UNIVERSITÀ DEGLI STUDI DI PADOVA

Sede Amministrativa: Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

SCUOLA DI DOTTORATO DI RICERCA IN: Ingegneria dell'Informazione

INDIRIZZO: Scienza e Tecnologia dell'Informazione

CICLO: XXVII

A NON-PARAMETRIC CALIBRATION ALGORITHM FOR DEPTH SENSORS EXPLOTING RGB CAMERAS

Direttore della Scuola:	*Ch.mo Prof. Matteo Bertocco*
Coordinatore d'indirizzo:	*Ch.mo Prof. Carlo Ferrari*
Supervisore:	*Ch.mo Prof. Emanuele Menegatti*

Dottorando: *Filippo Basso*

*"We can only see a short distance ahead,
but we can see plenty there that needs to be done."*

— Alan Turing

Abstract

Range sensors are common devices on modern robotic platforms. They endow the robot with information about distance and shape of the objects in the sensors field of view. In particular, the advent in the last few years of consumer RGB-D sensors such as the Microsoft Kinect, has greatly fostered the development of depth-based algorithms for robotics. In fact, such sensors can provide a large quantity of data at a relatively low price.

In this thesis three different calibration problems for depth sensors are tackled. The first original contribution to the state of the art is an algorithm to recover the axis of rotation of a 2D laser range finder (LRF) mounted on a rotating support. The key difference with other approaches is the use of kinematics point-plane constraints to estimate the pose of the LRF with respect to a static camera, and screw decomposition to recover the axis of rotation. The correct reconstruction of a small indoor environment after calibration validates the proposed algorithm.

The second and most important original contribution of the thesis is a fully automatic two-steps calibration algorithm for structured-light depth sensors (e.g. Kinect). The key novelty of this work is the separation of the depth error into two components, corrected with functions estimated on a pixel-basis. This separation, validated by experimental observations, allows to dramatically reduce the number of parameters in the final non-linear minimization and, consequently, the time for the solution to converge to the global minimum. The depth images of a test set corrected using the obtained calibration parameters are analyzed and compared to the ground truth. The comparison shows that they differ from the real ones just for an unpredictable noise. A qualitative analysis of the fusion between depth and RGB data further confirms the effectiveness of the approach. Moreover, a ROS package for both calibrating and correcting the Kinect data has been released as open source.

The third contribution reported in the thesis is a new distributed calibration algorithm for networks composed by cameras and already-calibrated depth sensors. A ROS package implementing the proposed approach has been developed and is available for free as a part of a big open source project for people tracking: OpenPTrack. The developed package is able to calibrate networks composed by a dozen sensors in real-time (i.e., batch processing is not needed), exploiting plane-to-plane constraints and non-linear least squares optimization.

Sommario

I sensori di profondità sono dispositivi comuni sui robot moderni. Essi forniscono al robot informazioni sulla distanza e sulla forma degli oggetti nel loro campo di visione, permettendogli di agire di conseguenza. In particolare, l'arrivo negli ultimi anni di sensori RGB-D di consumo come Microsoft Kinect, ha favorito lo sviluppo di algoritmi per la robotica basati su dati di profondità. Di fatto, questi sensori sono in grado di generare una grande quantità di dati ad un prezzo relativamente basso.

In questa tesi vengono affrontati tre diversi problemi riguardanti la calibrazione di sensori di profondità. Il primo contributo originale allo stato dell'arte è un algoritmo per stimare l'asse di rotazione di un laser range finder (LRF) 2D montato su un supporto rotante. La differenza chiave con gli altri approcci è l'utilizzo di vincoli punto-piano derivanti dalla cinematica per stimare la posizione del LRF rispetto ad una videocamera fissa, e l'uso di una *screw decomposition* per stimare l'asse di rotazione. La corretta ricostruzione di una stanza dopo la calibrazione valida l'algoritmo proposto.

Il secondo e più importante contributo originale di questa tesi è un algoritmo completamente automatico per la calibrazione di sensori di profondità a luce strutturata (ad esempio Kinect). La chiave di questo lavoro è la separazione dell'errore di profondità in due componenti, entrambe corrette pixel a pixel. Questa separazione, validata da osservazioni sperimentali, permette di ridurre sensibilmente il numero di parametri nell'ottimizzazione finale e, di conseguenza, il tempo necessario affinché la soluzione converga al minimo globale. Il confronto tra le immagini di profondità di un test set, corrette con i parametri di calibrazione ottenuti, e quelle attese, dimostra che la differenza tra le due è solamente di una quantità casuale. Un'analisi qualitativa della fusione tra dati di profondità e RGB conferma ulteriormente l'efficacia dell'approccio. Inoltre, un pacchetto ROS per calibrare e correggere i dati generati da Kinect è disponibile open source.

Il terzo contributo riportato nella tesi è un nuovo algoritmo distribuito per la calibrazione di reti composte da videocamere e sensori di profondità già calibrati. Un pacchetto ROS che implementa l'algoritmo proposto è stato rilasciato come parte di un grande progetto open source per il tracking di persone: OpenPTrack. Il pacchetto sviluppato è in grado di calibrare reti composte da una decina di sensori in tempo reale (non è necessario processare i dati in un secondo tempo), sfruttando vincoli piano-piano e un'ottimizzazione non lineare.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

At home as well as at work, robots have the potential to dramatically improve the quality of our lives. They are rapidly evolving from mere production tools to human-aware machines designed to interact with the surrounding environment. One of the key capabilities that lets today's robots pervade our human-centric world is perception. Vision has been, and still is, one of the fundamental ways for both humans and robots to gather what is happening around them, but while humans are able to mix vision with experience to interpret what they see, robots are not. For this reason, robots are typically equipped with other sensors rather than cameras, that let them provide for the missing experience.

Currently, depth sensors are some of the most accurate, yet useful, perception devices for robots. Many different technologies exist for depth sensors and, among them, many different sensor types have been developed and commercialized. 2D laser range finders, for example, are depth sensors able to provide range measurements up to 10-30 meters for the obstacles that intersect the scan plane. In robotics, such sensors are typically used in simultaneous localization and mapping (SLAM) [14, 40, 68] and object/people tracking [17, 35, 46]. Unfortunately, while being really accurate, these sensors are confined to measure ranges on a plane. Actually, 3D scanners exist, however they have typically prohibitive costs[1]. Therefore, if a 3D view of the environment is needed, a solution often exploited [8, 40, 41, 54] is to make the 2D laser range finder rotate, such that the whole scene is scanned.

For what concerns 3D depth sensing, a breakthrough technology has been, in 2010, the Microsoft Kinect. Initially commercialized for the Xbox game console, the complementary nature of the depth and visual information provided by the Kinect opened up new opportunities to solve fundamental problems in both

[1]Only recently, with the automotive industry moving towards driver-less cars, 3D scanners that cost less than 10,000$ started becoming available.

robotics and computer vision [22]. In fact, for less that 150$, one can have a device composed by a standard color camera and a structured-light depth sensor, that provides images of 640×480 pixels containing depth values of the framed scenes. Unfortunately, Kinect depth measurements are much more noisy than those provided by more expensive depth sensors such as laser range finders. Moreover they do not work outdoor.

In robotics, Kinect and similar devices are used, for example, in SLAM and 3D reconstruction systems [26, 39, 64] as well as in people tracking applications [20, 32, 36]. Have a look at [22] for a comprehensive review of Kinect-based computer vision algorithms and applications.

The common objective of the works presented in this thesis is to allow robots to have an RGB-D view of the surrounding environment, i.e. a 3D view containing, for each point, also the color information. To this end, three calibration problems involving depth sensors are addressed. They all exploit one or more cameras to correctly recover the desired parameters. The first problem is about recovering of the axis of rotation of an actuated 2D laser range finder. The second problem, instead, is the correction of the bias on the depth measurements of Kinect-like depth sensors. Finally, the RGB-depth registration problem is addressed: a distributed algorithm for estimating the poses of all the sensors in a network of cameras and depth sensors is described.

1.1 Thesis Outline and Contributions

In Chapter 2 we give a first insight to the notation used throughout the thesis. We also provide a brief introduction on the geometric concepts needed to understand the work, and a quick description of the technologies behind cameras and some depth sensors.

In Chapter 3, a novel calibration algorithm to recover the axis of rotation of an actuated 2D laser range finder is described. The algorithm uses kinematics point-plane constraints to have an initial estimate of the relative pose between the sensor and a camera. Then an initial estimate for the rotational axis is obtained performing a screw decomposition of the results of the first step. Finally, the obtained values are refined within a non-linear optimization procedure.

In Chapter 4 we detail a new algorithm to correct the depth data provided by Kinect-like depth sensors. First of all, an analysis of the error is made. Unlike existing approaches, the depth error is separated into two components: local and global. Then a novel automatic procedure to estimate a map able to recover the local shape of the cloud is presented: it exploits the planarity of a wall framed during the data acquisition step. Finally, a method to correct the global error exploiting a camera is described. In this case, the rigid transform between the two sensors and the global correction map parameters, after an initial estimation, are optimized together within a non-linear least squares optimization framework. This allows us to calibrate the depth sensor against an external camera, operation impossible in most of the similar works. Furthermore, a detailed analysis of the calibration results on a test set is reported.

In Chapter 5 a new distributed package for calibrating networks composed by both cameras and depth sensors is described. The calibration is performed in real-time, that is, the pose of a sensor is estimated as soon as enough data are available and refined with a non-linear optimization. To this end, the data analysis procedure is distributed among the PCs of the network in order to reduce both the bandwidth usage and the computational load on the main PC. An example of how to configure and use this calibration package is reported in Appendix A.

Finally, in Chapter 6 some conclusions are drawn.

1.2 Publications

The work described in this thesis has also been presented in the publications listed here below.

- [50] E. So, F. Basso, and E. Menegatti. “Calibration of a Rotating 2D Laser Range Finder using Point-Plane Constraints”. In: *Journal of Automation, Mobile Robotics & Intelligent Systems* 7.2 (2013), pp. 30–38

- [7] F. Basso, A. Pretto, and E. Menegatti. “Unsupervised Intrinsic and Extrinsic Calibration of a Camera-Depth Sensor Couple”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* Hong Kong, China, 2014, pp. 6244–6249. DOI: 10.1109/ICRA.2014.6907780

- [4] F. Basso, R. Levorato, and E. Menegatti. “Online Calibration for Networks of Cameras and Depth Sensors”. In: *Proceedings of the 12th Workshop on Non-classical Cameras, Camera Networks and Omnidirectional Vision (OMNIVIS).* Hong Kong, China, 2014

- [5] F. Basso, R. Levorato, M. Munaro, and E. Menegatti. “A Distributed Calibration Algorithm for Color and Range Camera Networks”. In: ed. by A. Koubaa. Studies in Systems, Decision and Control. (To appear). Springer

- [6] F. Basso, M. Munaro, S. Michieletto, and E. Menegatti. “Fast and Robust Multi-People Tracking from RGB-D Data for a Mobile Robot”. In: *Proceedings of the 12th Intelligent Autonomous Systems (IAS) Conference.*

Vol. 193. Jeju Island, Korea, 2012, pp. 265–276. DOI: 10.1007/978-3-642-33926-4_25

- [36] M. Munaro, F. Basso, and E. Menegatti. "Tracking People within Groups with RGB-D Data". In: *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*. Vilamoura, Portugal, 2012, pp. 2101–2107. DOI: 10.1109/IROS.2012.6385772
- [37] M. Munaro, F. Basso, S. Michieletto, E. Pagello, and E. Menegatti. "A Software Architecture for RGB-D People Tracking Based on ROS Framework for a Mobile Robot". In: *Frontiers of Intelligent Autonomous Systems*. Ed. by S. Lee, K.-J. Yoon, and J. Lee. Vol. 466. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2013, pp. 53–68. DOI: 10.1007/978-3-642-35485-4_5

Chapter 2

Preliminaries

The aim of this chapter is twofold: brush up on the part of geometry that is mandatory to master in order to understand the thesis, and introduce the sensors mostly used in our works. To this end, in Section 2.1 some fundamental notions of 3D geometry are reported. In Section 2.2, perspective cameras and the pinhole camera model are described, while in Section 2.3 some depth sensor technologies are briefly introduced. Finally, since an established way to represent the aforementioned concepts does not exist in literature, in Section 2.4 the notation used throughout the thesis is well defined.

2.1 3D Geometry

The notion of *rigid transformation* in 3D space is fundamental to clearly comprehend the rest of the thesis. To help the reader to brush it up, key notions such as the various representations for rotations in 3D space (Section 2.1.1), 3D transformation matrices (Section 2.1.2) and the concept of *reference frame* and how it relates to 3D transformations (Section 2.1.3) are reported.

2.1.1 Rotations

By definition, a rotation about the origin is an isometry that preserves the origin and orientation. Rotations in three dimensions are generally not commutative, so the order in which rotations are applied is important, even about the same point. Also, rotations about the origin have three degrees of freedom (DOF). A 3D rotation can be specified in a number of ways. The most usual methods are:

- rotation matrices;
- axis-angle representation;

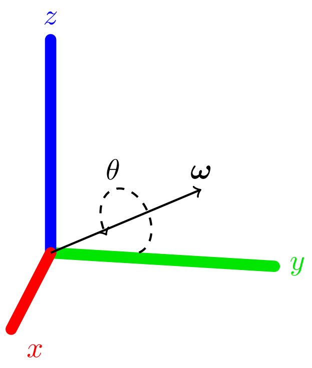


Figure 2.1: Axis-angle representation of a rotation. Here $\boldsymbol{\omega}$ is the axis of rotation and θ is the magnitude of the rotation about the axis.

- unit quaternions.

Rotation Matrix

Rotations are linear transformations of $\mathbb{R}^3$ and can therefore be represented by *matrices* once a basis of $\mathbb{R}^3$ has been chosen. They can be characterized as *orthogonal* matrices with determinant 1: a square matrix $\mathbf{R}$ is a rotation matrix if $\mathbf{R}^\mathsf{T} = \mathbf{R}^{-1}$ and $\det(\mathbf{R}) = 1$. The set of all orthogonal matrices of size 3 with determinant 1 forms a group known as the special orthogonal group SO(3).
Given a rotation matrix

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \in \mathrm{SO}(3) \quad ,$$

to rotate a point $\mathbf{x} = (x, y, z)^\mathsf{T} \in \mathbb{R}^3$ to a point $\mathbf{y} = (x', y', z')^\mathsf{T} \in \mathbb{R}^3$ it suffices to left multiply $\mathbf{x}$ by the rotation matrix. That is,

$$\mathbf{y} = \mathbf{R} \cdot \mathbf{x} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad .$$

Axis-Angle Representation

In mathematics, the *axis-angle* representation of a rotation parameterizes a rotation in a three-dimensional space by two values: a unit vector $\boldsymbol{\omega}$ indicating the direction of the axis of rotation, and an angle θ describing the magnitude of the rotation about the axis. The rotation occurs in the sense prescribed by the right-hand rule. The *Rodrigues' rotation formula* is an efficient algorithm for rotating a 3D point, given the rotation axis $\boldsymbol{\omega}$ and the angle θ of rotation. That is, let $\mathbf{x} = (x, y, z)^\mathsf{T} \in \mathbb{R}^3$ be a point, the Rodrigues' rotation formula to obtain the rotated vector $\mathbf{y}$ is

$$\mathbf{y} = \cos\theta \cdot \mathbf{x} + \sin\theta \cdot (\mathbf{x} \times \boldsymbol{\omega}) + (1 - \cos\theta) \cdot (\boldsymbol{\omega}^\mathsf{T}\mathbf{x}) \cdot \boldsymbol{\omega} \ ,$$

where $\times$ denotes the cross product.
Given a vector $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^\mathsf{T}$, the matrix for a rotation by an angle of θ about $\boldsymbol{\omega}$ is

$$\mathbf{R}_\omega(\theta) = \cos\theta \cdot \mathbf{I}_3 + \sin\theta \cdot [\boldsymbol{\omega}]_\times + (1 - \cos\theta) \cdot \boldsymbol{\omega}\boldsymbol{\omega}^\mathsf{T} \ ,$$

where

$$[\boldsymbol{\omega}]_\times = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} \ .$$

Quaternions

Quaternions give a simple way to encode the axis-angle representation in four numbers, and can be used to apply the corresponding rotation to a position vector, representing a point relative to the origin in $\mathbb{R}^3$. Recalling that a vector such as $\mathbf{x} = (x, y, z)^\mathsf{T}$ can be rewritten as $x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ where $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ are unit vectors representing the three Cartesian axes, a rotation through an angle of θ around the axis defined by a unit vector $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^\mathsf{T}$ can be represented using an extension of Euler's formula:

$$\mathbf{q} = e^{\frac{\theta}{2}(\omega_x\mathbf{i} + \omega_y\mathbf{j} + \omega_z\mathbf{k})} = \cos\frac{\theta}{2} + (\omega_x\mathbf{i} + \omega_y\mathbf{j} + \omega_z\mathbf{k})\sin\frac{\theta}{2} \ .$$

It can be shown that the rotation defined by a quaternion $\mathbf{q}$ can be applied to an ordinary vector $\mathbf{x} = (x, y, z)^\mathsf{T} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ (considered as a quaternion with a real coordinate equal to zero) by evaluating the conjugation of $\mathbf{x}$ by $\mathbf{q}$, that is

$$\mathbf{y} = \mathbf{q}\mathbf{x}\mathbf{q}^{-1} \ ,$$

where

$$\mathbf{q}^{-1} = e^{-\frac{\theta}{2}(\omega_x\mathbf{i} + \omega_y\mathbf{j} + \omega_z\mathbf{k})} = \cos\frac{\theta}{2} - (\omega_x\mathbf{i} + \omega_y\mathbf{j} + \omega_z\mathbf{k})\sin\frac{\theta}{2} \ .$$

Given a unit quaternion $\mathbf{q} = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$, the equivalent rotation matrix is

$$\mathbf{R}(\mathbf{q}) = \begin{pmatrix} 1 - 2q_y^2 - 2q_z^2 & 2(q_xq_y - q_zq_w) & 2(q_xq_z + q_yq_w) \\ 2(q_xq_y + q_zq_w) & 1 - 2q_x^2 - 2q_z^2 & 2(q_yq_z - q_xq_w) \\ 2(q_xq_z - q_yq_w) & 2(q_yq_z + q_xq_w) & 1 - 2q_x^2 - 2q_y^2 \end{pmatrix} \ .$$

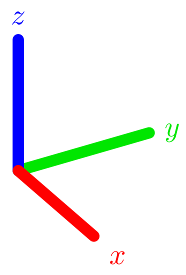


Figure 2.2: Typical colors for a 3D reference frame.

2.1.2 Transformations

Let $\mathbf{x} = (x, y, z)^\mathsf{T}$ be a point in 3D space. For any non-zero real number w, $(xw, yw, zw, w)^\mathsf{T}$ is the set of homogeneous coordinates associated to the point $\mathbf{x}$. In particular, when $w = 1$, the resulting 4D vector $\tilde{\mathbf{x}} = (x, y, z, 1)^\mathsf{T}$ is called the *normalized homogeneous form* of $\mathbf{x}$.

A rigid transformation in 3D space is represented by a linear transformation $\mathbf{T} \in \mathrm{SE}(3)$ on normalized homogeneous vectors

$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\mathsf{T} & 1 \end{pmatrix}$$

where $\mathbf{R} \in \mathrm{SO}(3)$ is the *rotation matrix* and $\mathbf{t} \in \mathbb{R}^3$ the *translation* vector.

To transform a 3D point $\mathbf{x}$ it is sufficient to left-multiply its normalized homogeneous form $\tilde{\mathbf{x}}$ by the transformation matrix $\mathbf{T}$

$$\tilde{\mathbf{y}} = \mathbf{T} \cdot \tilde{\mathbf{x}} \ . \tag{2.1}$$

The resulting vector $\tilde{\mathbf{y}}$ is the normalized homogeneous form of the desired 3D point $\mathbf{y}$. This operation is equivalent to perform an affine transformation in $\mathbb{R}^3$

$$\mathbf{y} = \mathbf{R} \cdot \mathbf{x} + \mathbf{t} \ .$$

2.1.3 Reference Frames

A *reference frame* $\mathcal{F}$ is a coordinate system used to represent and measure position and orientation of objects. A transformation matrix ${}^{\mathcal{T}}_{\mathcal{S}}\mathbf{T}$ from a source reference frame $\mathcal{S}$ to a target frame $\mathcal{T}$ is a transformation matrix that allows to convert the coordinates of a point from $\mathcal{S}$ to $\mathcal{T}$.

Let, for example, $\mathbf{x}$ be a point and let ${}^{\mathcal{S}}\mathbf{x}$ be its coordinates in $\mathcal{S}$, $\mathbf{x}$ coordinates in $\mathcal{T}$, namely ${}^{\mathcal{T}}\mathbf{x}$, can be computed as

$${}^{\mathcal{T}}\tilde{\mathbf{x}} = {}^{\mathcal{T}}_{\mathcal{S}}\mathbf{T} \cdot {}^{\mathcal{S}}\tilde{\mathbf{x}} \ .$$

(a) Basler Pilot

(b) PointGrey Flea 3

Figure 2.3: Two examples of commercially available perspective cameras.

When drawn, the x-axis of a reference frame is typically red, the y-axis is green while the z-axis is blue, as shown in Figure 2.2.

2.2 Perspective Cameras

A camera (some professional cameras are shown in Figure 2.3) is a device in which the 3D scene is projected down onto a 2D image. That is, let $\mathbb{I}_{\mathfrak{C}} = \{0, 1, \ldots, H_{\mathfrak{C}} - 1\} \times \{0, 1, \ldots, W_{\mathfrak{C}} - 1\}$ be a two dimensional index for images of size $W_{\mathfrak{C}} \times H_{\mathfrak{C}}$. Then, a camera $\mathfrak{C}$ provides a discrete representation of a scene by means of an image $\{\mathsf{I}_i\}_{i \in \mathbb{I}_{\mathfrak{C}}}$, that maps each pixel $(u, v) \in \mathbb{I}_{\mathfrak{C}}$ to a certain color space $\mathbb{W}$, i.e. $\mathsf{I} : \mathbb{I}_{\mathfrak{C}} \to \mathbb{W}$.

For what concerns the projection onto the image of the 3D space, the most commonly used projection in computer vision is 3D perspective projection. Section 2.2.1 describes perspective projection based on the pinhole camera model.

2.2.1 Pinhole Camera Model

Let $\mathfrak{C}$ be a camera and $\mathcal{C}$ its reference frame. The pinhole model (see Figure 2.4) describes the mathematical relationship between the coordinates of a 3D point and its projection onto the image plane. The model assumes that the camera has 4 parameters [9], namely *intrinsic parameters*:

- $(c_x, c_y)^\mathsf{T}$ is the *principal point* that is usually at the image center;
- f_x, f_y are the *focal lengths* expressed in pixel units;

usually arranged in a 3×3 matrix $\mathbf{K}_{\mathfrak{C}}$

$$\mathbf{K}_{\mathfrak{C}} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} .$$

The relationship between the coordinates of a 3D point ${}^{\mathcal{C}}\mathbf{x} = (x, y, z)^\mathsf{T}$ and its projection onto the image plane ${}^{\mathcal{I}}\mathbf{x} = (u, v)^\mathsf{T}$ in pixels is

$$ {}^{\mathcal{I}}\widetilde{\mathbf{x}} = \mathbf{K}_{\mathfrak{C}} \cdot \frac{{}^{\mathcal{C}}\mathbf{x}}{z} \ , \tag{2.2} $$

at least theoretically. Unfortunately, real lenses usually have some distortion. Therefore, the pinhole model is extended with a vector of *distortion coefficients*

$$ \mathbf{d}_{\mathfrak{C}} = (k_1, k_2, p_1, p_2, k_3, k_4, k_5, k_6) \ , $$

and a distortion function $\mathrm{d}_{\mathfrak{C}}(\cdot)$. Equation (2.2) thus becomes

$$ {}^{\mathcal{I}}\widetilde{\mathbf{x}} = \mathbf{K}_{\mathfrak{C}} \cdot \mathrm{d}_{\mathfrak{C}}\left(\frac{{}^{\mathcal{C}}\mathbf{x}}{z}\right) \ , \tag{2.3} $$

where

$$ \mathrm{d}_{\mathfrak{C}}\left(\frac{{}^{\mathcal{C}}\mathbf{x}}{z}\right) = \mathrm{d}_{\mathfrak{C}}\left(\left(\frac{x}{z}, \frac{y}{z}, 1\right)^\mathsf{T}\right) = \mathrm{d}_{\mathfrak{C}}\left((\widehat{x}, \widehat{y}, 1)^\mathsf{T}\right) = (x_\mathrm{d}, y_\mathrm{d}, 1)^\mathsf{T} $$

and

$$ \begin{cases} \begin{pmatrix} x_\mathrm{d} \\ y_\mathrm{d} \end{pmatrix} & = \dfrac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} \begin{pmatrix} \widehat{x} \\ \widehat{y} \end{pmatrix} + 2\widehat{x}\widehat{y} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + 2 \begin{pmatrix} \widehat{x} p_2 \\ \widehat{y} p_1 \end{pmatrix} + r^2 \begin{pmatrix} p_2 \\ p_1 \end{pmatrix} \ , \\ r^2 & = \widehat{x}^2 + \widehat{y}^2 \ . \end{cases} $$

In the following, given a camera $\mathfrak{C}$, the *reprojection* of a 3D point onto $\mathfrak{C}$'s image plane, i.e. (2.3), will be denoted by the function $\mathrm{r}_{\mathfrak{C}}(\cdot)$, that is

$$ {}^{\mathcal{I}}\mathbf{x} = \mathrm{r}_{\mathfrak{C}}\left({}^{\mathcal{C}}\mathbf{x}\right) \ . \tag{2.4} $$

Checkerboard Pose Estimation

Let $\mathfrak{B}$ be an $R \times C$ checkerboard and let $\mathcal{B}$ be its reference frame. Let also $\mathfrak{C}$ be a camera with reference frame $\mathcal{C}$, intrinsic parameters $\mathbf{K}_{\mathfrak{C}}$ and distortion coefficients $\mathbf{d}_{\mathfrak{C}}$. We can estimate the checkerboard pose ${}^{\mathcal{C}}_{\mathcal{B}}\mathbf{T}$ in the camera reference frame by finding its corners in the image and solving the correspondent *Perspective-n-Point* (PnP) problem [16].

