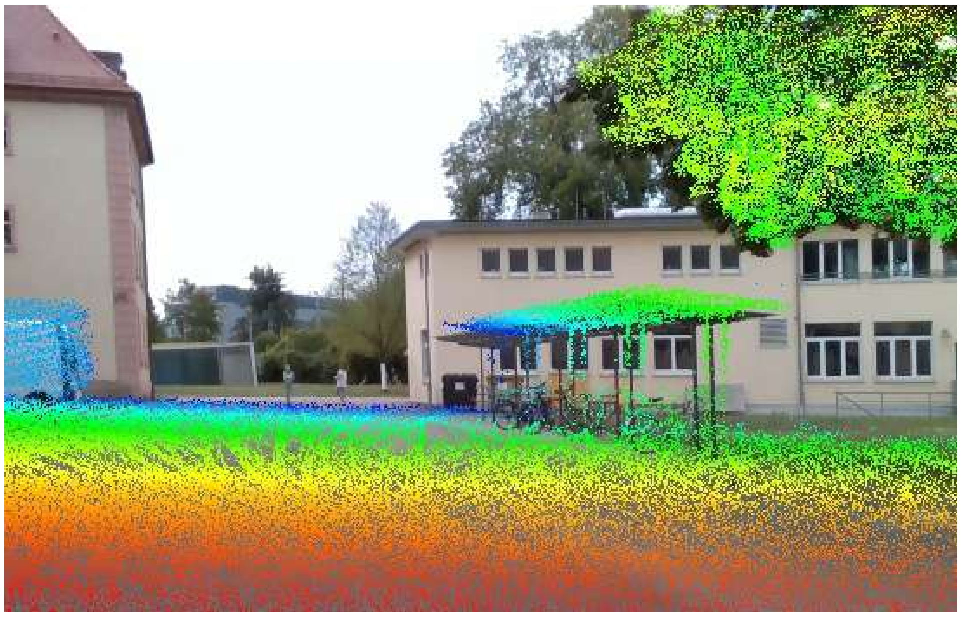


Figure 27: Translation and rotation error for different sensor setups in three indoor and three outdoor scenes.

Figures 28 and 29. Notably, the overlay using manual calibration is, in reality, a worse match to the image details compared to the calibration of our approach. This demonstrates that the rotational errors of our approach in Figure 27 are actually caused by the distance of manual calibration parameters to their true values. We also show that errors in translation do not noticeably affect the RGB-D results.


(a) Projecting the dense 3D map using manual calibration results.