That is, let $\mathbb{I}_{\mathfrak{B}} = \{1, \dots, R\} \times \{1, \dots, C\}$ be the corner indices, let also

$$ {}^{\mathcal{B}}\mathsf{B} = \left\{ {}^{\mathcal{B}}\mathbf{b}_{r,c} \right\}_{(r,c) \in \mathbb{I}_{\mathfrak{B}}} $$

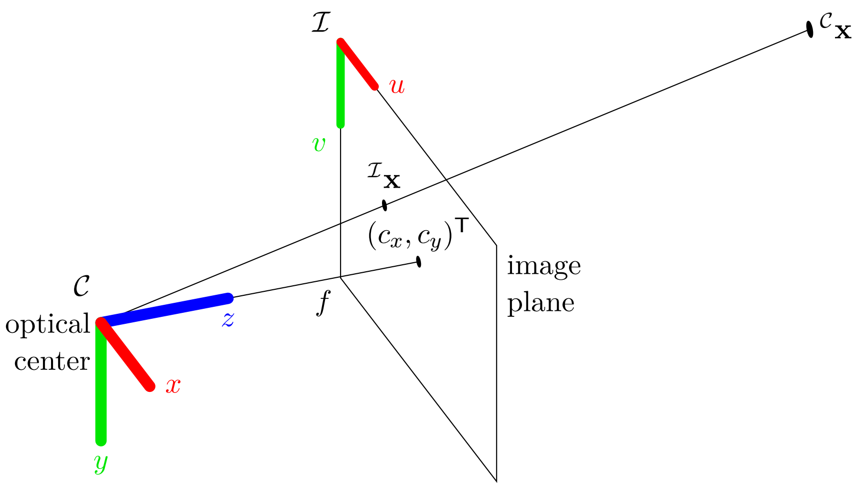


Figure 2.4: Perspective projection based on the pinhole camera model.

be the checkerboard corners and

$$ {}^{\mathcal{I}}\mathsf{B} = \left\{ {}^{\mathcal{I}}\mathbf{b}_{r,c} \right\}_{(r,c)\in\mathbb{I}_{\mathfrak{B}}} $$

the correspondent locations in the image. The checkerboard pose ${}^{\mathcal{C}}_{\mathcal{B}}\mathbf{T}$ can be estimated by means of a function called *solvePnP* [9]

$$ {}^{\mathcal{C}}_{\mathcal{B}}\mathbf{T} = \mathrm{solvePnP}\left(\mathbf{K}_{\mathfrak{C}}, \mathbf{d}_{\mathfrak{C}}, {}^{\mathcal{B}}\mathsf{B}, {}^{\mathcal{I}}\mathsf{B}\right) \quad . \tag{2.5} $$

It is the one that minimizes the *reprojection error*

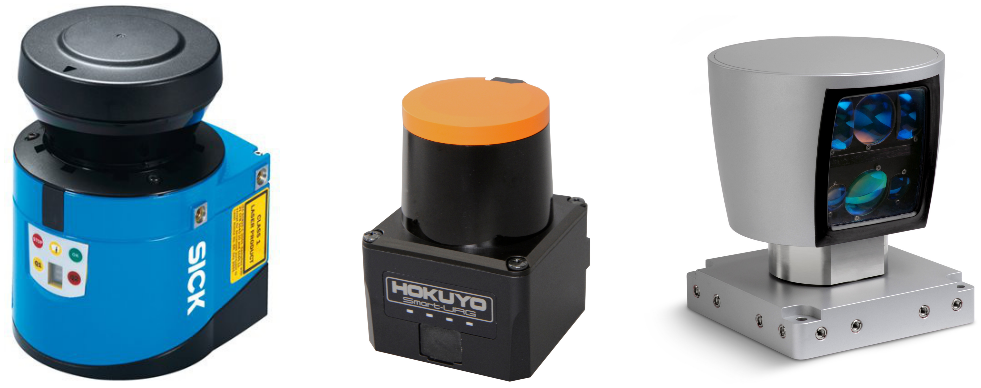
$$ \mathrm{e}_{\mathrm{r}_{\mathfrak{C}}}\left({}^{\mathcal{C}}_{\mathcal{B}}\mathbf{T}, {}^{\mathcal{B}}\mathsf{B}, {}^{\mathcal{I}}\mathsf{B}\right) = \sum_{(r,c)\in\mathbb{I}_{\mathfrak{B}}} \left\| {}^{\mathcal{I}}\mathbf{b}_{r,c} - \mathrm{r}_{\mathfrak{C}}\left({}^{\mathcal{C}}_{\mathcal{B}}\mathbf{T} \cdot {}^{\mathcal{B}}\mathbf{b}_{r,c}\right) \right\|^2 \quad . \tag{2.6} $$

2.3 Depth Sensors

Depth sensors are active devices able to estimate the depth of obstacles in their field of view. Many technologies have been developed in years. Among them, laser scanners (Section 2.3.1) and time-of-flight cameras (Section 2.3.2) are based on the known speed of light. On the contrary, structured-light depth sensors such as the Microsoft Kinect (Section 2.3.3) estimate distances using triangulation.

2.3.1 Laser Scanners

While for detecting intensity/color information of a scene a passive sensor like a camera is sufficient, when 3D information about the scene is needed active sensors



(a) Sick LMS100 (b) Hokuyo UST-10LX (c) Velodyne HDL-64E

Figure 2.5: Some examples of commercially available 2D (a-b) and 3D (c) laser scanners.

are mandatory. A first type of active sensors to retrieve depth information is the *laser scanner* (sometimes called *LiDAR*).

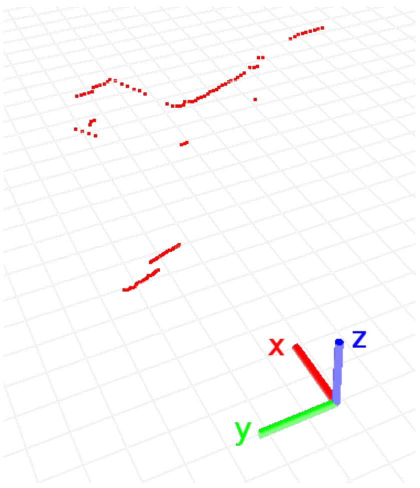
A time-of-flight laser scanner is an active scanner that uses laser light to probe the scene. At the heart of this type of scanner is a time-of-flight laser range finder. The laser range finder estimates the distance of a surface by timing the round-trip time (rtt) of a pulse of laser light. Since the speed of light c is known, the rtt determines the travel distance of the light. That is, let t be the rtt, then distance d is equal to

$$d = \frac{c \cdot t}{2} \ .$$

The laser range finder only detects the distance of one point in its direction of view. Thus, to probe the distance in its entire field of view, the scanner scans one point at a time by changing its direction of view with a system of rotating mirrors.

Range scanners can be divided into two categories: 2D and 3D scanners (Figure 2.5); examples of the data produced by such sensors is shown in Figure 2.6. 2D scanners are simpler and less expensive than 3D scanners, and are typically used in robotics applications like SLAM. 2D scanners can also be used to simulate a 3D one by making them rotate with, for example, a pan-tilt unit. With respect to a real 3D scanner, however, an actuated 2D scanner is slower and less precise, while being tremendously cheaper.

If we know the 3D location of a scanner center (typically the point to which range measurements are referred), we can estimate the 3D location of a scanner measurement by simple trigonometry. Supposing to deal with a 2D scanner, given the angle θ of a light beam and the returned range d, the 3D location of the point



(a) 2D scan

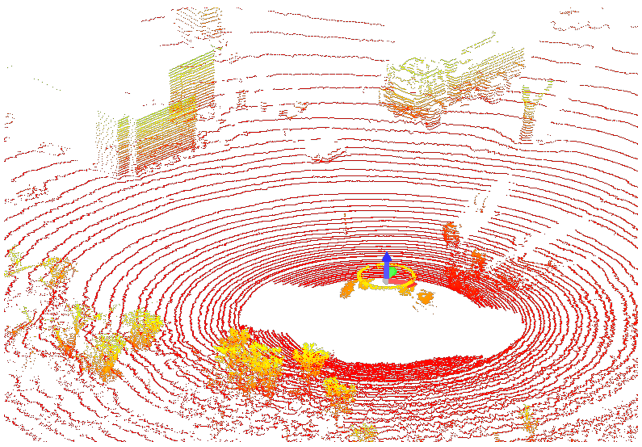

(b) 3D scan

Figure 2.6: Example of 2D (a) and 3D (b) laser scanner output.

$\mathbf{x}$, with respect to the laser center $\mathcal{L}$, is

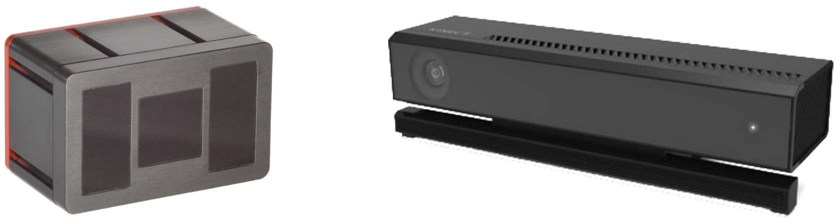
$$^{\mathcal{L}}\mathbf{x} = (d \cdot \cos\theta, d \cdot \sin\theta, 0)^{\mathsf{T}} \quad .$$

2.3.2 Time-of-Flight Cameras

Other examples of depth sensors based on the time-of-flight principle are *time-of-flight* (ToF) cameras. In contrast to laser scanning systems, in a time-of-flight camera no mechanical moving parts are needed, nevertheless they are able to measure the distances within a complete scene with a single shot, providing a depth value for each pixel of the image. The entire scene can be captured with each light pulse, as opposed to point-by-point with a laser beam such as in scanning LiDAR systems [25].

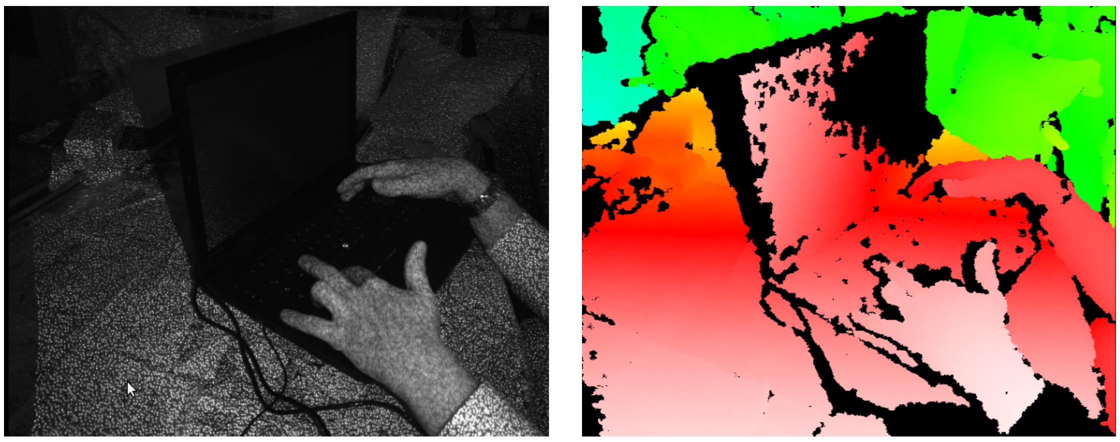
A ToF camera can be thought as a matricial organization of a multitude of single devices, each one made by an emitter and a co-positioned receiver. In practice, this configuration is not possible, thus some emitters are positioned on the sensor in order to mimick the presence of a single emitter co-positioned with the center of a matrix of receivers, which are implemented as CCD/CMOS lock-in pixels [12].

With respect to standard cameras, ToF cameras usually provide lower resolution images, e.g. 176×144 pixels for the Mesa SR-4500 (Figure 2.7(a)) and 512×424 pixels for the Kinect v2 (Figure 2.7(b)). Note that the Kinect v2 device also contains a perspective RGB camera.



(a) Mesa SR-4500 (b) Microsoft Kinect v2

Figure 2.7: Some examples of commercially available ToF cameras.



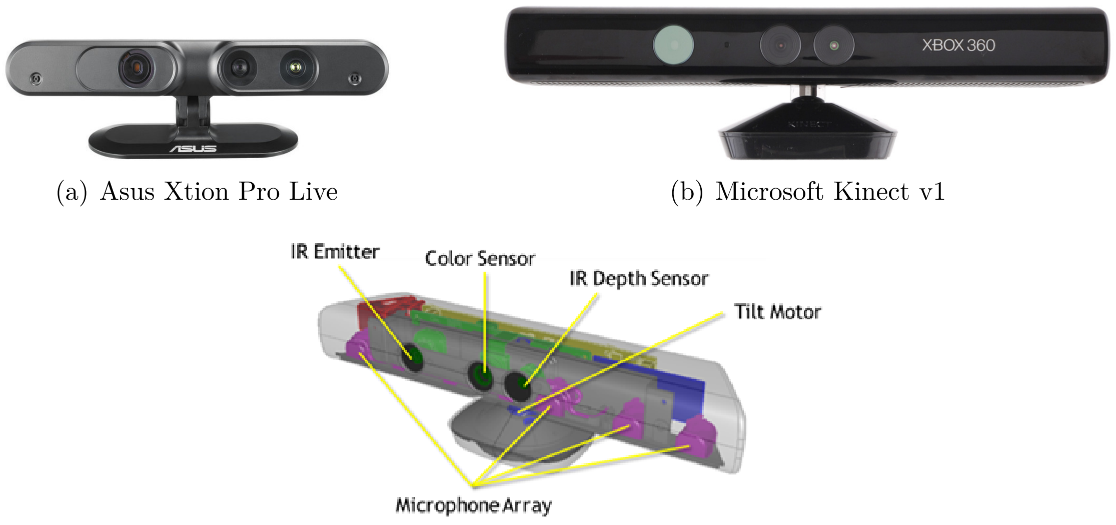
(a) Infrared image acquired by the infrared camera. The projected pattern is clearly visible.

(b) Depth image provided by the sensor in false colors, from white (closer) to blue (farther).

Figure 2.8: Example of a depth map computed by a Microsoft Kinect (b) starting from an image of the projected pattern (a).

2.3.3 Structured-Light Depth Sensors

The last class of range sensors we describe is not based on the time-of-flight principle but on triangulation. A structured-light depth sensor is a 3D scanning device for measuring the three-dimensional shape of an object using a projected light pattern and a camera [19]. A defined light pattern is projected in the infrared portion of the spectrum and the camera is used to analyze it. The way that the pattern deforms when striking surfaces allows vision systems to calculate the depth and surface information of the objects in the scene. An example of the pattern projected by a Microsoft Kinect is visible in Figure 2.8. The device compares the pattern of acquired image (Figure 2.8(a)) with the one expected, and estimates a depth value for all the visible points (Figure 2.8(b)).



(a) Asus Xtion Pro Live (b) Microsoft Kinect v1

(c) Internal view of a Kinect

Figure 2.9: RGB-D sensors based on depth technology developed by PrimeSense.

PrimeSense Devices: Microsoft Kinect and Asus Xtion Pro Live

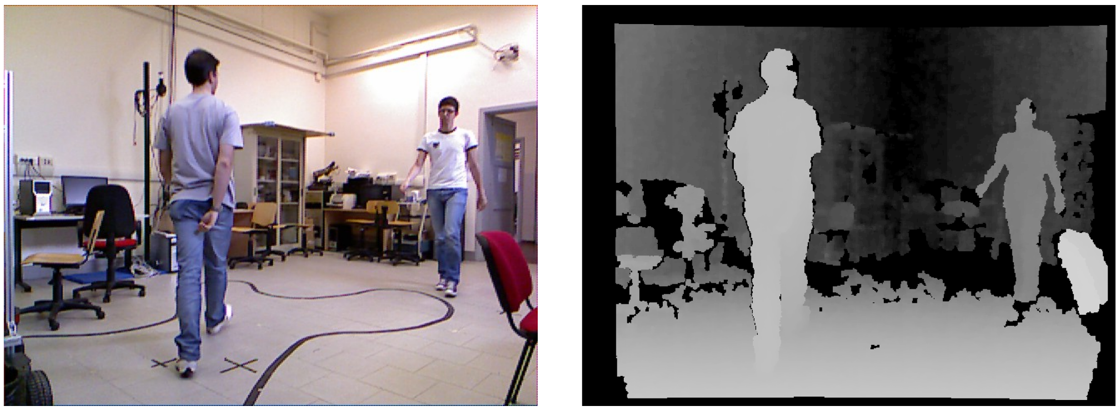
In 2010, Microsoft released a device, called Kinect, in an attempt to broaden Xbox 360's audience beyond its typical gamer base. This device was build on range camera technology developed by PrimeSense. Later, Asus teamed up with PrimeSense and release a very similar device, called Xtion Pro Live, to let developers create motion-sensing PC applications. The two devices (shown in Figure 2.9) and are commonly called RGB-D sensors.

They are composed by:

- an RGB camera with VGA resolution (640×480 pixels) that provides images at a frequency of 30 Hz;
- a structured-light depth sensor (infrared projector + camera) that provides depth images at the same resolution and frequency;
- an array of 4 microphones.

A view of the internal components of a Kinect is shown in Figure 2.9(c). For what concerns the RGB and depth images, in Figure 2.10 the output of the sensors is displayed.

The main drawback of these depth sensors is that they do not work if there are other sources of infrared light in the environment: the projected pattern becomes impossible to see, thus the depth cannot be estimated. For this reason, it is almost



(a) RGB image (b) Depth image

Figure 2.10: Examples of RGB and depth images provided by a Kinect. The depth image has been transformed according to some internal parameters to well register to the RGB image.

impossible to use Kinect depth sensors outdoor. Also, when using two or more of Kinects together, care must be taken to avoid the projected patterns to overlap.

Anyway, thanks to their low price (when compared to other depth sensor types) and thanks to the quantity of data they are able to provide (RGB + depth), they have become some of the most widely used devices for mobile robotics applications.

2.3.4 Depth Data

Similarly to a color camera, a structured-light depth sensor (and even a ToF camera) $\mathfrak{D}$ provides a discrete representation of the scene by means of an image $\{\mathsf{J}_i\}_{i \in \mathbb{I}_\mathfrak{D}}$ of size $W_\mathfrak{D} \times H_\mathfrak{D}$, that maps a pixel (u, v) to its depth value $d \in \mathbb{R}$, i.e. $\mathsf{J} : \mathbb{I}_\mathfrak{D} \to \mathbb{R}$.

While the data provided by a depth sensor is typically an image, the intrinsic parameters of the pinhole camera that composes the depth sensor let us estimate, for each pixel (u, v), the corresponding 3D point ${}^{\mathcal{D}}\mathbf{x} = (x, y, z)^\mathsf{T}$. According to the pinhole camera model (see Section 2.2.1), ${}^{\mathcal{D}}\mathbf{x}$ can be computed as

$$
{}^{\mathcal{D}}\mathbf{x} = \mathbf{K}_\mathfrak{D}^{-1} \cdot d \cdot {}^{\mathcal{J}}\tilde{\mathbf{x}} \ ,
$$

where $\mathbf{K}_\mathfrak{D}$ is the intrinsic parameters matrix of the depth sensor's camera.

Therefore, we can express the data provided by the depth sensor as a *point cloud* $\{\mathsf{C}_i\}_{i \in \mathbb{I}_\mathfrak{D}}$, $\mathsf{C} : \mathbb{I}_\mathfrak{D} \to \mathbb{R}^3$ that maps each pixel $(u, v)^\mathsf{T}$ to the corresponding 3D point ${}^{\mathcal{D}}\mathbf{x} = (x, y, z)^{\mathsf{T}}$. In the rest of the thesis we will typically use point clouds to represent depth data.

2.4 Notation

We use non-bold characters x to represent scalars, bold lower case letters $\mathbf{x}$ to represent vectors, and bold upper case letters $\mathbf{M}$ to represent matrices. Then, we use sans-serif, upper case letters to denote indexed sets S, that is $\mathsf{S} = \{\mathfrak{s}_i\}_{i \in \mathbb{I}}$ for some index $\mathbb{I} \subset \mathbb{N}^k$, $k > 0$, and $\mathsf{S}(i) = \mathfrak{s}_i$. Examples of indexed sets are images, where pixel coordinates are used to reach the color of the pixels, and checkerboards, where the corner locations are indexed by the pair row-column.

The reference frame of a body $\mathfrak{B}$ is in calligraphic style $\mathcal{B}$. The coordinates of an entity e with respect to the reference frame $\mathcal{F}$ are denoted by ${}^{\mathcal{F}}e$. According to this notation, the pose of a body $\mathfrak{B}$ in $\mathfrak{F}$'s coordinate system $\mathcal{F}$ is denoted as ${}^{\mathcal{F}}\mathcal{B}$ and the relative homogeneous transformation matrix is ${}^{\mathcal{F}}_{\mathcal{B}}\mathbf{T}$.

Finally, to improve readability, in the following:

- the homogeneous notation of Section 2.1.2 will be dropped, considering $\mathbf{y} = \mathbf{T} \cdot \mathbf{x}$ equivalent to $\tilde{\mathbf{y}} = \mathbf{T} \cdot \tilde{\mathbf{x}}$.

- with an abuse of notation, coordinate system transformations of *any entity* e will be expressed as a left multiplication by the desired transformation matrix. That is,
$$ {}^{\mathcal{T}}e = {}^{\mathcal{T}}_{\mathcal{S}}\mathbf{T} \cdot {}^{\mathcal{S}}e \ . $$

Chapter 3

Calibration of a Rotating 2D Laser Range Finder

Laser range finders (LRF) are used extensively in mobile robots for many tasks, including localization, mapping, and obstacle avoidance. 2D LRFs, such as the SICK LMS-100 and the Hokuyo UTM-30LX, have proved to be popular due to their low costs, although the range measurement data are confined to a single plane. 3D LRFs such as the Velodyne HDL-64 are commercially available, but they are not as widely employed due to their high costs. Affordable 3D range sensors such as time-of-flight/phase-shift cameras (e.g. Swiss Ranger SR-4500) and structured-light cameras (e.g. Microsoft Kinect) have recently became available, but they have limited range and are usually confined to indoor use.

An economical method for obtaining 3D range data from a 2D LRF is to rotate the LRF, by using a stepper motor or a pan-tilt device. To combine the LRF range data obtained at different angles into a single coordinate frame, the axis of rotation relative to the mirror center inside the laser ranger finder is required. As shown in Figure 3.2, this rotational axis is a line in 3D space with 4 degrees of freedom.

In this paper, we propose a method to recover the parameters of this axis by scanning several large planar checkerboard patterns with the LRF and imaging the checkerboards with a static camera. In particular, we use only correspondences between lines in the laser scans and planes in the camera images, which can be established easily even for non-visible lasers.

Essentially, the proposed calibration procedure consists of three main steps. First, static camera-LRF calibration is performed at two different rotational angles. Although there already exists numerous solutions to this problem in the literature, we propose instead to model the calibration problem using kinematics point-plane constraints. This allows us to use the minimum number of calibration planes, reducing the amount of time for data acquisition and processing.

In the second step, we obtain an initial estimate for the rotational axis by performing a screw decomposition of the relative LRF motion estimated in the first step.

In the final step, starting with an initial estimate of the parameters of the rotational axis, a nonlinear minimization is performed to recover its precise values. For the cost function, instead of the perpendicular point-to-plane distance that is commonly used, we propose to minimize the “line-of-sight” errors, which directly models the noise in the LRF range measurements.

3.1 Related Work

Several methods have been proposed to calibrate a static 2D LRF with a static camera [31, 34, 62, 69]. The most commonly-used solution is the one given by Zhang [69], which uses correspondences between lines in a laser scan and planes in a camera image, as we do in this paper. In [62] and [31], correspondences between a point in the laser scan and a line in a camera image are used. For calibration between a static 3D LRF and a static camera, Unnikrishnan [58] gives a solution similar to Zhang’s that uses correspondences between planes in a laser scan and planes in a camera image. As we will explain in Section 3.4, all of these methods can be cast as point-plane constraints problems. All of these work aim to find the transformation between the coordinate frames of a LRF and a camera.

On the other hand, our current work is focused on locating the axis of rotation of an articulated 2D LRF. The solution used by one of the earliest system is to have the supporting bracket precisely machined so that the axis of rotation is aligned with the mirror center of the LRF [54, 55]. If this is not the case, mechanical drawings and hand measurement are used to manually locate the offset distance [2]. For the rotating 2D LRF described in [45], a more elaborate method described by Weingarten [63] is used, which involves scanning a cuboid room measured by hand. In all of these methods, the axis of rotation is assumed to be perfectly aligned with one of the primary axis of the LRF, and only the offset distance is determined. In this paper, by modeling the axis of rotation with 4 degrees of freedom, we allow for rotational misalignment to occur.

In [41], Pradeep et al. describes a general calibration method for sensors on an articulated kinematic chain. This has been implemented as a package for the Robot Operating System (ROS) [42] for the PR-2 robot. Their work is similar to ours, but it relies on establishing correspondences between point features in the laser scans and the camera image. To do so, they use the reflectance values of the laser scan points to locate the checkerboard corners in the laser scans. However, reflectance values can be greatly affected by scanning geometry and ambient

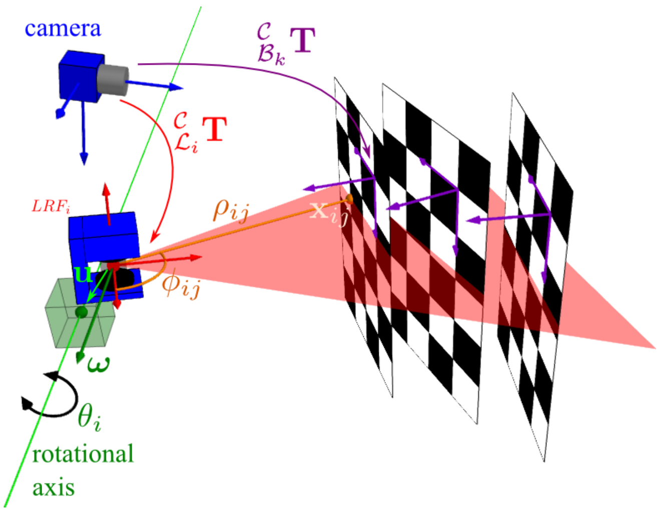


Figure 3.1: Coordinate frames and transforms used in calibration.

lighting, making checkerboard detection difficult. Furthermore, the checkerboard corners are likely to lie in between laser scan points, and attempting to account for such localization errors in the laser scans will make the minimization relatively complicated. Instead, by using only the range values in the laser scans, the noise in the calibration data can be easily modeled by minimizing the "line-of-sight" errors, as described in Section 3.6.

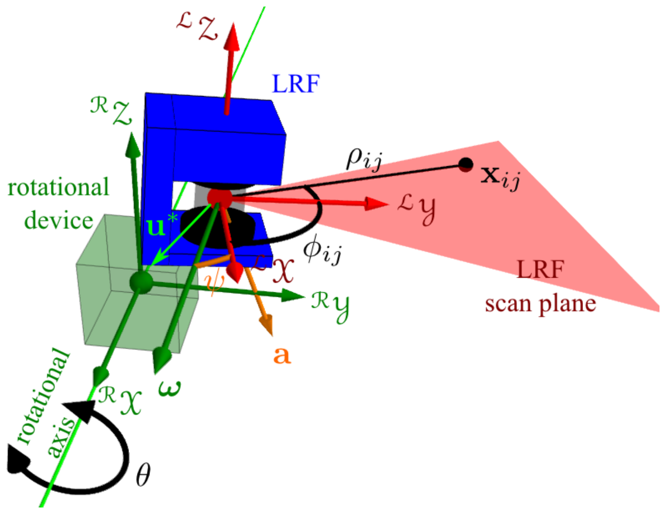
3.2 Geometric Model

3.2.1 Coordinate Frames

We attach a coordinate frame $\mathcal{L}$ to the mirror center of the LRF (Figure 3.1). The LRF rotates around a rotational axis (green line in Figure 3.1) that is parallel to $\boldsymbol{\omega}$ and passes through a point u with respect to the origin of $\mathcal{L}$. We denote the rotation angle of the LRF around the rotational axis by θ, and index it by i. The LRF coordinate frame after a rotation to θ_i is denoted by $\mathcal{L}_i$.

The laser beam of the LRF sweeps out the ${}^{\mathcal{L}}\mathcal{X}$-${}^{\mathcal{L}}\mathcal{Y}$ plane. We denote the angle of the laser beam from the ${}^{\mathcal{L}}\mathcal{X}$-axis by ϕ and indexed it by j. When the LRF is at rotation angle θ_i , a range measurement ρ_{ij} at beam azimuth angle ϕ_{ij} corresponds to a point with coordinates ${}^{\mathcal{L}_i}\mathbf{x}_{ij} = (\rho_{ij}\cos\phi_{ij}, \rho_{ij}\sin\phi_{ij}, 0)^{\mathsf{T}}$ with respect to $\mathcal{L}_i$.

We attach a coordinate frame $\mathcal{C}$ to the camera center, which does not move during calibration. We index the checkerboards used in the calibration by k, and denote them as $\mathfrak{B}_k$, attaching a coordinate frame $\mathcal{B}$ at each of their origin. The plane of the checkerboards are denoted as π_k, with normal $\mathbf{n}_k$ and distance d_k



${}^{\mathcal{L}}\mathcal{Z}$
LRF
${}^{\mathcal{R}}\mathcal{Z}$
rotational device
ρ_{ij}
$\mathbf{x}_{ij}$
${}^{\mathcal{L}}\mathcal{Y}$
$\mathbf{u}^*$
ϕ_{ij}
${}^{\mathcal{L}}\mathcal{X}$
ψ
${}^{\mathcal{R}}\mathcal{Y}$
LRF scan plane
$\mathbf{a}$
$\boldsymbol{\omega}$
${}^{\mathcal{R}}\mathcal{X}$
rotational axis
θ

Figure 3.2: Coordinate frames of the rotational device and the LRF. The axis of rotation of a rotating LRF relative to its mirror center has 4 degrees of freedom: 2 for the offset distance $\mathbf{u}$âĹŮ $= (0, u_y^*, u_z^*)^\mathsf{T}$ in the $\mathcal{X}$-$\mathcal{Y}$ plane of the LRF, and 2 for the unit vector $\boldsymbol{\omega}$ in the direction of the rotational axis.

with respect to the camera coordinate frame. Note that the plane equation is: $\mathbf{n}_k^\mathsf{T}\mathbf{x} - d_k = 0$.

3.2.2 Rotational Transformation of Coordinate Frames

To combine the LRF range data obtained at different rotational angles θ_i, all of the points x_{ij} should be converted to a common coordinate frame. A simple choice is to use the first LRF coordinate frame $\mathcal{L}_0$. After a rotation of $\Delta\theta = \theta_i - \theta_0$, the transformation of the LRF coordinate frame is

$$ {}^{\mathcal{L}_0}_{\mathcal{L}_i}\mathbf{T} = \begin{pmatrix} \mathbf{R}_{\boldsymbol{\omega}}(\Delta\theta) & (\mathbf{I}_3 - \mathbf{R}_{\boldsymbol{\omega}}(\Delta\theta))\,\mathbf{u} \\ \mathbf{0}^\mathsf{T} & 1 \end{pmatrix} . \tag{3.1} $$

Alternatively, we can attach a coordinate frame $\mathcal{R}$ on the rotational axis. If the ${}^{\mathcal{R}}\mathcal{X}$-axis is aligned with the rotational axis, two degrees of freedom remain for the transformation from $\mathcal{L}_0$ to $\mathcal{R}$ – a translation along the rotational axis, and a rotation about the rotational axis. In order to fix $\mathcal{R}$ unambiguously, we place its origin to lie in the ${}^{\mathcal{L}}\mathcal{Y}$-${}^{\mathcal{L}}\mathcal{Z}$ plane, and its orientation is chosen as the one having the minimal rotation from $\mathcal{L}_0$, as shown in Figure 3.2.

Thus, given the direction $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^\mathsf{T}$ and a point $\mathbf{u} = (u_x, u_y, u_z)^\mathsf{T}$ on the rotational axis with respect to $\mathcal{L}_0$, the transformation from $\mathcal{R}$ to $\mathcal{L}_0$ is

$$ {}^{\mathcal{R}}_{\mathcal{L}_0}\mathbf{T} = \begin{pmatrix} \mathbf{R}_{\mathbf{a}}(\phi)^\mathsf{T} & -\mathbf{R}_{\mathbf{a}}(\phi)^\mathsf{T}\mathbf{u}^* \\ \mathbf{0}^\mathsf{T} & 1 \end{pmatrix} , \tag{3.2} $$

where

$$\begin{aligned}
\mathbf{u}^* &= \mathbf{u} - \frac{u_x}{\omega_x} \cdot \boldsymbol{\omega} = \left(0, u_y - \frac{u_x}{\omega_x} \cdot \omega_y, u_z - \frac{u_x}{\omega_x} \cdot \omega_z\right)^\mathsf{T} \\
\mathbf{a} &= \frac{\mathbf{i} \times \boldsymbol{\omega}}{||\mathbf{i} \times \boldsymbol{\omega}||} = \frac{1}{\sqrt{\omega_y^2 + \omega_z^2}} \left(0, -\omega_z, \omega_y\right)^\mathsf{T} \\
\phi &= \arccos(\mathbf{i}^\mathsf{T}\boldsymbol{\omega}) = \arccos(\omega_x) \ ,
\end{aligned}$$

and, consequently,

$$\mathbf{R}_\mathbf{a}(\phi) = \begin{pmatrix} \omega_x & -\omega_y & -\omega_z \\ \omega_y & \omega_x + \omega_z^2 \cdot \frac{1-\omega_x}{\omega_y^2+\omega_z^2} & -\omega_y\omega_z \cdot \frac{1-\omega_x}{\omega_y^2+\omega_z^2} \\ \omega_z & -\omega_y\omega_z \cdot \frac{1-\omega_x}{\omega_y^2+\omega_z^2} & \omega_x + \omega_y^2 \cdot \frac{1-\omega_x}{\omega_y^2+\omega_z^2} \end{pmatrix} \ .$$

After a rotation of $\Delta\theta = \theta_i - \theta_0$, the transformation from $\mathcal{R}$ to $\mathcal{L}_i$ is thus

$$^{\mathcal{R}}_{\mathcal{L}_i}\mathbf{T} = {}^{\mathcal{R}}_{\mathcal{L}_0}\mathbf{T} \cdot {}^{\mathcal{L}_0}_{\mathcal{L}_i}\mathbf{T} = \begin{pmatrix} \mathbf{R}_\mathbf{a}(\phi)^\mathsf{T}\mathbf{R}_\boldsymbol{\omega}(\Delta\theta) & -\mathbf{R}_a(\phi)^\mathsf{T}\mathbf{u}^* \\ \mathbf{0}^\mathsf{T} & 1 \end{pmatrix} \ . \tag{3.3}$$

3.2.3 Representation of the Rotational Axis

In (3.1), the rotational axis is represented by a point $\mathbf{u}$ and a direction $\boldsymbol{\omega}$, giving a total of 5 DOF. However, since $\mathbf{u}$ is a vector from the origin of $\mathcal{L}$ to any point on the rotational axis, another point $\mathbf{u}' = \mathbf{u} + \lambda\boldsymbol{\omega}$ for any $\lambda \in \mathbb{R}$ can also be used. In (3.2) and (3.3), $\mathbf{u}$ is replaced by $\mathbf{u}^* = (0, u_y^*, u_z^*)^\mathsf{T}$, a point constrained to ${}^{\mathcal{L}}\mathcal{Y}$-${}^{\mathcal{L}}\mathcal{Z}$ plane, leaving it with only 2 DOF.

Another way to represent the rotational axis is to make use of the vector product $\mathbf{v} = \boldsymbol{\omega} \times \mathbf{u}$, known as the moment of the line, which is invariant under the choice of $\mathbf{u}$, since $\mathbf{v}' = \boldsymbol{\omega} \times (\mathbf{u} + \lambda\boldsymbol{\omega}) = \boldsymbol{\omega} \times \mathbf{u} = \mathbf{v}$. The pair of vectors $(\boldsymbol{\omega}, \mathbf{v})$ are known as the Plücker coordinates of the line [47]. The two vectors must satisfy the constraint $\boldsymbol{\omega}^\mathsf{T}\mathbf{v} = 0$ and are defined up to scale, so $(\lambda\boldsymbol{\omega}, \lambda\mathbf{v})$ for any $\lambda \neq 0$ defines the same line; thus, there are 4 DOF within the six parameters of $\boldsymbol{\omega}$ and $\mathbf{v}$.

Given the Plücker coordinates $(\boldsymbol{\omega}, \mathbf{v})$ of a line, we can go back to the point and direction representation by finding a point $\mathbf{u}$ on the line, which can be obtained simply as [47]

$$\mathbf{u} = \frac{\mathbf{v} \times \boldsymbol{\omega}}{||\boldsymbol{\omega}||^2} \ .$$

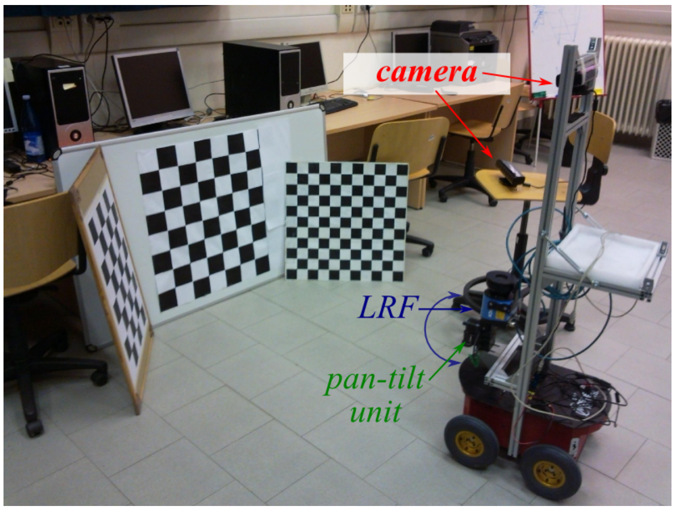


Figure 3.3: Calibration setup: 3 or more planar calibration patterns are scanned by the rotating LRF and imaged by a static camera.

3.3 Overview of Calibration Procedure

3.3.1 Data Acquisition

In our proposed calibration procedure, the data acquisition consists of placing $K \geq 3$ planar checkerboard patterns $\mathfrak{B}_k$ in front of the rotating LRF to be calibrated, as shown in Figure 3.3. A single image of all the checkerboards is captured to determine their poses ${}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{T}$ relative to the camera coordinate frame $\mathcal{C}$. The LRF is rotated to $N \geq 2$ different angles θ_i, at each of which a scan $\mathbf{S}_i = \{\mathbf{x}_{ij}\}$ of the checkerboards are made. The set of points in each scan $\mathbf{S}_i$ belonging to checkerboard $\mathfrak{B}_k$ is denoted by $\mathbf{S}_{i,k} = \{\mathbf{x}_{ij} : \mathbf{x}_{ij} \in \mathfrak{B}_k\}$.

Instead of using K different checkerboards and scanning them all at once, one single checkerboard can be scanned K times at different poses, giving an equivalent set of scan points $\{\mathbf{S}_{i,k}\}$.

3.3.2 Data Processing

The objective of the calibration is to estimate the parameters of the rotational axis, $(\boldsymbol{\omega}, \mathbf{v})$. In order to do so, we propose a calibration procedure with the three steps outlined below.

1. Initial static camera-LRF calibration: using scan $\mathbf{S}_0$, determine the pose ${}^{\mathcal{C}}_{\mathcal{L}_0}\mathbf{T}$ of the LRF at the initial rotation angle θ_0 with respect to the camera.

2. Initial estimate of the rotational axis $(\boldsymbol{\omega}, \mathbf{v})$.

   (a) Estimate the final LRF pose ${}^{\mathcal{C}}_{\mathcal{L}_N}\mathbf{T}$: sing scan $\mathbf{S}_N$, determine the pose of the LRF at the final rotation angle θ_N.

   (b) Extract $(\boldsymbol{\omega}, \mathbf{v})$ from ${}^{\mathcal{L}_0}_{\mathcal{L}_N}\mathbf{T}$ using the screw decomposition.

3. Non-linear refinement of ${}^{\mathcal{C}}_{\mathcal{L}_0}\mathbf{T}$ and $(\boldsymbol{\omega}, \mathbf{v})$. Given ${}^{\mathcal{C}}_{\mathcal{L}_0}\mathbf{T}$ and $(\boldsymbol{\omega}, \mathbf{v})$, the expected scan data $\{\mathbf{S}^*_{i,k}\}$ can be calculated. The task then is to find the optimal ${}^{\mathcal{C}}_{\mathcal{L}_0}\mathbf{T}^*$ and $(\boldsymbol{\omega}^*, \mathbf{v}^*)$ which minimizes the differences between $\{\mathbf{S}_{i,k}\}$ and $\{\mathbf{S}^*_{i,k}\}$.

Step 1 is equivalent to performing a calibration between a camera and a static 2D-LRF. This can be accomplished using any of the existing methods mentioned in Section 3.1. However, as we will explain in Section 3.4, static camera-LRF calibration can be modeled as a kinematics point-plane constraints problem. Doing so allows us to use the minimum number of calibration boards, reducing the amount time for data acquisition and processing.

Step 2 is used to obtain an initial estimate for the parameters of the rotational axis $(\boldsymbol{\omega}, \mathbf{v})$. First, the static calibration of step 1 is repeated, but with the LRF at the final rotational angle θ_N. The rotational axis can then be extracted using the screw decomposition, as we will explain in Section 3.5. An initial estimate for $(\boldsymbol{\omega}, \mathbf{v})$ can also be obtained through other means, such as using manual measurement or mechanical drawings. In such cases, step 2 can be skipped.

In step 3, starting with initial value for ${}^{\mathcal{C}}_{\mathcal{L}_0}\mathbf{T}$ and $(\boldsymbol{\omega}, \mathbf{v})$, we use non-linear optimization to find the optimal estimates ${}^{\mathcal{C}}_{\mathcal{L}_0}\mathbf{T}^*$ and $(\boldsymbol{\omega}^*, \mathbf{v}^*)$ which minimizes the errors between the actual scan data $\{\mathbf{S}_{i,k}\}$ and the expected scan values $\{\mathbf{S}^*_{i,k}\}$, which we will describe in detail in Section 3.6.

3.4 Static Camera-LRF Calibration using Point-Plane Constraints

3.4.1 Kinematics Point-Plane Constraints

In the point-plane constraints problem, n points $\mathbf{x}_i$ on a rigid body $\mathfrak{B}$, with reference frame $\mathcal{B}$, are constrained to lie on n planes π_i whose parameters are known with respect to a ground coordinate frame $\mathcal{G}$ (Figure 3.4). The goal is to recover all possible poses ${}^{\mathcal{G}}_{\mathcal{B}}\mathbf{T}$ of $\mathcal{B}$ with respect to $\mathcal{G}$. This is a problem studied in kinematics. In [48], Selig shows, using algebraic geometry, that with a minimum of 6 point plane constraints, up to 8 solutions can be obtained. In [59, 60], Wampler shows equivalent results, along with numerical methods for obtaining the solutions.

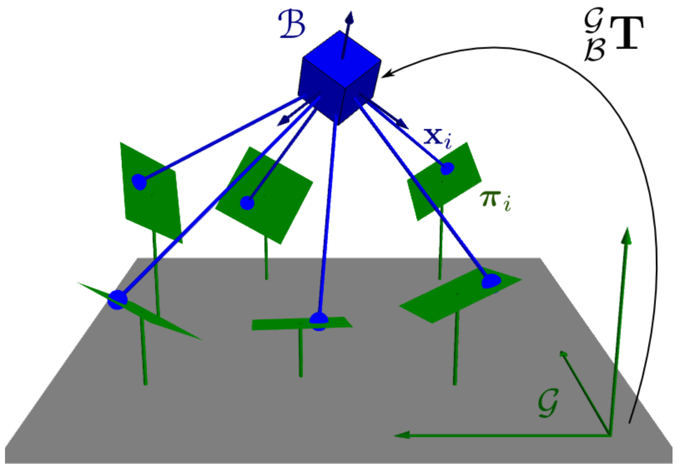


Figure 3.4: The kinematics point-plane constraints problem.

3.4.2 Modelling Static Camera-LRF Calibration

Treating the LRF as the body $\mathfrak{B}$ and the camera coordinate frame as the ground coordinate frame $\mathcal{G}$, it is straightforward to see that the static camera-LRF calibration problem can be modeled as a point-plane constraints problem. The LRF is able to measure points $\mathbf{x}_{ij}$ with respect to its own coordinate frame $\mathcal{L}$. These points are known to lie on a checkerboard plane π_k imaged by the camera. Thus, the plane parameters $(\mathbf{n}_k, d_k)$ are known with respect to the camera coordinate frame $\mathfrak{C}$. Each checkerboard pattern scanned by the LRF and imaged by the camera gives one line-on-plane constraint, which is equivalent to two independent point-on-plane constraints, as any other point on the same line will give a redundant point-on-plane constraint. With 3 checkerboards, we obtain the minimum number of n = 6 to solve the point-plane constraints problem.

3.4.3 Numerical Solution

For a point $\mathbf{x}_{ij}$ in the body coordinate frame $\mathcal{B}$, it lies on a plane π_k with normal $\mathbf{n}_k$ at a distance d_k from the origin in $\mathcal{G}$ if

$$\mathbf{n}_k^\mathsf{T}(\mathbf{R}\mathbf{x}_{ij} + \mathbf{t}) - d_k = 0 \ . \tag{3.4}$$

Here, $\mathbf{R} \in \mathrm{SO}(3)$ and $\mathbf{t} \in \mathbb{R}^3$ are the rotation and translation of the transformation ${}^{\mathcal{G}}_{\mathcal{B}}\mathbf{T}$ from $\mathcal{G}$ to $\mathcal{B}$.

Using quaternion coefficients $\mathbf{q} = (q_w, q_x, q_y, q_z)^\mathsf{T}$ to represent a rotation as

$\mathbf{R} = \frac{1}{\mathbf{q}^\mathsf{T}\mathbf{q}}\mathbf{R}(\mathbf{q})$, (3.4) can be rewritten as

$$F_{ijk}(\mathbf{q},\mathbf{t}) = \mathbf{n}_k^\mathsf{T}\mathbf{R}(\mathbf{q})\mathbf{x}_{ij} - \mathbf{q}^\mathsf{T}\mathbf{q}\cdot d_k + \mathbf{q}^\mathsf{T}\mathbf{q}\cdot\mathbf{n}_k^\mathsf{T}\mathbf{t} = \mathbf{q}^\mathsf{T}\mathbf{M}_{ijk}\cdot\mathbf{q} + \mathbf{q}^\mathsf{T}\mathbf{q}\cdot\mathbf{n}_k^\mathsf{T}\mathbf{t} = 0 \ . \quad (3.5)$$

Here, the quaternion $\mathbf{q}$ is a homogeneous vector and not necessarily a unit vector as is commonly employed. This was done to avoid the need to impose a quadratic constraint. As a result, the scalar product $q^\mathsf{T}q$ appears in the denominator to normalize the quaternion. $\mathbf{M}_{ijk}$ is a 4×4 symmetric matrix containing the elements of $\mathbf{x}_{ij}$, $\mathbf{n}_k$, and d_k.