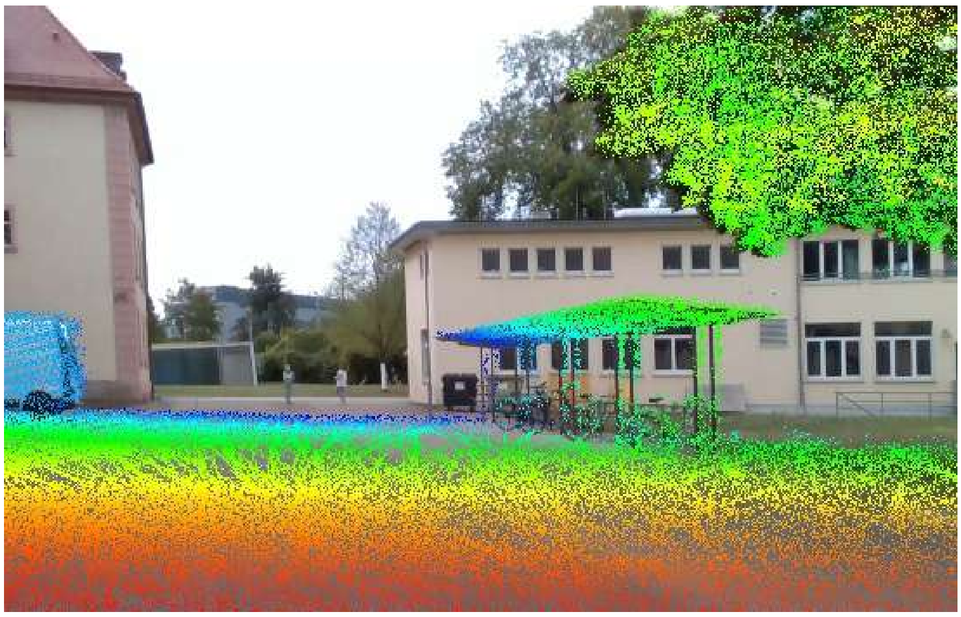
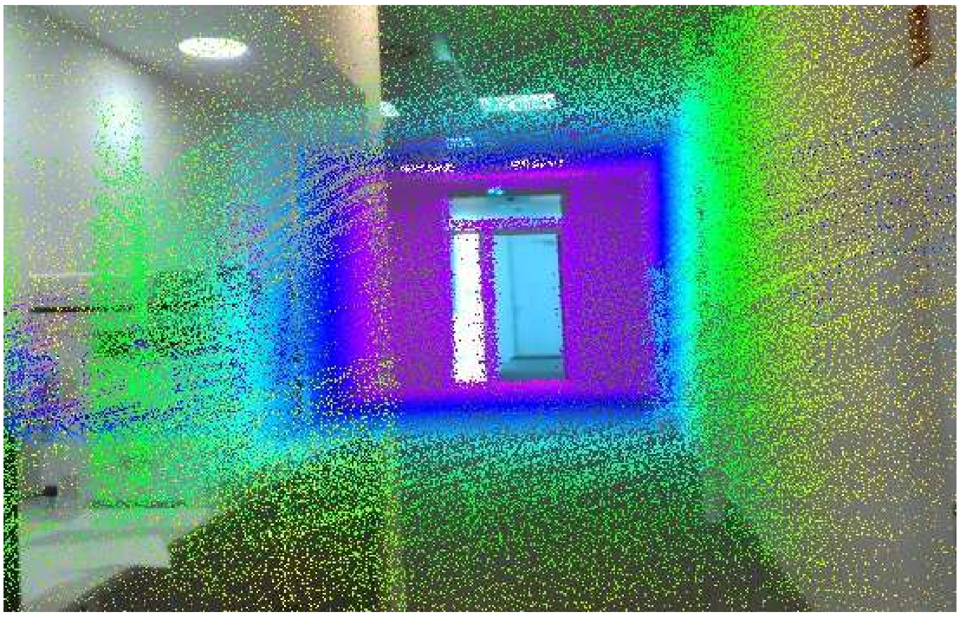

(b) Projecting the dense 3D map using calibration results of our method.

Figure 28: Qualitative comparison of results for the custom modular sensor on the "Campus buildings - 20" scene. Small angular measurement errors highly impact the quality of the results (a), especially for far away objects like the bicycle parking rain cover or the van.


(a) Projecting the dense 3D map using manual calibration results.