Given n point-plane constraints, a system of n equations in the form of (3.5) is obtained, which is quadratic in $\mathbf{q}$ and linear in $\mathbf{t}$. The linear terms $\mathbf{q}^\mathsf{T}\mathbf{q}\cdot\mathbf{t}$ can be eliminated through Gaussian elimination, leaving $n-3$ equations in $\mathbf{q}$. For the minimum case of $n=6$, we obtain a system of 3 quadratic polynomials in the 3 free variables of $\mathbf{q}$, which gives up to 8 solutions. This can be solved using Gröbner bases [11]. In [59, 60], Wampler gives a eigenvalue-based numerical solution, which he calls "numerical Gröbner bases".

For $n > 6$, we can find a unique solution by solving subsets of the n constraints and find the common solution among them. However, if there is noise in the data, there will be no solution that can simultaneously satisfy all the constraint equations. Instead, Wampler gives linear solutions for the separate cases of $n=7$, $8 \le n < 12$, and $n \ge 12$. In the following, we give a brief review of the solution for $8 \le n < 12$. The reader is referred to Wampler's papers [59, 60] for further details.

Equation (3.5) can be rewritten in a linear form in terms of the 2nd degree monomials of $\mathbf{q}$, that is

$$F_{ijk}(\mathbf{q},\mathbf{t}) = \mathbf{C}_{ijk}\cdot\tilde{\mathbf{q}} + \mathbf{q}^\mathsf{T}\mathbf{q}\cdot\mathbf{n}_k^\mathsf{T}\mathbf{t} \ . \quad (3.6)$$

Here, $\mathbf{C}_{ijk}$ is a 1×10 vector with rearranged entries of $\mathbf{M}_{ijk}$, and $\tilde{\mathbf{q}}$ is a 10×1 vector with the 2nd degree monomials of q in lexicographical ordering:

$$\tilde{\mathbf{q}} = (q_w^2, q_wq_x, q_wq_y, q_wq_z, q_x^2, q_xq_y, q_xq_z, q_y^2, q_yq_z, q_z^2)^\mathsf{T} \ .$$

With n points on n planes, we can stack the constraints of (3.6) to get a system of equations

$$F(\mathbf{q},\mathbf{t}) = \mathbf{C}\cdot\tilde{\mathbf{q}} + \mathbf{q}^\mathsf{T}\mathbf{q}\cdot\mathbf{N}\mathbf{t} \ . \quad (3.7)$$

Here, $\mathbf{C}$ is a $n\times10$ matrix, and $\mathbf{N}$ is a $n\times3$ matrix containing the normal vectors $\mathbf{n}_k^\mathsf{T}$. To solve this system, we need to have $\text{rank}(\mathbf{N})=3$, meaning that the normal vectors of the planes should span 3-space.

Pre-multiplying (3.7) with $\mathbf{N}^\perp$, a $(n-3)\times n$ matrix spanning the left null space of $\mathbf{N}$, we reduce the system to a set of $n-3$ homogeneous quadratic polynomials

Table 3.1: Minimum LRF-scan and image pairs required by the different method compared.

Method	Correspondences	Scan-Image Pairs	
		Original	Point-Plane *
Zhang [69]	line-on-plane	5	3 (4)
Wasielewski [62]	point-on-image-line	9	6 (7)
Li [31]	point-on-image-line	5	3 (4)

* number in brackets indicate pairs required for unique solution.

in $\mathbf{q}$, i.e.

$$F(\mathbf{q}) = \mathbf{N}^{\perp}\mathbf{C} \cdot \tilde{\mathbf{q}} \ . \tag{3.8}$$

This can be minimized using SVD, with the solution $\tilde{\mathbf{q}}$ being the right singular vector corresponding to the smallest singular value. The rotation $\mathbf{R}(\mathbf{q})$ is extracted from $\mathbf{q}$ and substituted back in (3.7) to recover the translation $\mathbf{t}$.

3.4.4 Minimal Conditions

In the camera-LRF calibration method given by Zhang [69], a linear solution is given to obtain an initial estimate for the camera to LRF transformation ${}^{\mathcal{C}}_{\mathcal{L}}\mathbf{T}$. Each line-on-plane correspondence gives two independent equations in nine unknown parameters, so a minimum of 5 LRF scan and image pairs are needed.

In Wasielewski's calibration method [62] using LRF point on image line correspondences, each set of correspondence gives one constraint equation in nine unknown parameters, so a minimum of 9 LRF scan and image pairs are needed. In Li's method [31] which uses a calibration target giving two image lines, a minimum of 5 LRF scan and image pairs are needed.

As mentioned above, a line-on-plane correspondence can be modeled as two point-plane constraints. As we describe in [51], a point-on-image-line correspondence can also be modeled as a point-plane constraint, where the plane passes through the camera center and the image line. Table 3.1 shows a summary of the minimum number of LRF-scan and image pairs required by the different methods, as well as the minimum conditions when modeled using point-plane constraints.

Although usually it is preferable to use more laser scans and image pairs to provide redundant data to reduce the effects of noise, leading to more accurate calibration results. However, we use the fact that we already have a lot of redundant data from the scans $\{\mathbf{S}_i\}$ at different rotational angles θ_i. Thus, we are motivated to find the minimal solution for the initial static camera-LRF calibration step, and instead rely on using redundant data over all the rotational angles θ_i in the final minimization described in Section 3.6 to reduce the effects of noise in the calibration.

Although using point-plane constraints to model the different type of correspondences reduces the minimal scan-image pairs required only very slightly, the advantage of using point-plane constraints is that all types of correspondences can be modeled uniformly and solved using a single method.

3.5 Initial Estimate of the Rotational Axis using Screw Decomposition

Using the static camera-LRF calibration method described in Section 3.4, we obtain an initial estimate for ${}^{\mathcal{C}}_{\mathcal{L}_0}\mathbf{T}$ and ${}^{\mathcal{C}}_{\mathcal{L}_N}\mathbf{T}$ for step 1 and step 2(a) respectively. Composing the camera to LRF transforms, we obtain the relative LRF motion between the two rotational angles θ_0 and θ_N

$$ {}^{\mathcal{L}_0}_{\mathcal{L}_N}\mathbf{T} = {}^{\mathcal{C}}_{\mathcal{L}_0}\mathbf{T}^{-1} \cdot {}^{\mathcal{C}}_{\mathcal{L}_N}\mathbf{T} \ . $$

According to Chasles's theorem, every spatial transformation can be decomposed as a screw displacement – a rotation about a line together with a translation along the same line [47], that is

$$ {}^{\mathcal{L}_0}_{\mathcal{L}_N}\mathbf{T} = \begin{pmatrix} \mathbf{R}_\omega(\Delta\theta) & {}^{\mathcal{L}_0}_{\mathcal{L}_N}\mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \tag{3.9} $$

$$ = \begin{pmatrix} \mathbf{R}_\omega(\Delta\theta) & \frac{\Delta\theta}{2\pi} \cdot p\boldsymbol{\omega} + (\mathbf{I}_3 - \mathbf{R}_\omega(\Delta\theta))\mathbf{u} \\ \mathbf{0}^T & 1 \end{pmatrix} \ . \tag{3.10} $$

Equation (3.10) is known as the "screw matrix" [33, 47], representing a rotation of angle $\Delta\theta$ around a line called the screw axis that has direction $\boldsymbol{\omega}$ and passes through the point $\mathbf{u}$. p is the pitch of the screw motion, corresponding to the translation along $\boldsymbol{\omega}$ for every revolution around the screw axis. Compared with (3.9), while the rotation matrix remains the same, the translation has been decomposed into a translation along the rotational axis $\frac{\Delta\theta}{2\pi} \cdot p\boldsymbol{\omega}$, and a translation perpendicular to the rotational axis $(\mathbf{I}_3 - \mathbf{R}_\omega(\Delta\theta))\mathbf{u}$.

Given $\mathbf{R}_\omega(\Delta\theta)$ and ${}^{\mathcal{L}_0}_{\mathcal{L}_N}\mathbf{t}$ of the relative LRF motion from θ_0 and θ_N, we can recover the parameters of its equivalent screw displacement as

$$ p = \frac{2\pi}{\Delta\theta} \cdot \boldsymbol{\omega}^{\mathsf{T}}\, {}^{\mathcal{L}_0}_{\mathcal{L}_N}\mathbf{t} \tag{3.11} $$

$$ \mathbf{u} = (\mathbf{I}_3 - \mathbf{R}_\omega(\Delta\theta))^{\dagger} \left({}^{\mathcal{L}_0}_{\mathcal{L}_N}\mathbf{t} - \frac{\Delta\theta}{2\pi} \cdot p\boldsymbol{\omega} \right) \ . \tag{3.12} $$

Since $(\mathbf{I}_3 - \mathbf{R}_\omega(\Delta\theta))\boldsymbol{\omega} = 0$, $\mathbf{I}_3 - \mathbf{R}_\omega(\Delta\theta)$ is a singular matrix; thus, its pseudo-inverse is used in (3.12) to recover $\mathbf{u}$.

Ideally, the decomposition of the relative motion ${}^{\mathcal{L}_0}_{\mathcal{L}_N}\mathbf{T}$ of the LRF from angle θ_0 to θ_N will give a screw displacement with zero pitch $p = 0$, and the screw axis $(\boldsymbol{\omega}, \mathbf{v})$ is the rotational axis of the LRF. Furthermore, if we repeat the static camera-LRF calibration at another rotation angle θ_i and compute ${}^{\mathcal{L}_0}_{\mathcal{L}_N}\mathbf{T}$, the screw decomposition should give the same screw axis $(\boldsymbol{\omega}, \mathbf{v})$. However, with noise in the calibration data, such ideal results will not be obtained. The task then remains to finding the optimal rotational axis $(\boldsymbol{\omega}^*, \mathbf{v}^*)$ that minimizes some error function which accounts for the noise in the calibration data.

3.6 Non-linear Refinement

The linear solution to the point-plane constraints problem given in Section 3.4.3 minimizes an algebraic error. To find the optimal calibration parameters, an optimization that minimizes a meaningful geometric error should be performed. The error function most commonly used in static camera-LRF calibration is the point-to-plane distance $e_{\text{pp}}(\cdot)$, that is

$$\arg\min_{{}^{\mathcal{C}}_{\mathcal{L}}\mathbf{T}} \sum_{j,k} e^2_{\text{pp}}({}^{\mathcal{L}}\mathbf{x}_j, {}^{\mathcal{C}}\pi_k) = \sum_{j,k} \left\| {}^{\mathcal{C}}\mathbf{n}_k^\top \left({}^{\mathcal{C}}_{\mathcal{L}}\mathbf{R} \cdot {}^{\mathcal{L}}\mathbf{x}_j + {}^{\mathcal{C}}_{\mathcal{L}}\mathbf{t} \right) - d_k \right\|^2$$

As shown in Figure 3.5, this is the perpendicular distance from the plane π_k to the point $\mathbf{x}_j$. However, if we assume that the points $\mathbf{x}_j = \text{f}(\phi_j, \rho_j)$ measured by the LRF, contains only zero-mean Gaussian noise $w_{\mathfrak{L}} \sim \mathcal{N}(0, \sigma^2_{\mathfrak{L}})$ in its range values ρ_j, so $\rho_j = \rho^*_j + w_{\mathfrak{L}}$, then instead of the point-to-plane errors $e_{\text{pp}}(\cdot)$, a better error to minimize is the "line-of-sight" distance $e_{\text{los}}(\cdot)$, as illustrated in Figure 3.5

$$\arg\min_{{}^{\mathcal{C}}_{\mathcal{L}}\mathbf{T}} \sum_{j,k} e^2_{\text{los}}({}^{\mathcal{L}}\mathbf{x}_j, {}^{\mathcal{C}}\pi_k) = \sum_{j,k} \left\| {}^{\mathcal{C}}\mathbf{x}^*_j - {}^{\mathcal{C}}\mathbf{x}_j \right\|^2 = \sum_{j,k} \left\| \rho^*_j - \rho_j \right\|^2 \quad . \tag{3.13}$$

In (3.13), given plane $\pi_k : (\mathbf{n}_k, d_k)$ for the k^{th} plane, the ideal coordinates $\mathbf{x}^*_j$ is calculated as the intersection point of the plane π_k with the line passing through the LRF mirror center and the measured point $\mathbf{x}_j$. Using $\hat{\mathbf{x}}_j$ to denote $\frac{\mathbf{x}_j}{||\mathbf{x}_j||}$, we obtain

$${}^{\mathcal{L}}\mathbf{x}^*_j = \frac{d_k - {}^{\mathcal{C}}_{\mathcal{L}}\mathbf{t}^\top \cdot {}^{\mathcal{C}}\mathbf{n}_k}{{}^{\mathcal{L}}\hat{\mathbf{x}}_j^\top \cdot {}^{\mathcal{C}}_{\mathcal{L}}\mathbf{R}^\top \cdot {}^{\mathcal{C}}\mathbf{n}_k} \cdot {}^{\mathcal{L}}\hat{\mathbf{x}}_j \quad .$$

Ignoring errors in π_k, (3.13) gives a maximum likelihood estimate for $\mathbf{R}$, $\mathbf{t}$. To account for errors in the pose of the checkerboard planes π_k estimated by the camera, assuming zero-mean Gaussian noise $w_{\mathfrak{C}} \sim \mathcal{N}(0, \sigma^2_{\mathfrak{C}})$ in the image coordinates

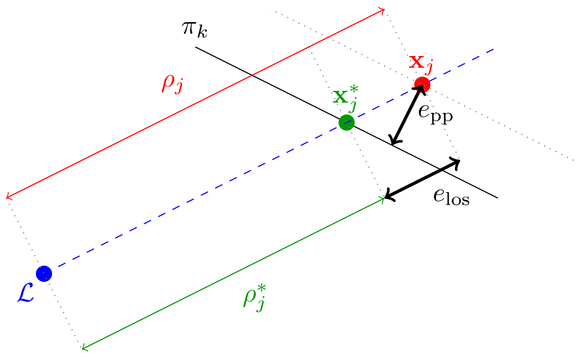


Figure 3.5: Point-to-plane error e_{pp} vs. line-of-sight error e_{los}.

${}^{\mathcal{I}}\mathbf{b}_{r,c}$ of the checkerboard corners with 3D coordinates ${}^{\mathcal{B}}\mathbf{b}_{r,c}$, the minimization can be augmented with the image reprojection errors of the checkerboard corners, as in the bundle adjustment approach [9], so

$$\arg\min_{{}^{\mathcal{C}}_{\mathcal{L}}\mathbf{T},{}^{\mathcal{C}}_{\mathcal{B}}\mathbb{T}} \frac{1}{\sigma^2_{\mathfrak{L}}}\sum_{j,k}\left\|{}^{\mathcal{L}}\mathbf{x}_j^* - {}^{\mathcal{L}}\mathbf{x}_j\right\|^2 + \frac{1}{\sigma^2_{\mathfrak{C}}}\sum_{r,c,k}\left\|{}^{\mathcal{I}}\mathbf{b}_{r,c} - \mathrm{r}_{\mathfrak{C}}\left({}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{R}\,{}^{\mathcal{B}}\mathbf{b}_{r,c} + {}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{t}\right)\right\|^2 .$$

Here, ${}^{\mathcal{C}}_{\mathcal{B}}\mathbb{T} \triangleq \{{}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{T} : 0 \le k \le K\}$ and ${}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{T}$ is the pose of the k^{th} checkerboard with respect to the camera. $\mathrm{r}_{\mathfrak{C}}$ is the reprojection function defined in (2.4).

To estimate the optimal parameters $({}^{\mathcal{C}}\boldsymbol{\omega}^*, {}^{\mathcal{C}}\mathbf{v}^*)$ for the LRF rotational axis, we perform the minimization with all points $\mathbf{x}_{ij}$ obtained over all rotational angles θ_i, $i = 0, \ldots, N$, that is

$$\arg\min_{{}^{\mathcal{C}}_{\mathcal{L}_0}\mathbf{T},{}^{\mathcal{C}}_{\mathcal{B}}\mathbb{T},{}^{\mathcal{C}}\boldsymbol{\omega},{}^{\mathcal{C}}\mathbf{v}} \frac{1}{\sigma^2_{\mathfrak{L}}}\sum_{j,k}\left\|{}^{\mathcal{L}_i}\mathbf{x}_{ij}^* - {}^{\mathcal{L}_i}\mathbf{x}_{ij}\right\|^2 + \frac{1}{\sigma^2_{\mathfrak{C}}}\sum_{r,c,k}\left\|{}^{\mathcal{I}}\mathbf{b}_{r,c} - \mathrm{r}_{\mathfrak{C}}\left({}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{R}\,{}^{\mathcal{B}}\mathbf{b}_{r,c} + {}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{t}\right)\right\|^2 . \tag{3.14}$$

In this optimization, only the pose of the LRF at the first rotational angle ${}^{\mathcal{C}}_{\mathcal{L}_0}\mathbf{T}$ is directly included as parameters; the pose of the LRF at the other rotational angles ${}^{\mathcal{C}}_{\mathcal{L}_i}\mathbf{T}$ are calculated from $({}^{\mathcal{C}}\boldsymbol{\omega}^*, {}^{\mathcal{C}}\mathbf{v}^*)$.

3.7 Experimental Results

To test our calibration algorithm, we use the LRF (SICK LMS-100) mounted on a pan-tilt unit (Directed Perception PTU-D46), as shown in Figure 3.6. Figure 3.7 shows the checkerboards setup for our calibration, with the checkerboard patterns being automatically detected using OpenCV [9]. Instead of using the minimum of 3 planes, which would give up to 8 solutions for the initial static camera-LRF calibration, 4 planes are used to simplify the computation. As described in Section 3.4, the normals of the checkerboard planes should span $\mathbb{R}^3$, so

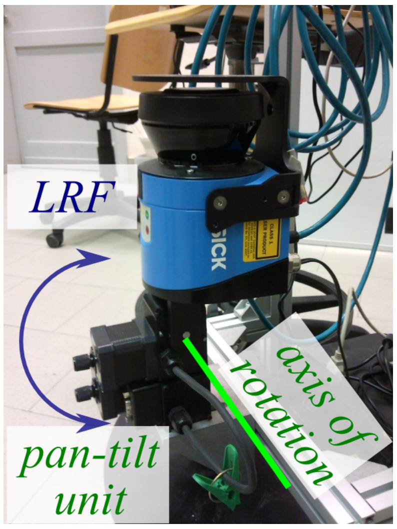


Figure 3.6: The 2D LRF mounted on a pan-tilt unit used in our calibration.

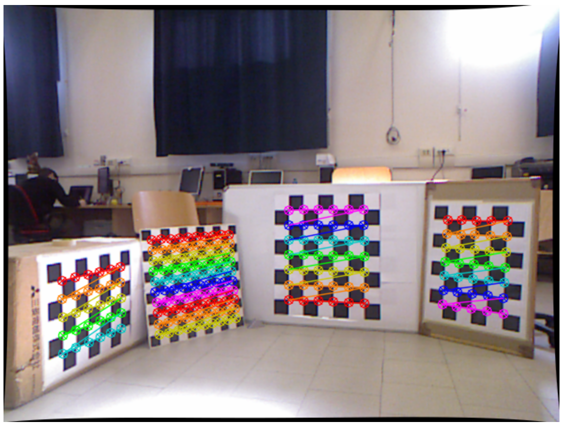


Figure 3.7: Checkerboard patterns extracted in camera image. As visible, not all the checkerboards are vertical, this is just to assure that the extracted normals span $\mathbb{R}^3$.

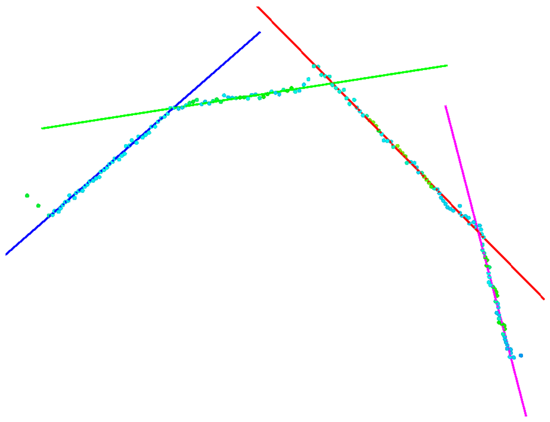


Figure 3.8: Results of the line-fitting procedure. Points in each laser scan belonging to the checkerboards are extracted using RANSAC line-fitting.

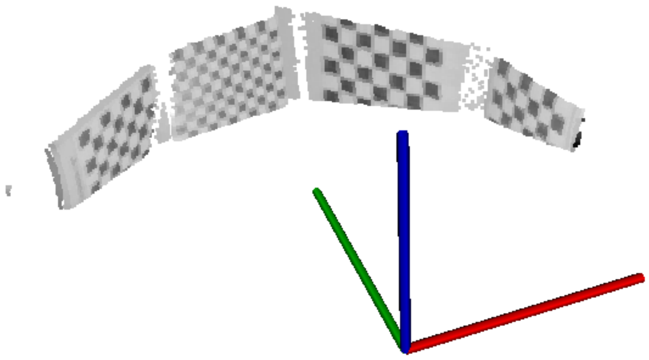


Figure 3.9: Laser points with intensity values reconstructed after calibration, showing a rough outline of the scanned checkerboard patterns.

care must be taken so that not all of the checkerboards are vertical. For example, if only the vertical walls in a room are used, $\{\mathbf{n}_k\}$ would only span $\mathbb{R}^2$, giving a degenerate configuration.

Figure 3.8 shows a single laser scan, with points belonging to the checkerboards identified by running RANSAC line-fitting over the scan data multiple times. For the final optimization in (3.14), we used $\sigma_{\mathfrak{L}} = 1.2$ cm, corresponding to the statistical error of the LMS-100 as given by the manufacturer, and $\sigma_{\mathfrak{C}} = 0.5$ px for the image localization noise. The results of our calibration are tabulated in Table 3.2, and checkerboards reconstructed using these parameters are shown in Figure 3.9. The reconstruction of a small indoor environment after calibration is shown in Figure 3.10.

The calibration results show a small change in the location and orientation of the rotation axis, as well as a slight decrease in the line-of-sight errors. Although

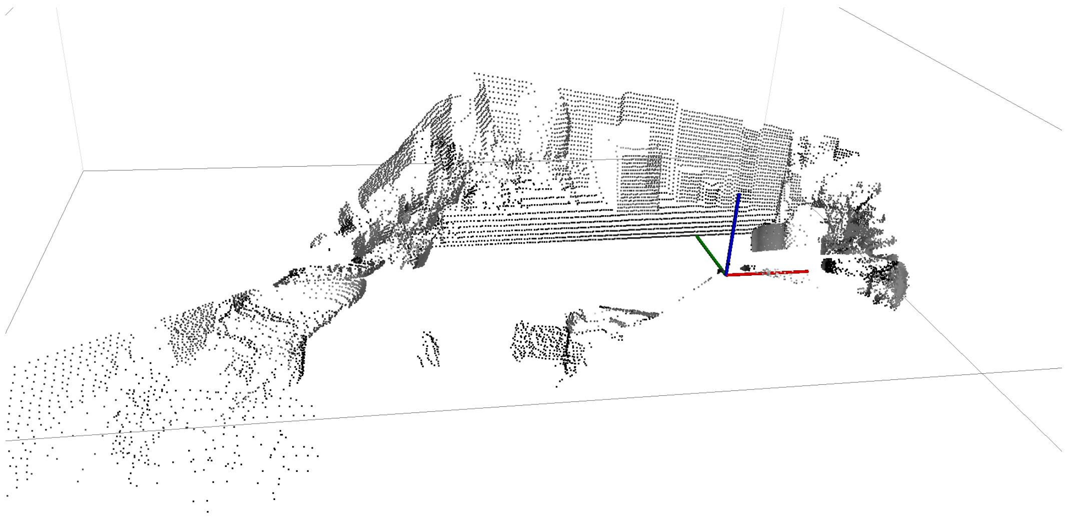


Figure 3.10: Indoor environment reconstructed after calibration.

Table 3.2: Calibration results.

Parameter	**Initial**	**Final**
Axis Translation Offset [cm]	$(0, -3, 16)$	$(0, -2.76, 15.89)$
Axis Rotational Offset [°]	0	10.14
Line-of-Sight Error [cm]	0.808	0.724

the final parameters of the rotational axis differ only slightly from those obtained using hand measurement, it accounts for all the degrees of freedom of the rotational axis. Furthermore, although the reconstructed checkerboards show only modest improvements, the importance of proper calibration will be more noticeable for data with larger range, as well as when fusing the range data with image data.

3.8 Conclusions

In this chapter, we proposed a method for recovering the rotational axis of a rotating 2D LRF. Instead of assuming that the rotational axis is aligned with one of the primary axis of the LRF, we modeled it as a line in 3D space with 4 DOF. The calibration consists of performing static camera-LRF calibration at two different rotational angles to obtain an initial estimate, followed by a non-linear optimization to refine the results.

However, instead of using existing static camera-LRF calibration methods, we modeled it as a kinematics point-plane constraints problem. This allows us to minimize the number of calibration planes that are needed, simplifying the calibration; also, various types of correspondences can be handled in a uniform manner. Furthermore, we described the minimization of the line-of-sight errors, which directly models the noise in the range measurements of a LRF.

Chapter 4

Depth Sensor Intrinsic Parameters Calibration

Typical robotic tasks like SLAM, navigation, object recognition and many others, highly benefit from having color and depth information fused together. While color information is typically provided by RGB cameras, there are plenty of sensors able to provide depth information: time-of-flight cameras, laser range scanners and sensors based on structured light. Even if there are some devices able to provide both color and depth data (e.g. the Kinect), as far as we know, there are no integrated sensors able to provide both color and depth information yet. In this chapter we focus on Kinect-like devices (among others, the Asus Xtion Pro Live). These sensors provide colored point clouds that suffer from a non accurate association between depth and RGB data, due to a non perfect alignment between the camera and the depth sensor. Moreover, depth images suffer from an irregular geometric distortion and have, for increasing distances, an increasing bias (i.e., a systematic error) in depth measurements.

These devices are factory calibrated, each one is sold with its own calibration parameter set stored inside a non-volatile memory. Unfortunately, the quality of this calibration is only adequate for gaming purposes: the depth distortion is not modeled in the factory calibration. Thus, a proper calibration method for robust robotics applications is needed.

In this chapter we propose a novel two-steps calibration method that employs a simple data collection procedure that only needs a minimally structured environment and that does not require any parameters tuning or a great interaction with the calibration software. Moreover, even if the principal targets of the method are the Kinect-like devices mentioned above, it is thought to be used also with any camera-depth sensor couples.

Given a calibrated camera and an uncalibrated depth sensor, the proposed method automatically infers the two correction maps for the depth sensor and, as a "side effect", the rigid body transformation that relates the two sensor frames. Unlike most of the existing works, our calibration method works directly on the depth images provided by the sensor: the (mostly unknown) relation between IR images and depth images is not taken into account.

For the depth sensor, we employ an error model that includes a pixel-based distortion error along with a systematic error (in the following also called "global error"). We propose to represent the undistortion map by means of a set of functions, iteratively fitted to the acquired data during a first calibration stage. We include the systematic error and alignment of the sensors in a second stage of the calibration: at this point, we exploit the plane-to-plane constraints between color and depth data to align the two sensors and to infer the systematic error inside a non-linear optimization framework.

The main contributions of this chapter to the scientific community are the following:

- a detailed analysis of the depth error of structured-light depth sensors such as the Kinect;
- an easy-to-implement calibration protocol, that provides the input data used for both the undistortion map and pose estimation processes;
- a spatial/parametric undistortion map that models in a compact and efficient way the distortion effect of Kinect-like depth sensors;
- a novel optimization framework that aims to estimate the camera-depth sensor alignment along with the parametric model that describes the systematic error on the depth measurements.

Moreover, an exhaustive set of tests aimed at proving the soundness of the proposed method is reported. The depth images of a test set corrected using the obtained calibration parameters are analyzed and compared to the ground truth. Results show that the calibration algorithm works as expected: the real depth is recovered up to an unpredictable noise. Finally, some results of the fusion of RGB and depth data are provided. Even in this case, the results are as good as expected, definitely better than those obtainable with the factory-provided calibration parameters.

The chapter is structured as follows: in Section 4.1 the existing works on the estimation of the depth sensor intrinsic parameters are reported and described. In Section 4.2 the error on the depth values provided by the sensor is analyzed. Section 4.3 gives a quick overview of the calibration procedure. The first calibration step is detailed in Section 4.4, while Section 4.5 describes the second calibration step. The results of the calibration procedure on a test for 3 different sensors are reported in Section 4.6 and details on the ROS implementation of the described algorithm are given in Section 4.7. Finally, some conclusions are drawn in Section 4.8.

4.1 Related Work

In recent years, many attempts to correct the Kinect depth data have been proposed. In one of the first works on the Kinect, Smisek et al. [49] showed that Kinect depth sensors are affected by a sort of radially symmetric distortions. To correct such distortion, they estimated a z-correction image constructed as the pixel-wise mean of the residuals of the plane fitted to the depth data. The z-correction image was subtracted from the z coordinate of the cloud points to obtain the real depth values.

Zhang et al. [67] proposed to treat the depth value z provided by the depth sensor of the Kinect as a linear function of the real one z^*, that is $z = \mu z^* + \eta$. They also detailed a method to estimate the parameters $\mu, \eta \in \mathbb{R}$ within the calibration algorithm.

In their papers, Herrera et al. [23, 24] described a calibration approach based on checkerboards, big planes and a third high resolution camera. Their method works on the raw data provided by the sensor (instead of on the metric data) and, alongside the depth correction, estimates the camera-depth sensor relative displacement. For what concerns the depth correction, they used a "spatially varying offset that decays as the Kinect disparity increases," i.e. they estimated a coefficient for each pixel $\mathsf{D}(u, v) \in \mathbb{R}$ and two global coefficients $\alpha_0, \alpha_1 \in \mathbb{R}$ such that the real depth value d^* can be computed as

$$d^* = d + \mathsf{D}(u, v) \cdot \exp(\alpha_0 - \alpha_1 \cdot d) \ .$$

Moreover, they used the 4 corners of the checkerboard plane as the initial guess of the relative displacement between the cameras and the depth sensor. For short distances their approach seems to work well as reported also in [52, 65].

An improvement over the work of Herrera et al. is the one presented by Raposo et al. [43]. They proposed several modifications to the estimation pipeline that allow their algorithm to achieve a calibration accuracy similar to [24] while using less than 1/6 of the input frames and running in 1/30 of the time.

Canessa et al. [10], instead, proposed to estimate a second degree polynomial for each pixel. In their work, authors first estimated the pose of a "virtual" depth sensor with respect to the RGB camera using an incandescent lamp bulb to light the checkerboard and make the depth map saturate in correspondence of the white squares. Then, they positioned a Kinect in front of a plane with a checkerboard attached and acquired a set of images from 0.6 m to 2 m. Finally, they fitted a second degree polynomial to the sample set of every pixel.

Teichman et al. [56] proposed a completely different calibration approach for Kinect-like devices: the undistortion map is estimated by means of a SLAM framework, in an unsupervised way. Their algorithm estimates 6 depth multipliers, at $0, 2, \ldots, 10$ meters, and corrects the depth measurements using a linear interpolation: to the best of our knowledge, this is the only approach that proved to be able to correct depth data at more than 5 meters. The main drawback of their approach is the time it needs to reach a solution: the optimization procedure takes hours to converge. Moreover, according to [65], it seems not to perform as well as [24] for short distances. In fact, differently from our approach, close-range data (less than 2 meters) are considered reliable and used in the SLAM pipeline to infer the geometry of the scene.

Di Cicco et al. [13] even proposed an unsupervised calibration method. Their algorithm estimates an undistortion map for the depth data using a machine learning approach. The best plane fitted to the data is used as reference, i.e., the average depth is considered reliable.

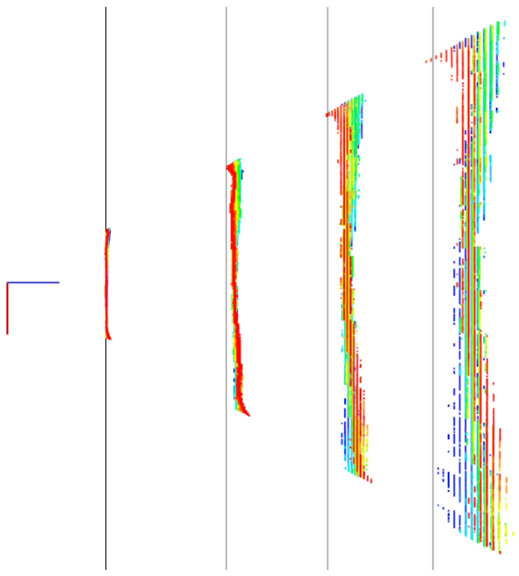
Fiedler et al. [15] investigated a possible influence of thermal and environmental conditions when calibrating Kinect. The experiment turned out that variations of the temperature and air draft have a notable influence on Kinect images and range measurements. Based on the findings, they derived temperature-related rules to reduce errors in the calibration and measurement process of the Kinect.

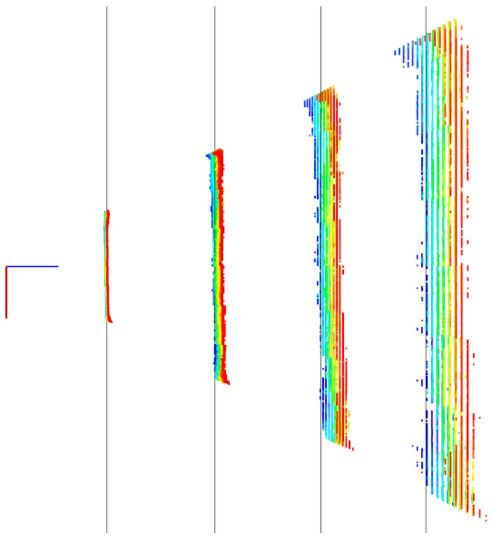
In 2008, Kim et al. [28] presented a calibration approach for ToF cameras. Even though the error pattern and bias of a ToF camera are completely different from the ones of a Kinect, we have noticed that they used an approach very similar to ours. They firstly analyzed the depth error, separated it into components, and derived a way to correct each component.

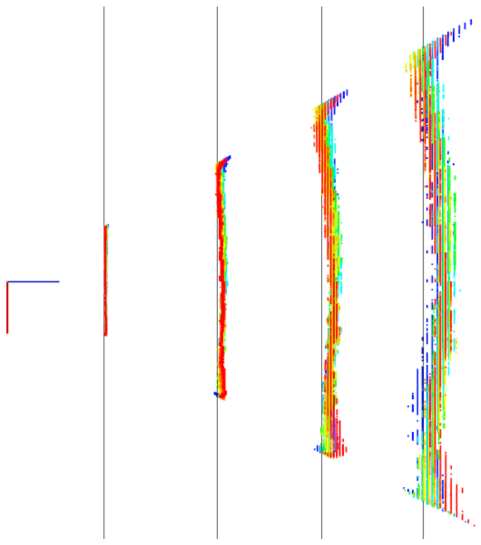
4.2 Depth Error Analysis

The noise on the range values provided by Kinect-like depth sensors has been described in many works [27, 66]. It is well known that, since the Kinect is essentially a stereo camera (see Section 2.3.3), the depth resolution is proportional to the square of the depth, while the error on the depth measurements is a combination of a noise component and a quantization error component introduced by the hardware constraints.

To analyze the error on the depth measurements, we positioned three sensors, rigidly mounted on a single support, in front of a flat wall at increasing distances. For each position, we collected the sensors readings while measuring the real dis-


(a) KINECT47A – top


(b) KINECT51A – top


(c) ASUS – top

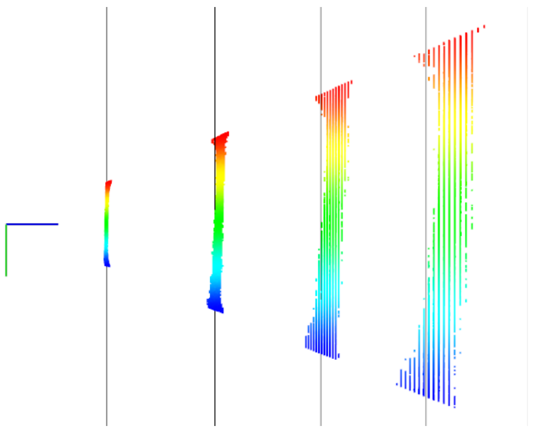

(d) KINECT51A – side

Figure 4.1: Top and side views of the point clouds generated by some different projected-light depth sensors: two Kinects (KINECT47A, KINECT51A) and an Asus Xtion Pro Live (ASUS). The gray lines represent the ground truth measured with the laser distance meters.

tances (i.e., the ground truth) by means of two high precision laser distance meters (we used both the laser readings to avoid the support to rotate around its vertical axis). Some qualitative results of such analysis are reported in Figure 4.1. From the images we immediately notice that:

1. the 3D surface defined by the points is not properly planar as it should be;
2. the average depth of the points is not correct and, sometimes, even the average orientation is wrong;
3. the quantization error becomes not negligible for increasing distances;
4. each sensor has a different "distortion pattern" and this effect becomes more accentuated for increasing distances (this is the *myopic* property defined in [56]).

We call *distortion* the depth error that produces a local alteration of an object shape and *global error* the *systematic* wrong estimation of the average depth. In Figure 4.2 the global error for three different sensors, i.e. the difference between the average depth of the acquired clouds and the ground truth, is reported. It can be noticed that such error is super-linear with respect to the measured depth values.

We also compared the measured point clouds with the planes that best fit to them (some results are in Figure 4.3). In particular, for each incoming point cloud, we computed the Root Mean Square (RMS) error on the (signed) distance between the plane and the points. Also in this case, the error is clearly super-linear.

4.2.1 Error Correction Model

To model the effects of the errors introduced by depth sensors, as in [10, 13, 24, 56], we propose to estimate a depth correction function in a per-pixel basis. That is, given a depth image J of size $H_{\mathfrak{D}} \times W_{\mathfrak{D}}$, a pixel $(u, v)^\mathsf{T} \in \mathbb{I}_{\mathfrak{D}}$ and the corresponding depth value d, the real depth d^* is computed as

$$d^* = \mathrm{f}_{u,v}(d). \tag{4.1}$$

Starting from the considerations made in Section 4.2, we express each $\mathrm{f}_{u,v}(\cdot)$ in (4.1) as a composition of two functions: $\mathrm{u}_{u,v}(\cdot)$ that takes into account the local distortion, and $\mathrm{g}_{u,v}(\cdot)$ that makes a global correction of the depth values. That is, the real depth d^* is estimated as

$$d^* = \mathrm{f}_{u,v}(d) = (\mathrm{g} \circ \mathrm{u})_{u,v}(d), \tag{4.2}$$

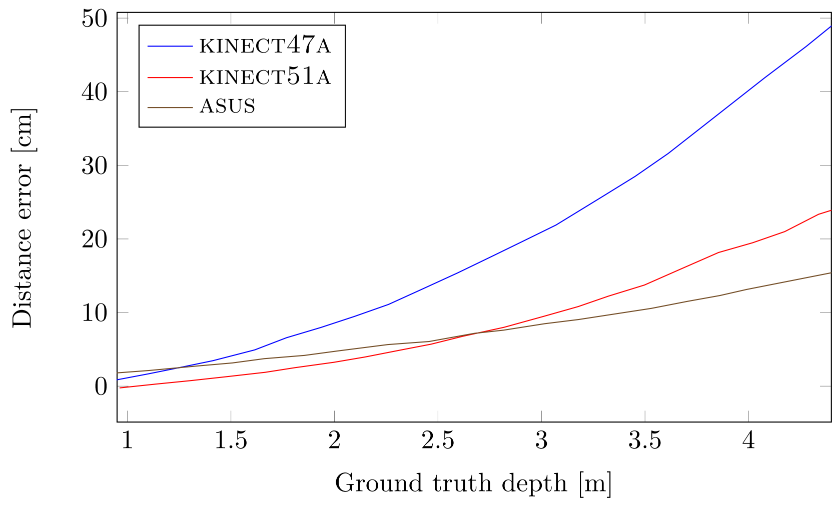


Figure 4.2: Error on the average distance point estimation, for three different depth sensors. The error is computed, for each cloud, by averaging the depth of all its points and comparing it to the ground truth computed with the two laser distance meters.

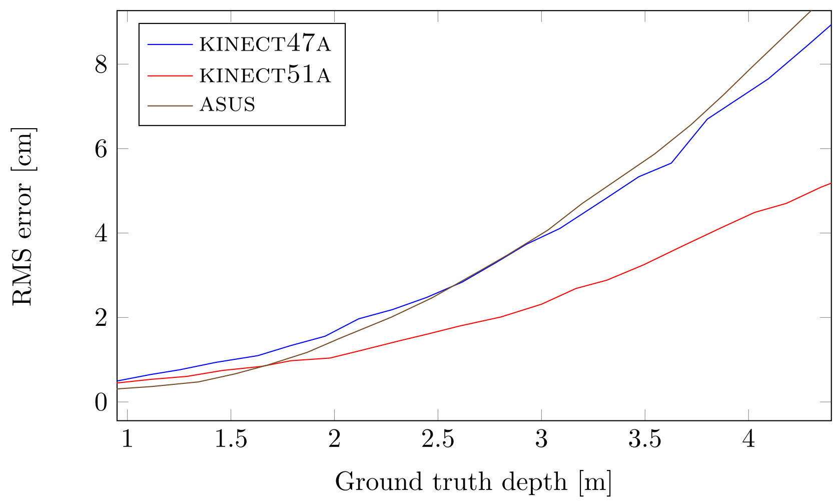


Figure 4.3: RMS error caused by the distortion, for three different depth sensors. The error is computed by fitting a plane to the depth data acquired in front of a flat wall and computing the point-plane distance for all its points.

or, alternatively, the real 3D point $\mathbf{x}^*$ is estimated as

$$\mathbf{x}^* = \mathrm{f}_{u,v}(\mathbf{x}) = (\mathrm{g} \circ \mathrm{u})_{u,v}(\mathbf{x}),$$

where

$$\mathrm{f}_{u,v}(\mathbf{x}) = \mathbf{x} \cdot \frac{\mathrm{f}_{u,v}(d)}{d}.$$

We define U as the map that associates a pixel $(u,v)^\mathsf{T} \in \mathbb{I}_\mathfrak{D}$ to an undistortion function $\mathrm{u} : \mathbb{R} \to \mathbb{R}$, that is, $\mathsf{U}(u,v) = \mathrm{u}_{u,v}(\cdot)$. In the same way, $\mathsf{G}(u,v) = \mathrm{g}_{u,v}(\cdot)$.

4.3 Calibration Approach

As confirmed by the experimental evidence (Section 4.2), the error on the depth measurements is a continuous function. Thus we can assume that given two close 3D points $\mathbf{x}$ and $\mathbf{y}$ along the same direction, i.e $\mathbf{y} = (1+\varepsilon) \cdot \mathbf{x}$ with $\varepsilon \simeq 0$,

$$\mathbf{y}^* = \mathrm{f}(\mathbf{y}) = \mathrm{f}((1+\varepsilon) \cdot \mathbf{x}) \simeq (1+\varepsilon) \cdot \mathrm{f}(\mathbf{x}) = (1+\varepsilon) \cdot \mathbf{x}^*.$$

where $\mathrm{f}(\cdot)$ is the error correction function defined in (4.2). This means that, if we know how to "correct" a point $\mathbf{x}$ (i.e. we know the correction function parameters for this point), we can correct close points with a good approximation using the same parameters.

This assumption is the basis of our algorithm to estimate both the undistortion map U and the global error correction map G. Exploiting the fact that both distortion and quantization error become more severe for increasing distances, we introduce the idea to estimate the distortion error iteratively, starting from short distances and estimating the error for greater distances using as initial guess the current correction parameters.

The proposed calibration framework requires the depth sensor to be coupled with an RGB camera that frames the same scene, the rigid body transformation that relates the two sensors will be estimated while inferring the depth error correction function. It also requires the two sensors to collect data framing a scene that includes a wall with a checkerboard attached on it, at different distances and orientations.

The calibration is performed in two steps: in the first step the algorithm estimates the undistortion map U; only a rough calibration between the camera and the depth sensor is necessary during this step, the checkerboard is used just to have an idea of the wall location. In the second step the global correction map G is computed. Here the checkerboard poses estimated with the (calibrated) RGB camera, are used as a ground truth. That is, the undistorted planes estimated in the first step are forced to match the ones defined by the checkerboard. To this end, the real rigid displacement between the RGB camera $\mathfrak{C}$ and the depth sensor $\mathfrak{D}$ needs to be known. Unfortunately, to estimate the pose of one sensor with respect to the other, a good estimate of their intrinsic parameters is mandatory. One way to satisfy this circular dependency is to estimate the global correction map G and the rigid body displacement ${}^{\mathcal{C}}_{\mathcal{D}}\mathbf{T}$ simultaneously.

At this point a question arises: why the depth error is corrected in two steps? Actually, the reason is simple. To guarantee the best results, the camera-depth sensor transformation ${}^{\mathcal{C}}_{\mathcal{D}}\mathbf{T}$ and the global correction map G need to be refined together within an optimization framework. Refine a map such as U, with a different function every pixel, is not a feasible solution: at least 10 thousands parameters are needed. G instead, whose scope is to transform planes into planes, is an object much more simple to deal with in the optimization phase: as we will see later, such map needs no more than a dozen parameters.

4.3.1 Pipeline

The algorithm is organized as in Figure 4.4. First of all, RGB images I_k are analyzed and the checkerboard corners are extracted from the images. The corners, ${}^{\mathcal{I}}\mathsf{B}_k$, the point clouds ${}^{\mathcal{D}}\mathsf{C}_k$ and the initial camera-depth transform ${}^{\mathcal{C}}_{\mathcal{D}}\mathbf{T}_0$ are the inputs for the undistortion map estimation procedure. Once the undistortion map U has been estimated, the point clouds are undistorted (${}^{\mathcal{D}}\widehat{\mathsf{C}}_k$) and passed to the procedure that estimates both the global correction map G and the camera-depth transformation ${}^{\mathcal{C}}_{\mathcal{D}}\mathbf{T}$.

4.4 Undistortion Map Estimation

The proposed algorithm (Algorithm 4.1) estimates the undistortion map U taking as input a list $({}^{\mathcal{D}}\mathsf{C}_1, {}^{\mathcal{D}}\mathsf{C}_2, \ldots, {}^{\mathcal{D}}\mathsf{C}_M)$ of point clouds acquired when the depth sensor $\mathfrak{D}$ is pointing a planar surface (e.g. a wall) at different distances and orientations. It also requires the positions of the checkerboard corners $({}^{\mathcal{I}}\mathsf{B}_1, {}^{\mathcal{I}}\mathsf{B}_2, \ldots, {}^{\mathcal{I}}\mathsf{B}_M)$, extracted from the images, and a rough estimate of the rigid-body transformation that relates the two sensors ${}^{\mathcal{C}}_{\mathcal{D}}\mathbf{T}_0$.