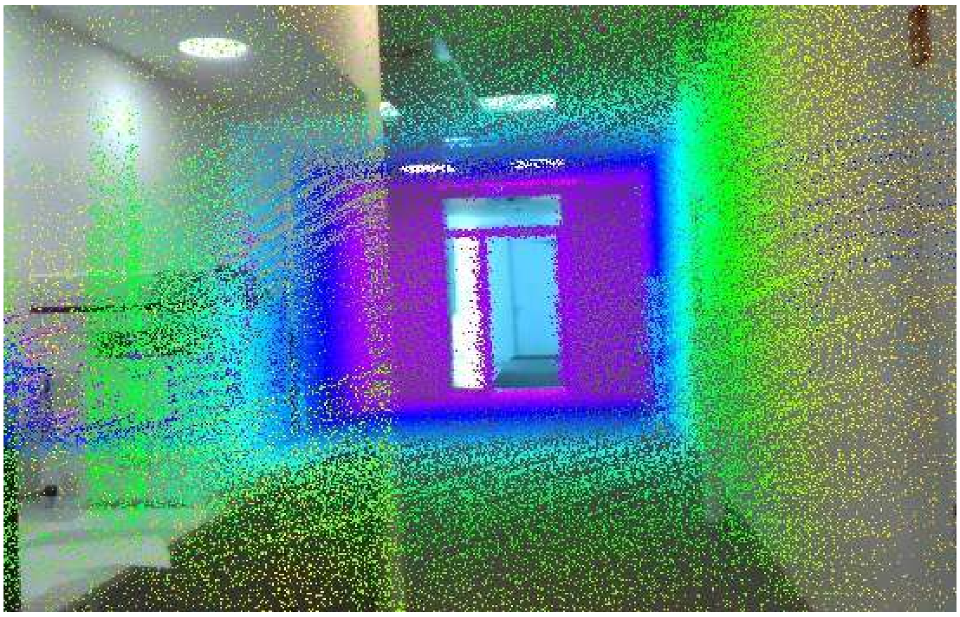

(b) Projecting the dense 3D map using calibration results of our method.

Figure 29: Qualitative comparison of results for the custom modular sensor setup with the highest rotational error (i.e. with 40% overlap of the FoV of the sensors) on the "AiS hallway - 40" dataset. The difference in results is especially apparent on the entrance door frame.

Overall, qualitative and quantitative comparison of our method to the manual calibration, state-of-the-art and our previous feature-based approach show, that their calibration accuracy is comparable. We also present data that indicate that the performance of our approach is independent of the overlap in the FoV between the sensors.

6 Conclusion

In this thesis, we presented a novel approach to calibrate a LiDAR and a monocular camera without the use of any calibration objects. Our method relies on first estimating the ego-motion of the sensors and then detecting and matching edges already present in the scene. The sensor setup is navigated through the environment and the algorithm initially estimates the odometry of each sensor and reconstructs a dense 3D map of the scene. Afterwards, we extract features from the images and from the reconstructed dense point cloud. We have shown how we determine the initial extrinsic calibration using on the calculated ego-motions and how we detect edges in both 2D images and 3D point clouds. In the refinement step, we create a hyper-graph from matched features and odometries whose optimization provides us with the refined extrinsic calibration parameters. Also, we demonstrate that our approach iteratively improves the calibration parameters, while simultaneously removing false 2D-3D feature associations.

The presented results illustrate the good performance and validity of the proposed approach compared to the state-of-the-art method. Contrary to checkerboard calibration, our method does not depend on any calibration objects or artificial scene modification. Furthermore, we show that our approach does not depend on overlapping FoV from the sensors, making it more versatile than the state-of-the-art approach.

One possible use case of our approach is quick on-site calibration where carrying a cumbersome calibration checkerboard is not always possible. Also, our calibration method can be used in scenarios where the sensors do not have an overlapping field of view which is not possible with traditional approaches.

A probable direction of future research based on this thesis is implementing features that can be detected by other sensor modalities. That way, we would create a modular approach to calibrate different types of sensors for which one can estimate the ego-motion, such as thermal cameras, radars, stereo cameras, etc. Furthermore, extending the types of features to different structures would enable our approach to be used not only in urban environments but also in forests and fields.

Our modular approach is not only limited to the sensor modalities that perceive visual information of the environment. It can also be used for calibrating sensors by only matching their estimated ego-motion. One example is the calibration between an IMU and a GPS sensor.

Overall, we hope to set the foundation for a modular calibration approach with this thesis. This means that our approach is implemented in such a way that it is easily extendable to suit different sensor modalities. By defining new features that can be detected by different sensor modalities, our method is able to calibrate other sensors, as well.

7 Acknowledgments

First and foremost, I would like to thank my thesis advisers Wera Winterhalter and Freya Fleckenstein of the Faculty of Engineering at the University of Freiburg. Even with the virus pandemic, they were always very assertive and supportive whenever I ran into a trouble spot or had a question about my research. They constantly allowed this thesis to be my own work but steered me in the right direction whenever they saw I needed it.