Firstly ($\ell\ell$. 1-2) the undistortion map U is initialized as an $H_{\mathfrak{D}} \times W_{\mathfrak{D}}$ matrix of identity functions $\mathbb{1} : \mathbb{R} \to \mathbb{R}$ while the sample matrix E_{U} is initialized as an $H_{\mathfrak{D}} \times W_{\mathfrak{D}}$ matrix of empty sets (the matrix E_{U} keeps in memory the samples to fit the undistortion functions $\mathrm{u}_{u,v}(\cdot)$). Then, the point cloud list $({}^{\mathcal{D}}\mathsf{C}_1, {}^{\mathcal{D}}\mathsf{C}_2, \ldots, {}^{\mathcal{D}}\mathsf{C}_M)$ is sorted in ascending order according to the distance of the main plane (i.e., the plane with the checkerboard) from the sensor (ℓ. 3), to exploit the continuity

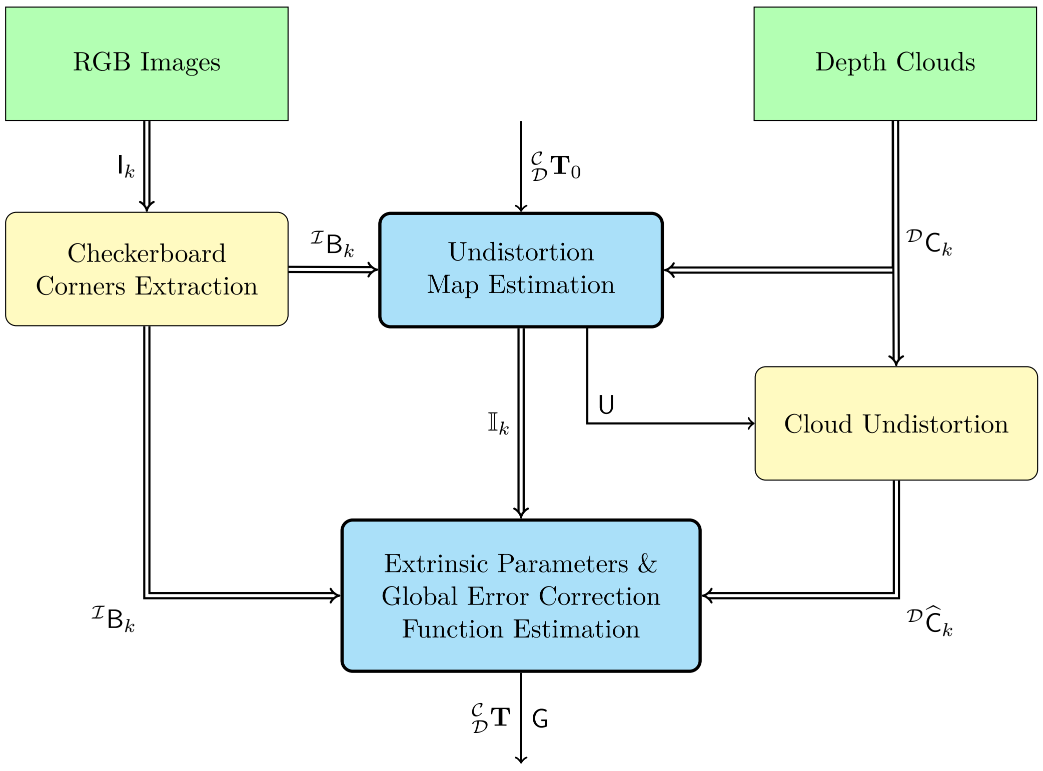

RGB Images
Depth Clouds
I_k
${}^{\mathcal{C}}_{\mathcal{D}}\mathbf{T}_0$
Checkerboard Corners Extraction
${}^{\mathcal{I}}\mathsf{B}_k$
Undistortion Map Estimation
${}^{\mathcal{D}}\mathsf{C}_k$
$\mathbb{I}_k$
U
Cloud Undistortion
Extrinsic Parameters & Global Error Correction Function Estimation
${}^{\mathcal{I}}\mathsf{B}_k$
${}^{\mathcal{D}}\widehat{\mathsf{C}}_k$
${}^{\mathcal{C}}_{\mathcal{D}}\mathbf{T}$
G

Figure 4.4: Calibration algorithm pipeline. Double-lined arrows mean that a set of data is passed from one block to the other.

Algorithm 4.1 Undistortion Map Estimation

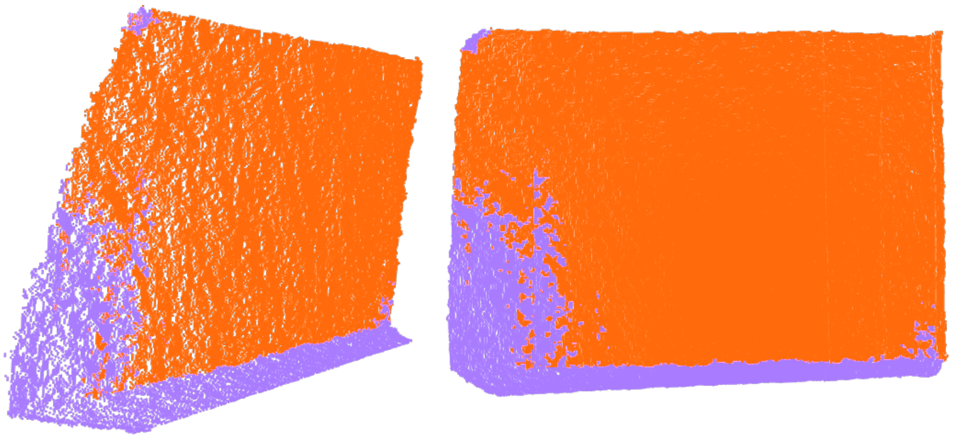
```
Input: $({}^{\mathcal{D}}\mathsf{C}_1, {}^{\mathcal{D}}\mathsf{C}_2, \dots, {}^{\mathcal{D}}\mathsf{C}_M)$    ▷ Depth point clouds
Input: $({}^{\mathcal{I}}\mathsf{B}_1, {}^{\mathcal{I}}\mathsf{B}_2, \dots, {}^{\mathcal{I}}\mathsf{B}_M)$    ▷ Checkerboard corners in the images
Input: ${}^{\mathcal{C}}_{\mathcal{D}}\mathbf{T}_0$    ▷ Camera-depth sensors initial transformation
Output: $\mathsf{U}$    ▷ Undistortion map
 1: $\mathsf{U} \leftarrow (\mathbb{1}_i)_{i \in \mathbb{I}_{\mathfrak{D}}}$
 2: $\mathsf{E}_{\mathsf{U}} \leftarrow (\emptyset_i)_{i \in \mathbb{I}_{\mathfrak{D}}}$
 3: $({}^{\mathcal{D}}\mathsf{C}_{s_1}, {}^{\mathcal{D}}\mathsf{C}_{s_2}, \dots, {}^{\mathcal{D}}\mathsf{C}_{s_M}) \leftarrow \textsc{sort}({}^{\mathcal{D}}\mathsf{C}_1, {}^{\mathcal{D}}\mathsf{C}_2, \dots, {}^{\mathcal{D}}\mathsf{C}_M)$
 4: for $k \leftarrow 1, 2, \dots, M$ do
 5:     for all $(u, v) \in \mathbb{I}_{\mathfrak{D}}$ do
 6:         $\mathrm{u} \leftarrow \mathsf{U}(u, v)$
 7:         ${}^{\mathcal{D}}\mathbf{x} \leftarrow {}^{\mathcal{D}}\mathsf{C}_{s_k}(u, v)$
 8:         ${}^{\mathcal{D}}\widehat{\mathsf{C}}_{s_k}(u, v) \leftarrow \mathrm{u}({}^{\mathcal{D}}\mathbf{x})$
 9:     end for
10:     $\mathbb{I}_{s_k} \leftarrow \textsc{selectWallPoints}({}^{\mathcal{D}}\widehat{\mathsf{C}}_{s_k}, {}^{\mathcal{I}}\mathsf{B}_{s_k}, {}^{\mathcal{C}}_{\mathcal{D}}\mathbf{T}_0)$
11:     ${}^{\mathcal{D}}\pi_{s_k} \leftarrow \textsc{fitPlane}({}^{\mathcal{D}}\mathsf{C}_{s_k}, \mathbb{I}_{s_k})$
12:     $(\mathsf{U}, \mathsf{E}_{\mathsf{U}}) \leftarrow \textsc{updateMap}(\mathsf{U}, \mathsf{E}_{\mathsf{U}}, {}^{\mathcal{D}}\mathsf{C}_{s_k}, \mathbb{I}_{s_k}, {}^{\mathcal{D}}\pi_{s_k})$
13: end for
```

described in Section 4.3.

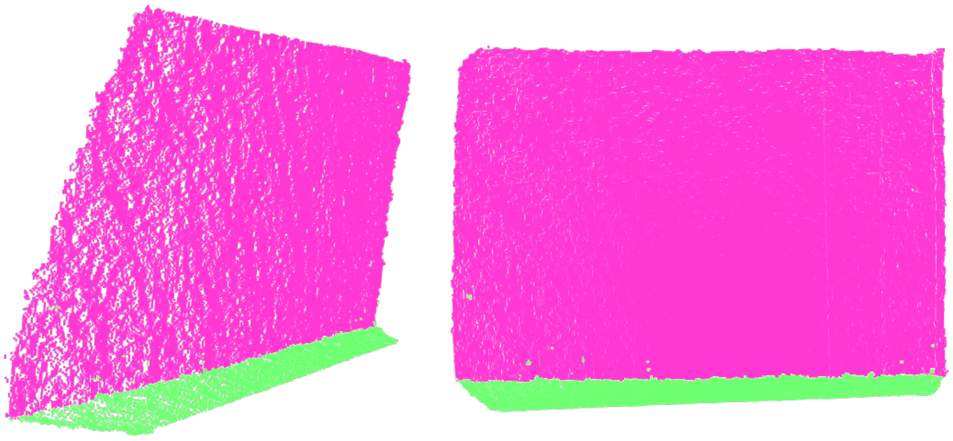
The undistortion map is created iteratively: at each step only one point cloud is analyzed. At step k, for example, the k^{th} cloud in the list, ${}^{\mathcal{D}}\mathsf{C}_{s_k}$, is undistorted using the *current* estimation of U ($\ell\ell$. 5-9). Then, the indices $\mathbb{I}_{s_k} \subseteq \mathbb{I}_{\mathfrak{D}}$ of the wall points are extracted from the undistorted cloud ${}^{\mathcal{D}}\widehat{\mathsf{C}}_{s_k}$ and used to fit a plane ${}^{\mathcal{D}}\pi_{s_k}$ to the initial cloud ${}^{\mathcal{D}}\mathsf{C}_{s_k}$ ($\ell\ell$. 10-11). Actually, to increase stability, instead of fitting a plane to the whole original point cloud, only the pixels within a defined ray from the image center are used, as reported in [13]. Finally (ℓ. 12), the estimated plane ${}^{\mathcal{D}}\pi_{s_k}$ is used to compute the undistortion map U. The procedure ends as soon as the last cloud in the list has been processed.

4.4.1 Wall Points Selection

The selection of the wall point-indices is performed automatically, as opposed to the manual selection of [24]. We take advantage of the RGB camera and the checkerboard to select the right plane and extract its indices from the undistorted cloud with a RANSAC-based approach [16, 44]. As shown in Figure 4.5, the undistorted cloud let us always extract the correct points, while the original one does not. In particular, points near the image corners are likely to be excluded from the inliers when using the original cloud.



(a) Wall points extracted from the original point cloud ${}^{\mathcal{D}}\mathsf{C}_{s_k}$.



(b) Wall points extracted from the undistorted point cloud ${}^{\mathcal{D}}\widehat{\mathsf{C}}_{s_k}$.

Figure 4.5: Comparison between (a) the wall points extracted from the original point cloud ${}^{\mathcal{D}}\mathsf{C}_{s_k}$ and (b) the ones extracted from the undistorted cloud ${}^{\mathcal{D}}\widehat{\mathsf{C}}_{s_k}$. As clearly visible, in both cases the floor points are correctly discarded. In the former case, however, the wall segmentation is wrong.

4.4.2 Map Update

In the map update function (Algorithm 4.2), all the points ${}^{\mathcal{D}}\mathbf{x} = (x, y, z)^\mathsf{T}$ of the cloud are projected on the previously extracted plane ${}^{\mathcal{D}}\pi_{s_k}$ along their line-of-sight (ℓℓ. 3-4). That is, let $\mathbf{n}^\mathsf{T}\mathbf{x} - d = 0$ be the plane equation, and let $l\,{}^{\mathcal{D}}\mathbf{x}$, $l \in \mathbb{R}$, be the points along ${}^{\mathcal{D}}\mathbf{x}$ line-of-sight, then the line-of-sight projection of ${}^{\mathcal{D}}\mathbf{x}$ onto ${}^{\mathcal{D}}\pi_{s_k}$, say ${}^{\mathcal{D}}\mathbf{x}_\pi = (x_\pi, y_\pi, z_\pi)^\mathsf{T}$, is

$$\mathbf{x}_\pi = l\mathbf{x} = \frac{d\mathbf{x}}{\mathbf{n}^\mathsf{T}\mathbf{x}} \ .$$

The pair (z, z_π) is used as a sample for the curve-fitting procedure (ℓ. 5), and the undistortion function $\mathsf{U}(u, v)$ is re-estimated by fitting a new curve to the sample set $\mathsf{E}_\mathsf{U}(u, v)$ (ℓ. 6).

Algorithm 4.2 Undistortion Map Estimation – Update Map

```
1: function UPDATEMAP($\mathsf{U}, \mathsf{E}_\mathsf{U}, {}^{\mathcal{D}}\mathsf{C}, \mathbb{I}, {}^{\mathcal{D}}\pi$)
2:     for all $(u, v) \in \mathbb{I}$ do
3:         ${}^{\mathcal{D}}\mathbf{x} \leftarrow {}^{\mathcal{D}}\mathsf{C}(u, v)$
4:         ${}^{\mathcal{D}}\mathbf{x}_\pi \leftarrow$ LOSPROJECT(${}^{\mathcal{D}}\mathbf{x}, {}^{\mathcal{D}}\pi$)
5:         $\mathsf{E}_\mathsf{U}(u, v) \leftarrow \mathsf{E}_\mathsf{U}(u, v) \cup \{(z, z_\pi)\}$
6:         $\mathsf{U}(u, v) \leftarrow$ FITCURVE($\mathsf{E}_\mathsf{U}(u, v)$)
7:     end for
8:     return $(\mathsf{U}, \mathsf{E}_\mathsf{U})$
9: end function
```

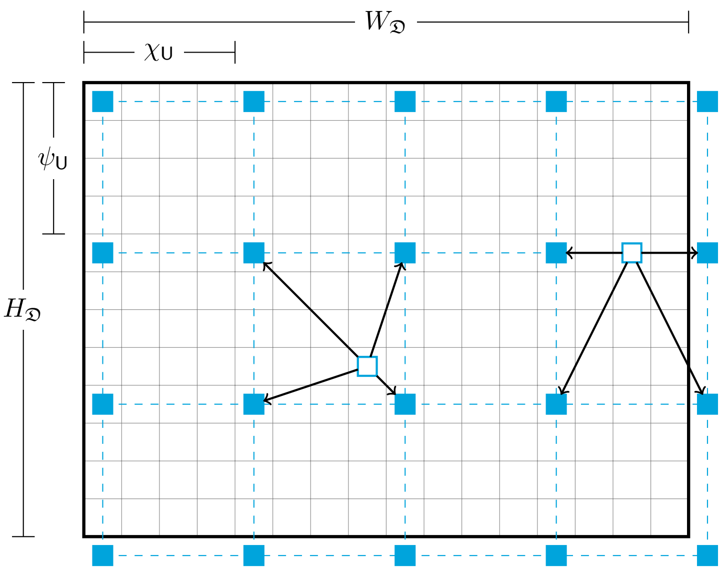
4.4.3 Implementation Details

Undistortion Map

To decrease the incidence of noise on the map estimation we reduce the number of functions fitted to the data. That is, instead of estimating an undistortion function for each pixel, similarly to [56], we discretize the map into bins. So, let $\chi_\mathsf{U}, \psi_\mathsf{U} \in \mathbb{N}$ be the bin size in pixels, along the image x- and y directions, respectively.

Given a pixel $(u, v)^\mathsf{T} \in \mathbb{I}_\mathfrak{D}$, we define $\mathsf{S}_\mathsf{U}(u, v)$ as the set of 4 pixels *surrounding* $(u, v)^\mathsf{T}$ according to the sampling factors χ_U and ψ_U (see Figure 4.6). We also define $\mathbb{S}_\mathsf{U} \triangleq \{\mathsf{S}_\mathsf{U}(u, v) : (u, v)^\mathsf{T} \in \mathbb{I}_\mathfrak{D}\}$ as the set of all the *surrounding pixels*.

We estimate the undistortion function $\mathrm{u}_{u,v}(\cdot)$ only for the pixels in $\mathbb{S}_\mathsf{U}$. For all the others, instead, this function is computed as a linear combination of the functions computed for the pixels set $\mathbb{S}_\mathsf{U}$. That is, given a pixel $(u, v)^\mathsf{T}$, its undistortion



■ Pixel $(u,v)^{\mathsf{T}} \in \mathbb{S}_{\mathsf{U}}$. For this pixel an undistortion function $\mathrm{u}_{u,v}(\cdot)$ has been estimated.

□ Pixel $(u,v)^{\mathsf{T}} \in \mathbb{I}_{\mathfrak{D}}$. For this pixel the undistortion function $\mathrm{u}_{u,v}(\cdot)$ is a linear combination of the functions of the pixels in $\mathsf{S}_{\mathsf{U}}(u,v)$.

$\rightarrow$ Connection from a pixel $(u,v)^{\mathsf{T}} \in \mathbb{I}_{\mathfrak{D}}$ to one in $\mathsf{S}_{\mathsf{U}}(u,v)$.

Figure 4.6: Visualization of an undistortion map U. Given two parameters, χ_{U} and ψ_{U}, an undistortion function is estimated for all and only the pixels in $\mathbb{S}_{\mathsf{U}}$. For all the others, instead, the function is computed as a linear combination of the ones estimated for the pixels in $\mathbb{S}_{\mathsf{U}}$.

function $\mathsf{U}(u,v)$ is

$$\mathsf{U}(u,v) = \sum_{(s,t)\in\mathsf{S}_{\mathsf{U}}(u,v)} \mathrm{w}_{\chi_{\mathsf{U}}}(u,s) \cdot \mathrm{w}_{\psi_{\mathsf{U}}}(v,t) \cdot \mathsf{U}(s,t)$$

where

$$\mathrm{w}_{\chi_{\mathsf{U}}}(u,s) \triangleq 1 - \frac{|u-s|}{\chi_{\mathsf{U}}}, \quad \mathrm{w}_{\psi_{\mathsf{U}}}(v,t) \triangleq 1 - \frac{|v-t|}{\psi_{\mathsf{U}}} \tag{4.3}$$

and

$$\sum_{(s,t)\in\mathsf{S}_{\mathsf{U}}(u,v)} \mathrm{w}_{\chi_{\mathsf{U}}}(u,s) \cdot \mathrm{w}_{\psi_{\mathsf{U}}}(v,t) = 1 \ .$$

Curve Fitting

As shown in Section 4.2 (Figure 4.3), the distortion is super-linear, therefore an appropriate correction function must be chosen. Moreover, as described in the previous section, since we are not estimating a function for every pixel, the fitting procedure is not straightforward.

For what concerns the former point, suppose the error is corrected by a second degree polynomial, that is, $\mathsf{U}(u, v) = \mathrm{u}_{u,v}(z) = a + bz + cz^2$, for some $a, b, c \in \mathbb{R}$. To estimate the polynomial coefficients we solve a non-linear least squares problem of the form

$$\arg\min_{a,b,c} \sum_{(z,z_\pi)\in \mathsf{E}_\mathsf{U}(u,v)} \frac{1}{\sigma^2(z)} \left\| a + bz + cz^2 - z_\pi \right\|^2$$

where $\sigma(z)$ is the error on the depth measurements.

For what concerns the latter point, i.e. how to deal with the discretized undistortion map, we slightly modify the sample set generation and the function fitting procedure described in Algorithm 4.2. In the new algorithm (Algorithm 4.3), every pixel $(u, v)^\mathsf{T}$ contributes to the sample set of its four surrounding pixels $\mathsf{S}_\mathsf{U}(u, v)$ with a weight calculated as in (4.3). That is, let

$$\mathsf{S}_\mathsf{U}^{-1}(s, t) \triangleq \{(u, v) \in \mathbb{I}_\mathfrak{D} : (s, t) \in \mathsf{S}_\mathsf{U}(u, v)\}$$

be the set of pixels which have (s, t) as one of their surrounding pixels. For each cloud, the temporary sample set $\mathsf{E}_w(s, t)$ for a pixel $(s, t) \in \mathbb{S}_\mathsf{U}$, is ($\ell\ell$. 2-10)

$$\mathsf{E}_w(s, t) \triangleq \bigcup_{(u,v)\in \mathsf{S}^{-1}(s,t)} (w, z, z_\pi)$$

where

$$w \triangleq \mathrm{w}_{\chi_\mathsf{U}}(u, s) \cdot \mathrm{w}_{\psi_\mathsf{U}}(v, t).$$

$\mathsf{E}_w(s, t)$ is used to generate the sample set for the aforementioned curve fitting procedure ($\ell\ell$. 11-15). Basically, the pair $(\bar{z}, \bar{z}_\pi)$ is calculated as the weighted arithmetic mean of the values in $\mathsf{E}_w(s, t)$, that is

$$\begin{aligned} W &\triangleq \sum_{(w,z,z_\pi)\in \mathsf{E}_w(u,v)} w, \\ \bar{z} &\triangleq \frac{1}{W} \sum_{(w,z,z_\pi)\in \mathsf{E}_w(u,v)} w \cdot z, \\ \bar{z}_\pi &\triangleq \frac{1}{W} \sum_{(w,z,z_\pi)\in \mathsf{E}_w(u,v)} w \cdot z_\pi, \end{aligned}$$

and added to the sample set $\mathsf{E}_\mathsf{U}(s, t)$.

Algorithm 4.3 Undistortion Map Estimation – Update Map Revised

```
1:  function updateMap($\mathsf{U}, \mathsf{E}_\mathsf{U}, {}^{\mathcal{D}}\mathsf{C}, \mathbb{I}, {}^{\mathcal{D}}\pi$)
2:      $\mathsf{E}_w \leftarrow (\emptyset_i)_{i \in \mathbb{I}_\mathfrak{D}}$
3:      for all $(u, v) \in \mathbb{I}$ do
4:          ${}^{\mathcal{D}}\mathbf{x} \leftarrow {}^{\mathcal{D}}\mathsf{C}(u, v)$
5:          ${}^{\mathcal{D}}\mathbf{x}_\pi \leftarrow$ losProject(${}^{\mathcal{D}}\mathbf{x}, {}^{\mathcal{D}}\pi$)
6:          for all $(s, t) \in \mathsf{S}_\mathsf{U}(u, v)$ do
7:              $w \leftarrow \mathrm{w}_{\chi_\mathsf{U}}(u, s) \cdot \mathrm{w}_{\psi_\mathsf{U}}(v, t)$
8:              $\mathsf{E}_w(s, t) \leftarrow \mathsf{E}_w(s, t) \cup \{(w, z, z_\pi)\}$
9:          end for
10:     end for
11:     for all $(s, t) \in \mathbb{S}_\mathsf{U}$ do
12:         $(\bar{z}, \bar{z}_\pi) \leftarrow$ weightedMean($\mathsf{E}_w(s, t)$)
13:         $\mathsf{E}_\mathsf{U}(s, t) \leftarrow \mathsf{E}_\mathsf{U}(s, t) \cup \{(\bar{z}, \bar{z}_\pi)\}$
14:         $\mathsf{U}(s, t) \leftarrow$ fitCurve($\mathsf{E}_\mathsf{U}(s, t)$)
15:     end for
16:     return $(\mathsf{U}, \mathsf{E}_\mathsf{U})$
17: end function
```

4.5 Global Correction Map Estimation

Our original solution to deal with the global, systematic error was to have a unique function, say $\mathrm{g}(\cdot)$, to correct the wrong depth measurements after the undistortion phase, i.e. $\mathsf{G}(u, v) = \mathrm{g}(\cdot)$, for all $(u, v)^\mathsf{T} \in \mathbb{I}_\mathfrak{D}$. Unfortunately, this solution had one important limitation: in some cases the undistorted clouds were both translated and rotated around a non-predictable axis. For this reason we moved to a correction map G someway similar to the previously described undistortion map U. The actual implementation of such map is described in Section 4.5.3. Our algorithm takes as input a set of already undistorted point clouds $({}^{\mathcal{D}}\widehat{\mathsf{C}}_1, {}^{\mathcal{D}}\widehat{\mathsf{C}}_2, \dots, {}^{\mathcal{D}}\widehat{\mathsf{C}}_M)$, the correspondent wall point indices $(\mathbb{I}_1, \mathbb{I}_2, \dots, \mathbb{I}_M)$ and the checkerboard corners extracted from the images, $({}^{\mathcal{I}}\mathsf{B}_1, {}^{\mathcal{I}}\mathsf{B}_2, \dots, {}^{\mathcal{I}}\mathsf{B}_M)$. After an initialization step where a rough estimate of the map G is computed (Section 4.5.1), the map is refined, along with the camera-depth sensor transformation ${}^{\mathcal{D}}_{\mathcal{C}}\mathbf{T}$, within a non-linear optimization framework (Section 4.5.2).