I would also like to thank my examiners Prof. Dr. Wolfram Burgard and Prof. Dr. Thomas Brox of the Faculty of Engineering at the University of Freiburg for their guidance during this work. The resources and equipment of the Autonomous Intelligent Systems laboratory were beneficial for my research.

Furthermore, I would like to thank the proofreaders who were involved in the validation for this thesis. Without their passionate participation and input, the verification could not have been successfully conducted.

Also, I am gratefully indebted to the German Academic Exchange Service (DAAD) for funding the full duration of my master's studies. This accomplishment would not have been achievable without them. Thank you.

Finally, I wish to express my most profound gratitude to my parents for providing me with constant support and encouragement throughout the years of my studies and in the process of researching and writing this thesis.

Bibliography

[1] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision.* Cambridge University Press, 2 ed., 2004.

[2] R. Behringer, S. Sundareswaran, B. Gregory, R. Elsley, B. Addison, W. Guthmiller, R. Daily, and D. Bevly, "The darpa grand challenge - development of an autonomous vehicle," in *IEEE Intelligent Vehicles Symposium, 2004*, pp. 226–231, 2004.

[3] V. Fremont and P. Bonnifait, "Extrinsic calibration between a multi-layer lidar and a camera," in *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pp. 214–219, 2008.

[4] J. Lázaro, J.-M. Lavest, C. Luna, A. Gardel, and I. Bravo, "Calibration of a high-accuracy 3-d coordinate measurement sensor based on laser beam and cmos camera," *Instrumentation and Measurement, IEEE Transactions on*, vol. 58, pp. 3341 – 3346, 10 2009.

[5] H. Aliakbarpour, P. Nunez, J. Prado, K. Khoshhal, and J. Dias, "An efficient algorithm for extrinsic calibration between a 3d laser range finder and a stereo camera for surveillance," in *2009 International Conference on Advanced Robotics*, pp. 1–6, 2009.

[6] P. Moghadam, M. Bosse, and R. Zlot, "Line-based extrinsic calibration of range and image sensors," in *2013 IEEE International Conference on Robotics and Automation*, pp. 3685–3691, 2013.

[7] Y. Zhuang, F. Yan, and H. Hu, "Automatic extrinsic self-calibration for fusing data from monocular vision and 3-d laser scanner," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 7, pp. 1874–1876, 2014.

[8] Qilong Zhang and R. Pless, "Extrinsic calibration of a camera and laser range finder (improves camera calibration)," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2301–2306 vol.3, 2004.

[9] P. Fuersattel, C. Plank, A. Maier, and C. Riess, "Accurate laser scanner to camera calibration with application to range sensor evaluation," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, 12 2017.

[10] G. Pandey, J. McBride, S. Savarese, and R. Eustice, "Extrinsic calibration of a 3d laser scanner and an omnidirectional camera," 01 2010.

[11] W. Wang, K. Sakurada, and N. Kawaguchi, "Reflectance intensity assisted automatic and accurate extrinsic calibration of 3d lidar and panoramic camera using a printed chessboard," *Remote Sensing*, vol. 9, p. 851, 08 2017.

[12] D. Scaramuzza, A. Harati, and R. Siegwart, "Extrinsic self calibration of a camera and a 3d laser range finder from natural scenes," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4164–4169, 2007.

[13] R. Ishikawa, T. Oishi, and K. Ikeuchi, "Lidar and camera calibration using motions estimated by sensor fusion odometry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7342–7349, 2018.

[14] Z. Taylor and J. Nieto, "Motion-based calibration of multimodal sensor extrinsics and timing offset estimation," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1215–1229, 2016.

[15] S. Esquivel, F. Woelk, and R. Koch, "Calibration of a multi-camera rig from non-overlapping views," vol. 4713, pp. 82–91, 09 2007.

[16] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, p. 381–395, June 1981.

[17] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[18] Y. Zheng, Y. Kuang, S. Sugimoto, K. Åström, and M. Okutomi, "Revisiting the pnp problem: A fast, general and optimal solution," in *2013 IEEE International Conference on Computer Vision*, pp. 2344–2351, 2013.

[19] C. G. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey Vision Conference*, 1988.

[20] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," vol. 3951, 07 2006.

[21] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," vol. 3951, 07 2006.

[22] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 404–417, Springer Berlin Heidelberg, 2006.

[23] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011.

[24] P. Fernández Alcantarilla, “Fast explicit diffusion for accelerated features in nonlinear scale spaces,” 09 2013.

[25] S. A. K. Tareen and Z. Saleem, “A comparative analysis of sift, surf, kaze, akaze, orb, and brisk,” in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pp. 1–10, 2018.

[26] M. J. M. M. Mur-Artal, Raúl and J. D. Tardós, “ORB-SLAM: a versatile and accurate monocular SLAM system,” *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[27] D. Nister, “An efficient solution to the five-point relative pose problem,” in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2, pp. II–195, 2003.

[28] R. I. Hartley, “In defense of the eight-point algorithm,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 6, pp. 580–593, 1997.

[29] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. USA: Cambridge University Press, 2 ed., 2003.

[30] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, “Comparing icp variants on real-world data sets,” *Autonomous Robots*, 04 2013.

[31] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *International Journal of Robotics Research (IJRR)*, 2013.

[32] J. Zhang and S. Singh, “Loam : Lidar odometry and mapping in real-time,” *Robotics: Science and Systems Conference (RSS)*, pp. 109–111, 01 2014.

[33] R. B. Rusu and S. Cousins, “3d is here: Point cloud library (pcl),” in *2011 IEEE International Conference on Robotics and Automation*, pp. 1–4, 2011.

[34] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, p. 11–15, Jan. 1972.

[35] J. H. Lee, S. Lee, G. Zhang, J. Lim, W. K. Chung, and I. H. Suh, "Outdoor place recognition in urban environments using straight lines," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5550–5557, 2014.

[36] N. Xue, S. Bai, F. Wang, G.-S. Xia, T. Wu, and L. Zhang, "Learning attraction field representation for robust line segment detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[37] R. Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall, "Lsd: A fast line segment detector with a false detection control," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, pp. 722–32, 04 2010.

[38] T. Hackel, J. D. Wegner, and K. Schindler, "Contour detection in unstructured 3d point clouds," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1610–1618, 2016.

[39] A. Sampath and J. Shan, "Segmentation and reconstruction of polyhedral building roofs from aerial lidar point clouds," *IEEE T. Geoscience and Remote Sensing*, vol. 48, pp. 1554–1567, 03 2010.

[40] X. Lu, Y. Liu, and K. Li, "Fast 3d line segment detection from unorganized point cloud," 01 2019.

[41] Y. C. Shiu and S. Ahmad, "Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form ax=xb," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 16–29, 1989.

[42] H. P. Gavin, "The levenberg-marquardt method for nonlinear least squares curve-fitting problems," 2013.

[43] S. Agarwal, K. Mierle, and Others, “Ceres solver.” `http://ceres-solver.org`.

[44] H. Yu, W. Zhen, W. Yang, J. Zhang, and S. Scherer, “Monocular camera localization in prior lidar maps with 2d-3d line correspondences,” 2020.

[45] R. Kümmerle, G. Grisetti, and W. Burgard, “Simultaneous parameter calibration, localization, and mapping,” vol. 26, pp. 3716–3721, 09 2011.

[46] I. J. Cox and G. T. Wilfong, eds., *Autonomous Robot Vehicles.* Berlin, Heidelberg: Springer-Verlag, 1990.

[47] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “G2o: A general framework for graph optimization,” in *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613, 2011.

[48] Stanford Artificial Intelligence Laboratory et al., “Robotic operating system.”