4.5.1 Initial Estimation

The initial estimation of the map functions as well as the computation of the rigid transform between the RGB and the depth sensor is reported in Algorithm 4.4. Firstly, the pose of one sensor with respect to the other is estimated, that is, for each color-depth image pair both the plane defined by the checkerboard in the image ${}^{\mathcal{C}}\pi_{\mathsf{B}_k}$ (in camera coordinates), and the one extracted from the point cloud ${}^{\mathcal{D}}\pi_{\widehat{\mathsf{C}}_k}$ (in depth sensor coordinates) are computed from the given input data (*ℓℓ.* 1-6). The checkerboard pose is extracted from its corners ${}^{\mathcal{I}}\mathsf{B}_k$ as described in Section 2.2.1 (*ℓℓ.* 2-3): the equation of the plane framed by the RGB camera is hence computed taking 3 non-collinear corners (*ℓ.* 4). The equation of the plane in the depth image, instead, is computed using a SVD approach. Once all the planes have been computed, the rigid displacement between the two sensors is estimated (*ℓℓ.* 7-9) using the plane-to-plane calibration method described in [58].

The plane equations extracted from the images are then represented w.r.t. the depth sensor reference frames using the transformation matrices just computed. These planes are used as reference locations for the curve fitting procedure (*ℓℓ.* 10-13), as we did with the undistortion map U in Algorithm 4.1.

Algorithm 4.4 Global Correction Map Initial Estimation

```
Input: $({}^{\mathcal{D}}\widehat{\mathsf{C}}_1, {}^{\mathcal{D}}\widehat{\mathsf{C}}_2, \dots, {}^{\mathcal{D}}\widehat{\mathsf{C}}_M)$    ▷ Undistorted point clouds
Input: $(\mathbb{I}_1, \mathbb{I}_2, \dots, \mathbb{I}_M)$    ▷ Wall point indices
Input: $({}^{\mathcal{I}}\mathsf{B}_1, {}^{\mathcal{I}}\mathsf{B}_2, \dots, {}^{\mathcal{I}}\mathsf{B}_M)$    ▷ Checkerboard corners in the images
Output: $\mathsf{G}$    ▷ Global error correction map
Output: ${}^{\mathcal{D}}_{\mathcal{C}}\mathbf{T}$    ▷ Camera-depth sensor transformation matrix
1: for $k \leftarrow 1, 2, \dots, M$ do
2:     ${}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{T} \leftarrow \textsc{solvePnP}(\mathbf{K}_{\mathfrak{C}}, \mathbf{d}_{\mathfrak{C}}, {}^{\mathcal{B}}\mathsf{B}, {}^{\mathcal{I}}\mathsf{B}_k)$
3:     ${}^{\mathcal{C}}\mathsf{B}_k \leftarrow {}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{T} \cdot {}^{\mathcal{B}}\mathsf{B}$
4:     ${}^{\mathcal{C}}\pi_{\mathsf{B}_k} \leftarrow \textsc{fitPlane}({}^{\mathcal{C}}\mathsf{B}_k)$
5:     ${}^{\mathcal{D}}\pi_{\widehat{\mathsf{C}}_k} \leftarrow \textsc{fitPlane}({}^{\mathcal{D}}\widehat{\mathsf{C}}_k, \mathbb{I}_k)$
6: end for
7: $\mathbf{\Pi}_{\mathsf{B}} \leftarrow ({}^{\mathcal{C}}\pi_{\mathsf{B}_1}, {}^{\mathcal{C}}\pi_{\mathsf{B}_2}, \dots, {}^{\mathcal{C}}\pi_{\mathsf{B}_M})$
8: $\mathbf{\Pi}_{\widehat{\mathsf{C}}} \leftarrow ({}^{\mathcal{D}}\pi_{\widehat{\mathsf{C}}_1}, {}^{\mathcal{D}}\pi_{\widehat{\mathsf{C}}_2}, \dots, {}^{\mathcal{D}}\pi_{\widehat{\mathsf{C}}_M})$
9: ${}^{\mathcal{D}}_{\mathcal{C}}\mathbf{T} \leftarrow \textsc{estimateTransform}(\mathbf{\Pi}_{\mathsf{B}}, \mathbf{\Pi}_{\widehat{\mathsf{C}}})$
10: for $k \leftarrow 1, 2, \dots, M$ do
11:     ${}^{\mathcal{D}}\pi_{\mathsf{B}_k} \leftarrow {}^{\mathcal{D}}_{\mathcal{C}}\mathbf{T} \cdot {}^{\mathcal{C}}\pi_{\mathsf{B}_k}$
12:     $(\mathsf{G}, \mathsf{E}) \leftarrow \textsc{updateMap}(\mathsf{G}, \mathsf{E}, {}^{\mathcal{D}}\widehat{\mathsf{C}}_k, \mathbb{I}_k, {}^{\mathcal{D}}\pi_{\mathsf{B}_k})$
13: end for
```

4.5.2 Non-linear Refinement

Once the global correction map G and the camera-depth sensor transformation matrix ${}^{\mathcal{D}}_{\mathcal{C}}\mathbf{T}$ have been estimated, we refine them within a non-linear optimization

framework. To take into account the error on the checkerboard poses estimation, we follow the bundle-adjustment approach as described in [57]: we also refine *all* the checkerboard poses ${}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{T}$, with $k = 1, \ldots, M$.
So, let define ${}^{C}\mathbb{T}_{\mathfrak{B}} \triangleq \left\{ {}^{\mathcal{C}}_{\mathcal{B}_1}\mathbf{T}, {}^{\mathcal{C}}_{\mathcal{B}_2}\mathbf{T}, \ldots, {}^{\mathcal{C}}_{\mathcal{B}_M}\mathbf{T} \right\}$ as the set of checkerboard poses in camera coordinates, estimated with the *solvePnP* function.
Formally, the results of the non-linear refinement is

$$\left(\mathsf{G}, {}^{\mathcal{D}}_{\mathcal{C}}\mathbf{T}, {}^{C}\mathbb{T}_{\mathfrak{B}}\right) = \arg \min_{\mathsf{G}, {}^{\mathcal{D}}_{\mathcal{C}}\mathbf{T}, {}^{C}\mathbb{T}_{\mathfrak{B}}} \sum_{k=1}^{M} e_{\text{repr}}(k) + e_{\text{pos}}(k) \ ,$$

where

$$e_{\text{repr}}(k) \triangleq \sum_{(r,c)\in\mathbb{I}_{\mathfrak{B}}} \frac{1}{\sigma_{\mathfrak{C}}^2} \cdot \left\| \mathrm{r}_{\mathfrak{C}}\left({}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{T} \cdot {}^{\mathcal{B}}\mathsf{B}_k(r,c) \right) - {}^{\mathcal{I}}\mathsf{B}_k(r,c) \right\|^2$$

and

$$e_{\text{pos}}(k) \triangleq \sum_{(u,v)\in\mathbb{I}_k} \frac{1}{|\mathbb{I}_k| \cdot \sigma_{\mathsf{U}}^2(z)} \cdot \left\| \mathrm{p}_{{}^{\mathcal{D}}\pi_k}\left(\mathrm{g}_{u,v}\left({}^{\mathcal{D}}\widehat{\mathsf{C}}_k(u,v) \right) \right) - \mathrm{g}_{u,v}\left({}^{\mathcal{D}}\widehat{\mathsf{C}}_k(u,v) \right) \right\|^2 \ .$$

Here e_{repr} takes into account the reprojection error of the checkerboard corners onto the images and depends on the checkerboard poses only. The residuals are weighted by the inverse of the variance of the corner estimation error $\sigma_{\mathfrak{C}}^2$, where $\sigma_{\mathfrak{C}} = 0.2$.

e_{pos} represents the error between the planes defined by the checkerboards and the ones defined by the undistorted point clouds. Formally speaking, this error is the distance between the cloud point ${}^{\mathcal{D}}\widehat{\mathsf{C}}_k(u,v)$ corrected with the current estimation of G and its line-of-sight projection onto the plane ${}^{\mathcal{D}}\pi_k$ defined by the checkerboard corner set ${}^{\mathcal{D}}\mathsf{B}_k$. Such set is computed as

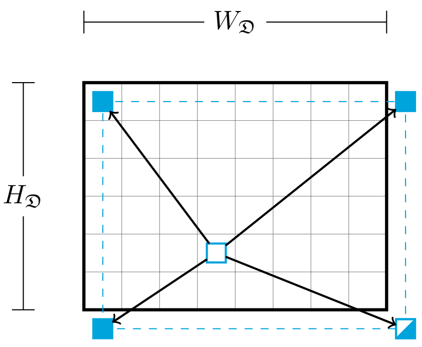
$${}^{\mathcal{D}}\mathsf{B}_k = {}^{\mathcal{D}}_{\mathcal{C}}\mathbf{T} \cdot {}^{\mathcal{C}}_{\mathcal{B}_k}\mathbf{T} \cdot {}^{\mathcal{B}}\mathsf{B} \ .$$

Finally, each residual is weighted by the variance on the depth measurements *after* the undistortion phase $\sigma_{\mathsf{U}}^2(z)$, multiplied by the number of wall points, i.e. $|\mathbb{I}_k|$.

4.5.3 Implementation Details

Global Correction Map

As mentioned before, the global correction map G is someway similar to the undistortion map U, but more simple. In fact, G needs to transform planes into planes and, recalling that a plane transformation has 3 degrees of freedom, we just need 3 functions to satisfy the requirements. So, we end up with a discretized map constructed as U but with $\chi_{\mathsf{G}} = W_{\mathfrak{D}}$ and $\psi_{\mathsf{G}} = H_{\mathfrak{D}}$, that is, only 4 pixels



■ Pixel $(u,v)^{\mathsf{T}} \in \mathbb{S}_{\mathsf{G}}$. For this pixel a correction function $\mathrm{g}_{u,v}(\cdot)$ has been estimated.

◪ Pixel $(u,v)^{\mathsf{T}} \in \mathbb{S}_{\mathsf{G}}$. To guarantee planarity, for this pixel the correction function $\mathrm{g}_{u,v}(\cdot)$ has been computed starting from the ones of the other 3 pixels in $\mathbb{S}_{\mathsf{G}}$.

□ Pixel $(u,v)^{\mathsf{T}} \in \mathbb{I}_{\mathfrak{D}}$. For this pixel the correction function $\mathrm{g}_{u,v}(\cdot)$ is a linear combination of the functions of the pixels in $\mathsf{S}_{\mathsf{G}}(u,v)$.

→ Connection from a pixel $(u,v)^{\mathsf{T}} \in \mathbb{I}_{\mathfrak{D}}$ to one in $\mathsf{S}_{\mathsf{G}}(u,v)$.

Figure 4.7: Visualization of a global correction map G. A global correction function is estimated for all and only the pixels in $\mathbb{S}_{\mathsf{G}}$. For all the others, instead, the function is computed as a linear combination of the ones estimated for the pixels in $\mathbb{S}_{\mathsf{G}}$.

contain a correction function $\mathrm{g}_{u,v}(\cdot)$, for all the others, instead, such function is computed as a linear combination of the ones of those 4 pixels. Actually, allowing 4 pixels to control the whole map usually leads to wrong results. Let suppose, for example that 3 of such pixels contain an identity function and the fourth does not. Clearly the resulting surface will not be a plane anymore. For this reason, only 3 of these pixels are actually computed, the fourth is instead estimated exploiting the following invariant:

$$\frac{\mathrm{g}_{0,0}(d) + \mathrm{g}_{W_{\mathfrak{D}},H_{\mathfrak{D}}}(d)}{2} = \frac{\mathrm{g}_{W_{\mathfrak{D}},0}(d) + \mathrm{g}_{0,H_{\mathfrak{D}}}(d)}{2} \quad . \tag{4.4}$$

Suppose now that $\mathrm{g}_{W_{\mathfrak{D}},H_{\mathfrak{D}}}(\cdot)$ is the dependent function: $\mathrm{g}_{W_{\mathfrak{D}},H_{\mathfrak{D}}}(\cdot)$ can be estimated by fitting a function on an adequate set of pairs $(d, \mathrm{g}_{W_{\mathfrak{D}},H_{\mathfrak{D}}}(d))$, where

$$\mathrm{g}_{W_{\mathfrak{D}},H_{\mathfrak{D}}}(d) = \mathrm{g}_{W_{\mathfrak{D}},0}(d) + \mathrm{g}_{0,H_{\mathfrak{D}}}(d) - \mathrm{g}_{0,0}(d)$$

is computed from (4.4). An example of the presented global correction map G is visible in Figure 4.7.

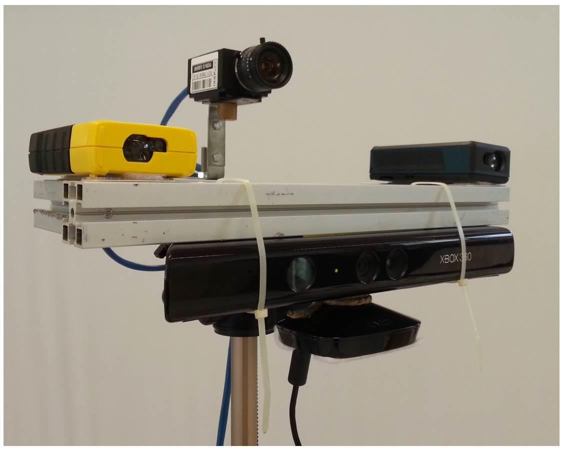


Figure 4.8: Structure we set up to acquire the data for both the test set and training set. The two laser meters are located on the left and right of the structure to guarantee that the sensor is correctly aligned.

Curve Fitting

Since the correction map G is constructed in the same way as the undistortion map U, the considerations on the curve fitting procedure made in Section 4.4.3 are still valid also in the global error case.

4.6 Experimental Evaluation

4.6.1 Setup

We validated the accuracy of our calibration method in a real-world scenario. In particular we performed the calibration of two Microsoft Kinect (in the following called KINECT47A and KINECT51A) and one Asus Xtion Pro Live (in the following called ASUS) depth sensor.

We calibrated the RGB camera of each device using the calibration tool provided by the ROS libraries [42]: a good RGB camera calibration is an essential requirement to have valuable results. Then, we mounted each sensor, one at a time, on a structure composed by a tripod, two laser meters and a high resolution PointGrey camera (see Figure 4.8). Finally, we attached a checkerboard on a big wall, collecting two datasets for each device: a *training set* and a *test set*.

The training set contains views of the checkerboard from the device camera, the depth sensor and the high resolution camera from different locations and orientation (e.g., see Figure 4.9). The test set, instead, has been acquired by positioning the structure orthogonal to the wall at different distances and measuring such distances with the two laser distance meters. An example of the data acquired for the test sets is visible in Figure 4.10, the correspondent distances are reported in Table 4.1 .

4.6.2 Undistortion Map

To evaluate the performances of our undistortion approach, we introduce a metric called *planarity error*. For each cloud of the test set, we extract the wall point indices $\mathbf{I}$ from its undistorted version as described in Section 4.4.1. Then, we define the planarity error as

$$e_{\text{plan}} = \sqrt{\frac{1}{|\mathbb{I}|} \sum_{(u,v)\in\mathbb{I}} \|\mathbf{n}^{\mathsf{T}}\mathsf{C}(u,v) - d\|^2} \ ,$$

where C is a generic point cloud of the test set (we consider both the original and the undistorted versions) and π is the plane with equation $\mathbf{n}^{\mathsf{T}}\mathbf{x} - d = 0$ fitted to the wall points with indices in $\mathbb{I}$.

Undistortion Map Functions

In the previous sections, we have always talked about "undistortion functions" without providing many details about the nature of these functions. Actually, analyzing the error on the plane estimation described in Section 4.2, and especially the one in Figure 4.3, we evinced that such error is well described by a *quadratic polynomial*.

Such hypothesis is further confirmed by the analysis of the sample sets used to generate the undistortion map. Recalling that the sample sets are composed by pairs (d, d_π), where d is the depth measurement provided by the sensor and d_π is the depth value that the point should have to lay on the plane fitted to the cloud, a comparison between quadratic and linear functions is reported in Figure 4.11. In all our tests the quadratic functions outperformed the linear ones. We also tried to fit higher degree polynomials to the sample sets, unfortunately, we did not obtain any improvement over quadratic functions. In fact, as reported in Figure 4.12, the fitted polynomials overlap. Actually, the difference is out of the training set region: the 4$^{\text{th}}$ degree polynomials tend to overfit the training data and therefore perform worse than lower degree polynomials.

To prove that our hypotheses are correct, we further evaluate the polynomial functions from an undistortion quality point of view. For each cloud in the test set, we computed the planarity error introduced above. The plot in Figure 4.13 clearly shows that all the super-linear functions provide better undistortion results

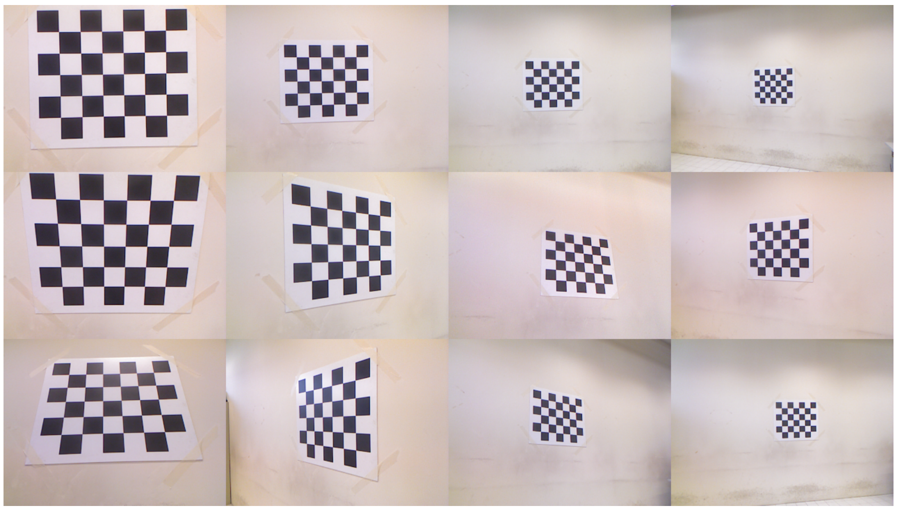


Figure 4.9: Some images of the training set of KINECT47A. The checkerboard is framed from different locations and orientations from 1 to about 4 meters.

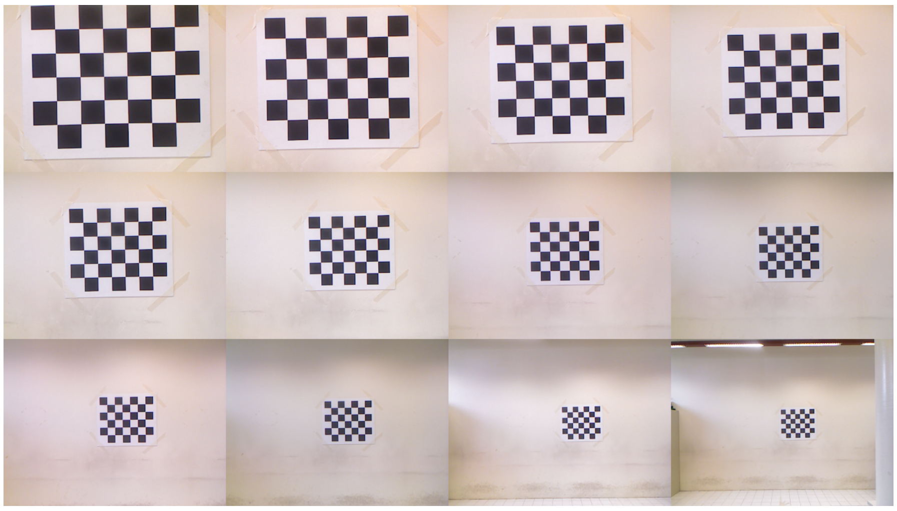


Figure 4.10: Some images of the test set of KINECT47A. The checkerboard is framed from different locations from 1 to about 4.5 meters, paying attention to always have a zero rotation with respect to the wall plane.

Table 4.1: Distances from the sensor to the wall for the images of the test set reported in Figure 4.10.

Test set wall distances			
0.943 m	1.099 m	1.241 m	1.413 m
1.615 m	1.935 m	2.261 m	2.595 m
2.906 m	3.292 m	4.079 m	4.635 m

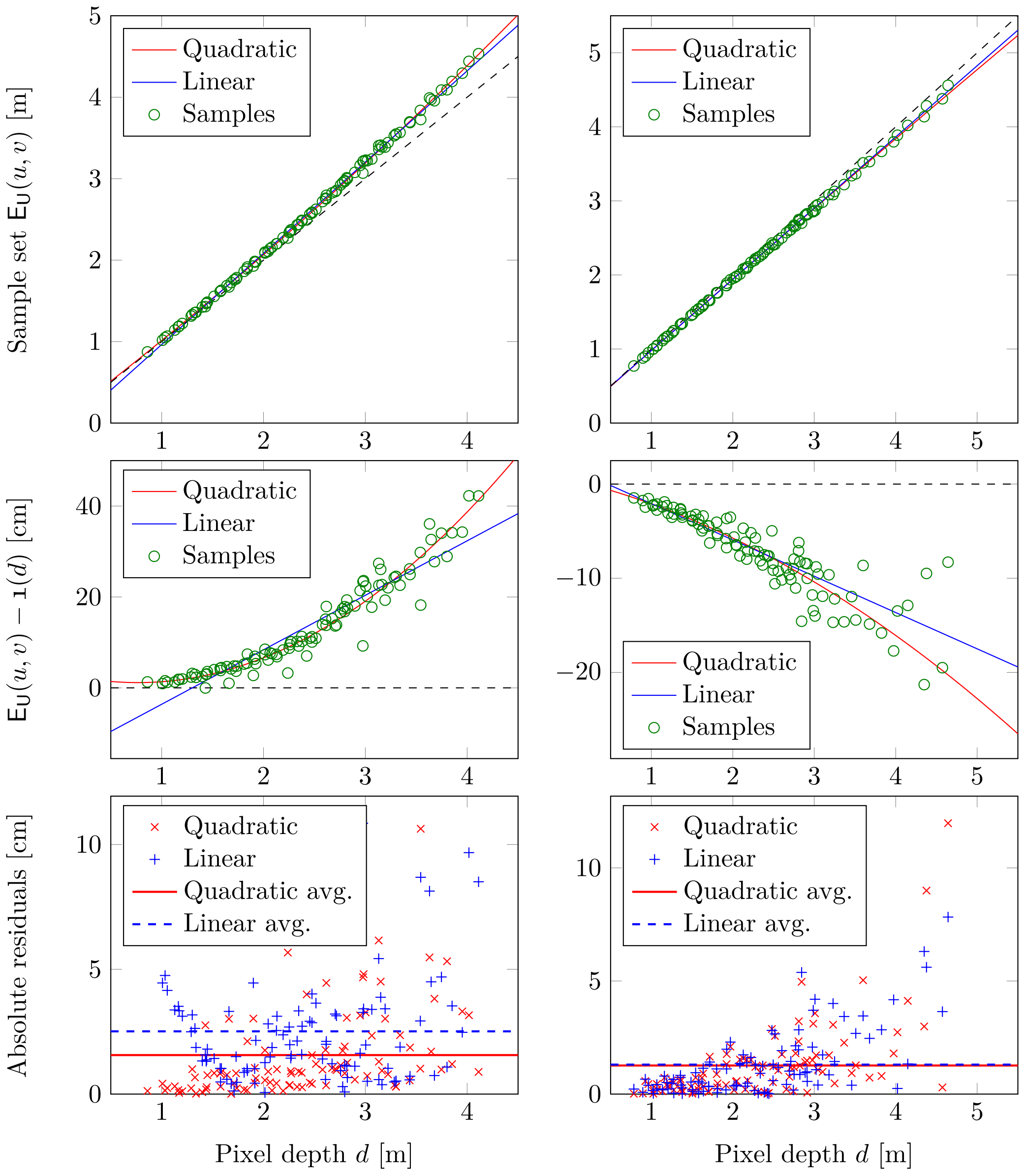


Figure 4.11: Analysis of the sample set distribution for two different pixels. The sample set composed by pairs (d, d_π), where d is the sensor-provided depth while d_π is the depth the point should have to lay on the plane fitted to the whole cloud. The first line shows the samples as well as a linear and a quadratic function that best approximate the samples. The second line, instead, shows the difference between the y and x sample values. Finally, in the third line, the residuals for the two functions are reported.

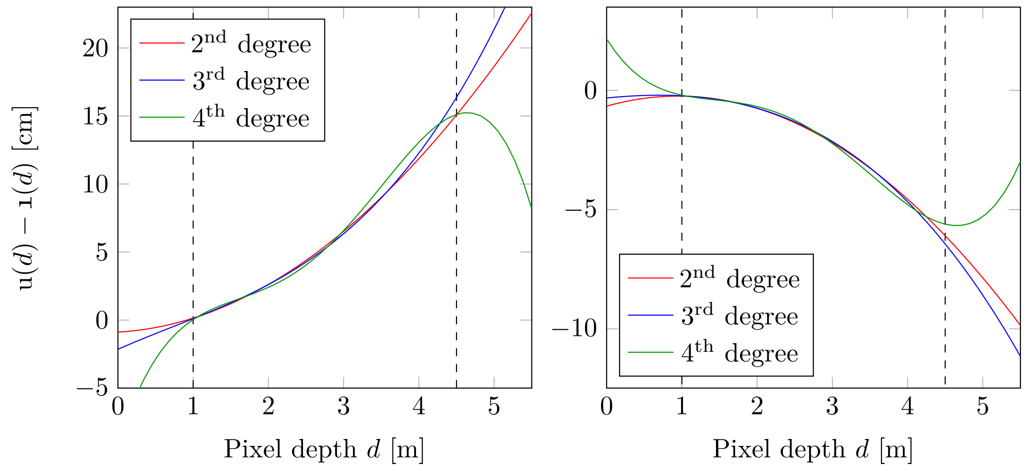


Figure 4.12: Undistortion polynomial $\mathrm{u}(d)$ for two different pixels estimated varying the maximum degree (from 2 to 4). For each polynomial, to better show the differences among them, the difference with the identity function $\mathrm{u}(d) - \imath(d)$ is plotted. The two vertical, dashed lines, delimit the training data depth range (1-4.5 m). Analyzing the images, it is clear that the 4th degree polynomials are not suitable for the map, since they are likely to overfit the training data.

when compared to the linear functions. Therefore, all the tests presented in following sections have been performed using quadratic undistortion functions: in our experience, this class of functions provides good undistortion results without presenting overfitting problems.

Map Discretization

To select the most appropriate bin size values (i.e., χ_{U} and ψ_{U}, described in Section 4.4.3), we evaluated the planarity error varying the two parameters. The results are reported in Figure 4.14. Differently from what we expected, such parameters do not affect so much the results. Actually, up to a 8×8 pixels size, the planarity error is almost identical. Only with greater sizes, starting from 16×16, the error increases, especially close to the image corners. Even evaluating the planes that result from the undistortion of a real cloud (Figure 4.15), there is not much difference between them. Finally, a comparison between the generated maps is reported in Figure 4.16. Looking at the maps in Figure 4.16, computed associating to each pixel $(u, v)^\mathsf{T}$ the value $\mathrm{u}_{u,v}(d) - d$ for a given d, we immediately notice that the maps becomes smoother as the bin size increases.

In our experience, we found that a bin size of 4×4 pixels represents a good trade-off between computational efficiency and robustness. Actually, as mentioned

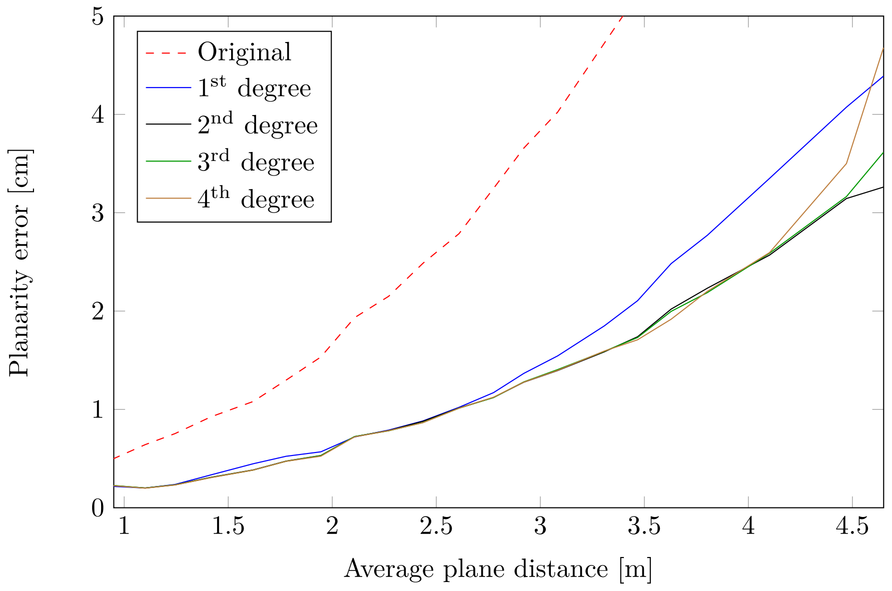


Figure 4.13: Planarity error when varying the degree of the undistortion polynomials. From the plot we can see that linear functions are not able to correctly model the distortion introduced by the sensor. On the other hand, quartic functions tend instead to overfit the training data (e.g., in the right part of the plot the error of the quartic functions increases w.r.t. the errors of the quadratic and cubic functions).

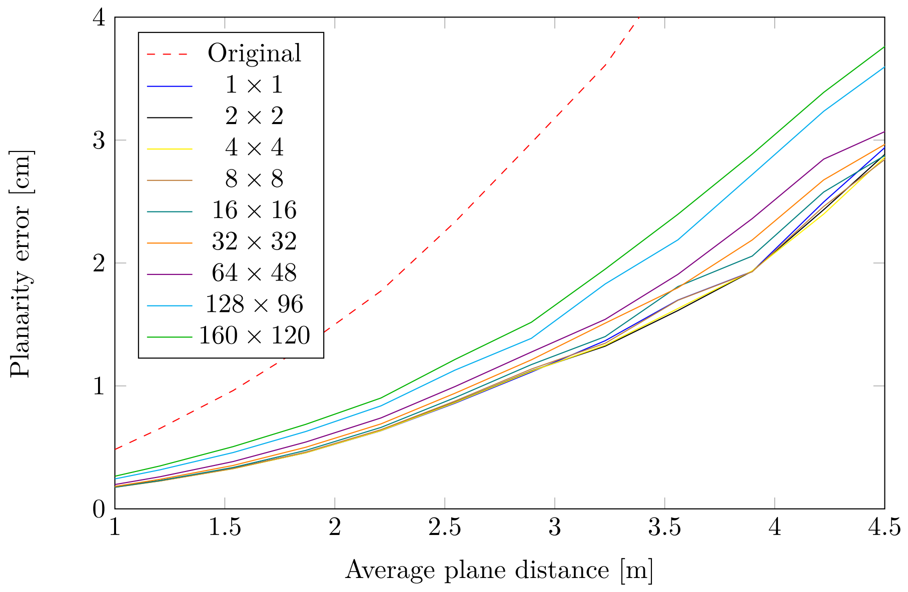


Figure 4.14: Planarity error of the wall points when changing the bin size. In the plot it is visible that increasing the bin size also the error increases, but not as much as expected. Actually, only with bin sizes staring from 16 × 16 the error increase starts being non-negligible.

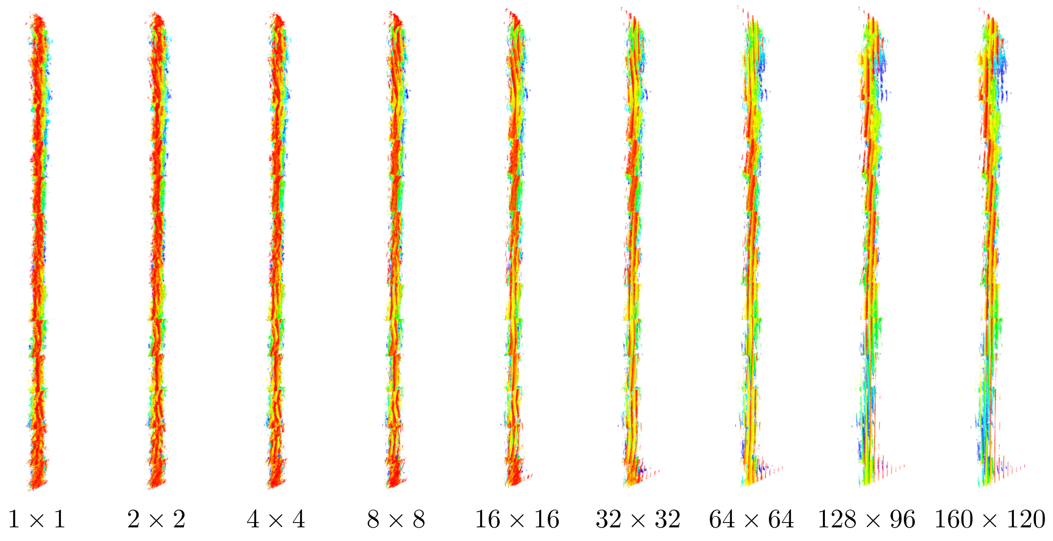


Figure 4.15: Top-view of the cloud of a planar surface undistorted using maps with different bin sizes. Note how the resulting clouds are similar, especially the four on the left.

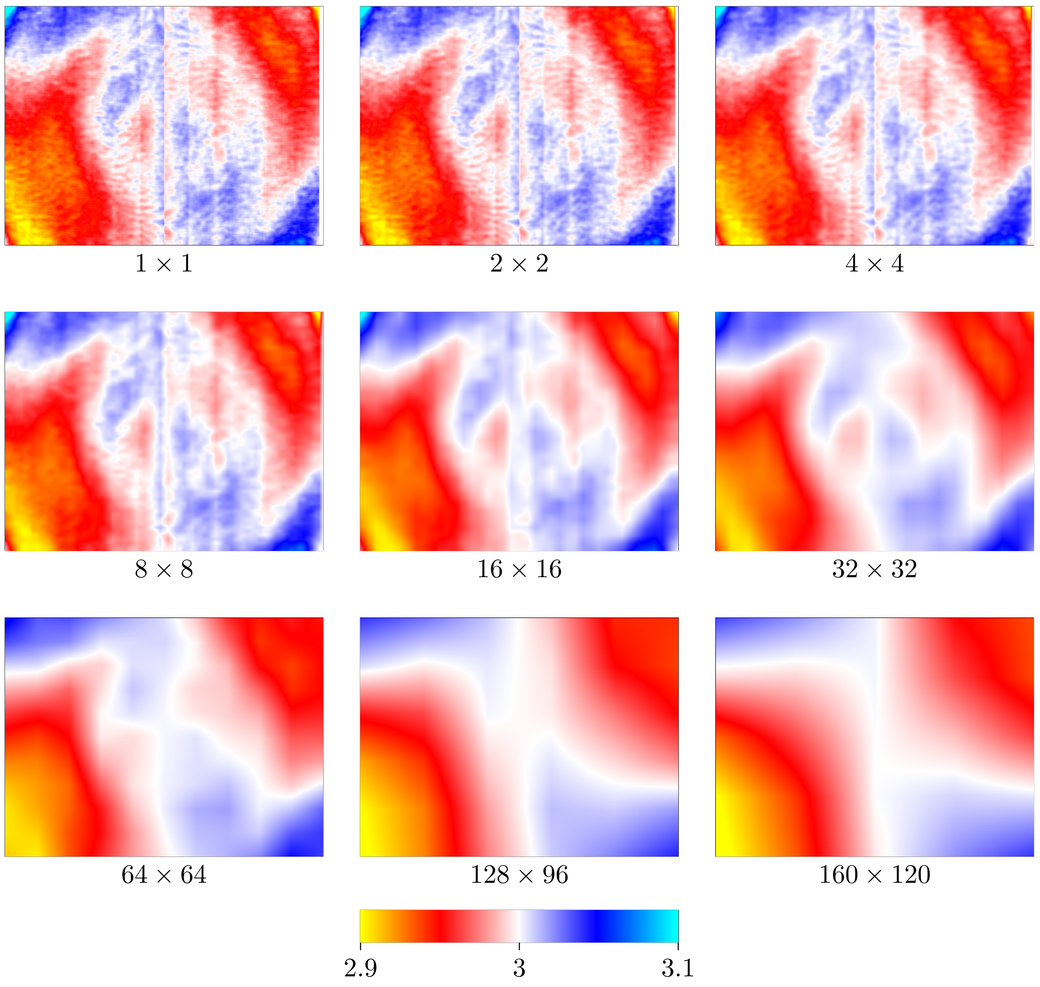


Figure 4.16: Undistortion map computed using different bin size, evaluated at a distance of 3 meters.

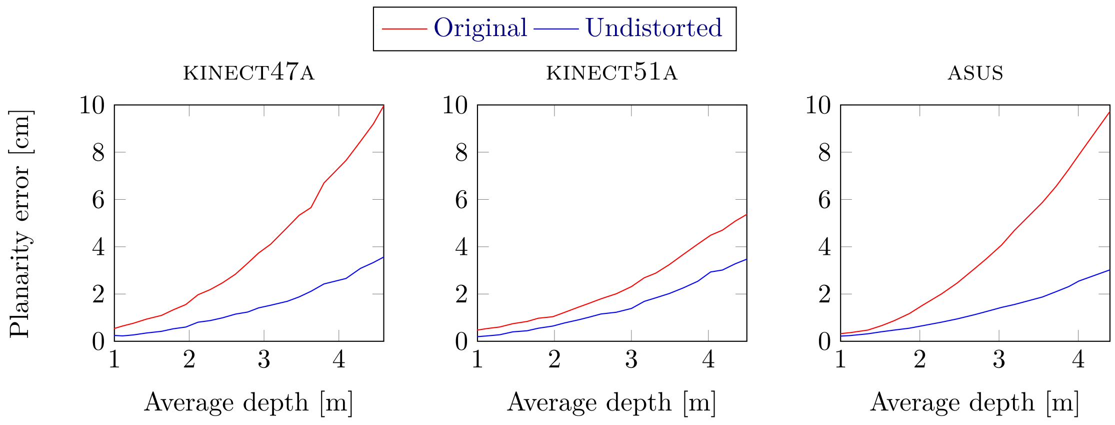


Figure 4.17: Planarity error for the three tested sensors. As visible, the proposed approach is able to drastically reduce the distance of the measured points from the plane that best fits the data. Actually, the error curve after the undistortion phase is mainly due to noise.

before, larger bins tend to fail close to the image corners, while in other experiments we noticed that smaller bins tend to perform badly with small calibration datasets because of the lack of data for some pixels.

Test Set Results

We finally tested our algorithm against the test sets acquired for the three sensors. Results of the planarity error evaluation are reported in Figure 4.17. As expected, the proposed method permits to drastically improve the planarity of the depth data generated by the calibrated sensor. Looking at the plots one could argue: why isn't the error after the undistortion closer to zero? Actually, to the best of our knowledge, the error curve calculated after the undistortion is mainly due to the sensor noise and the quantization error. Therefore it is not possible to further reduce the error.

In Figure 4.18 the estimated undistortion maps, evaluated at 4 defined depths, are shown. Looking at the scales, we can see that the magnitude of the correction is consistent with the planarity error of the original data. Then, in Figure 4.19 the results of the algorithm applied to the clouds of Figure 4.1 are reported. As expected, the clouds are now planar but they are not in the correct positions yet.

4.6.3 Global Correction Map

In this section we report some results of the evaluation of the estimated global correction map G. Recalling that the map is computed using the checkerboard as

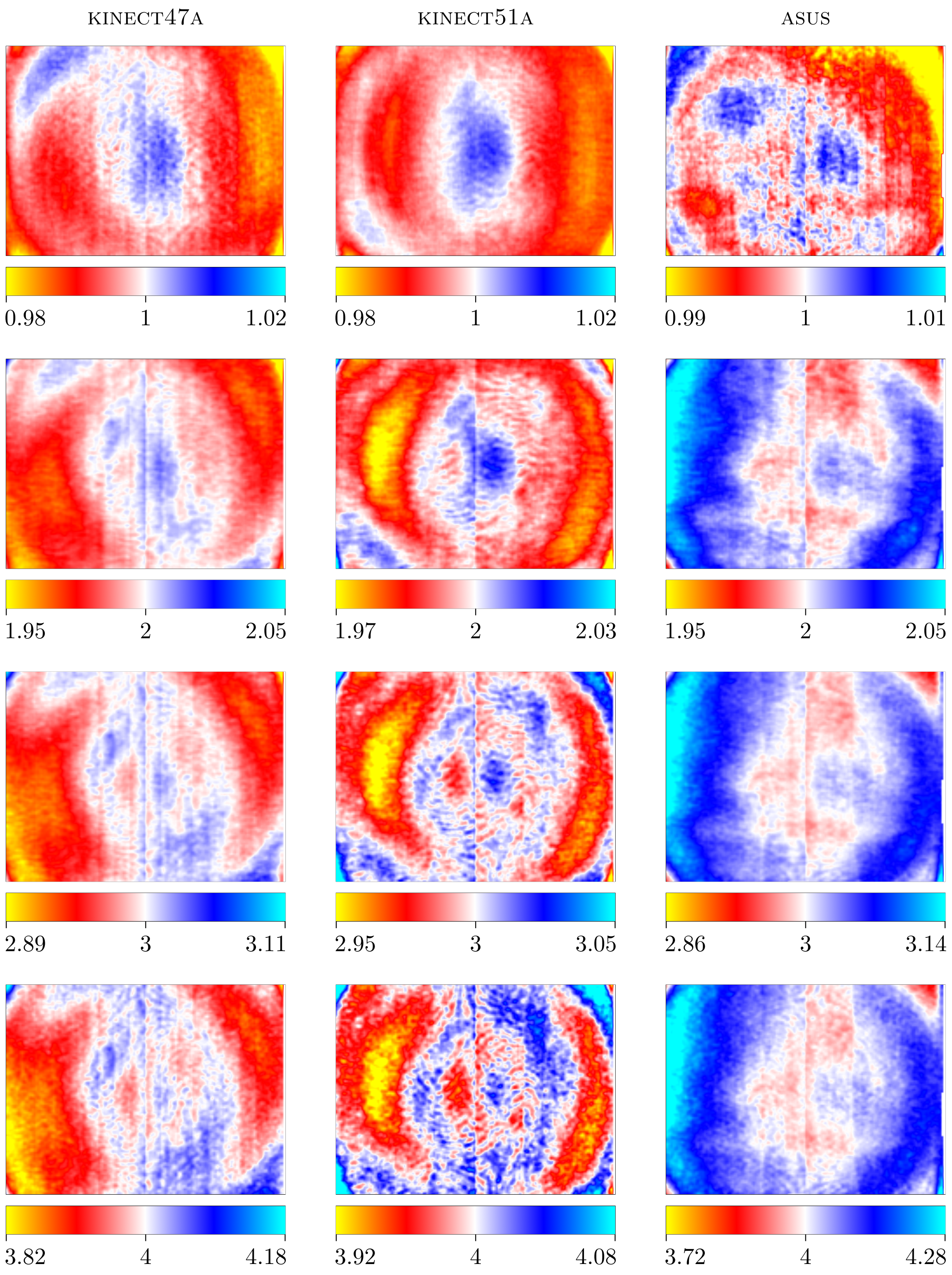
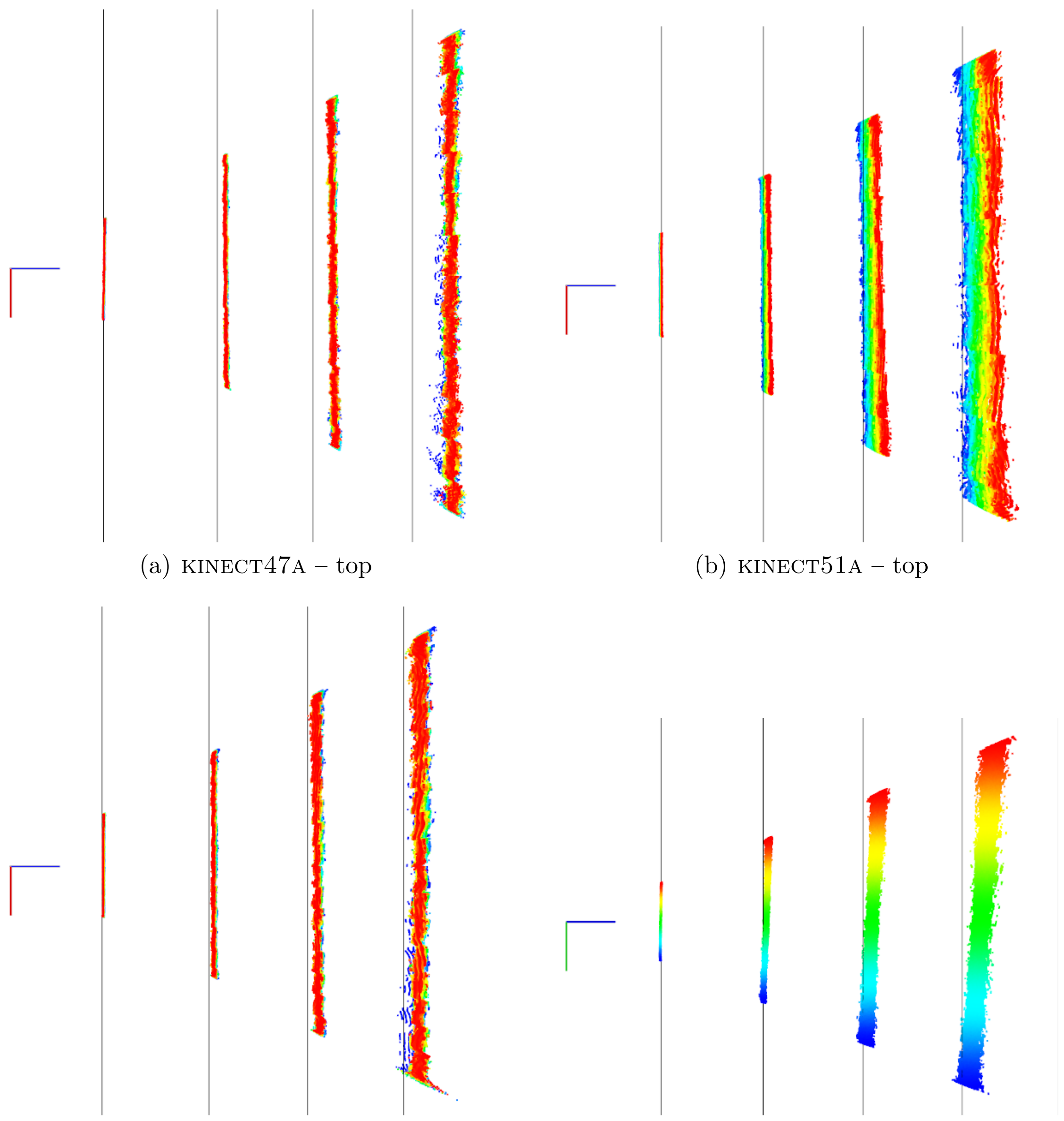


Figure 4.18: Maps generated for the three sensors. For each sensor the evaluation of the respective map at 1, 2, 3, and 4 meters is reported. Note that each map has its own scale and all values are in meters.


(a) KINECT47A – top (b) KINECT51A – top

(c) ASUS – top (d) KINECT51A – side

Figure 4.19: Top and side views of the point clouds of Figure 4.1 after the undistortion phase. Again, the gray lines show the depth measured by means of the laser meters. As we can see, the clouds are now more planar than the original ones, however, they are not in the right position, not even correctly oriented.

the reference plane and that the map and the RGB camera-depth sensor transformation are refined together, also the estimated transformation is taken into account to evaluate the results. We estimate the error of the plane that results after the global correction with respect to the plane defined by the checkerboard.

Firstly, the pose of the checkerboard with respect to the camera is estiamted using the corners extracted from the image. Secondly, the checkerboard plane is transformed into depth sensor coordinates. Finally, the average distance of the wall points (extracted from the cloud) to the checkerboard-defined plane is computed.

Global Correction Map Functions

Before evaluating the global correction map G, as we did for the undistortion map U, we analyze the sample sets generated to compute the map, i.e. E_G, just to evince the most appropriate function type to fit to the data. A first analysis of the error has been reported in Section 4.2, in Figure 4.2. Such error was computed by evaluating the difference between the average depth of a distorted point cloud and the measurements provided by a laser distance meter. Our sample sets, instead, contain pairs $(\widehat{d}, d_{\pi_\mathsf{B}})$, where $\widehat{d}$ is the depth value after the undistortion phase, and d_{π_B} is the expected depth, i.e. the depth of the pixel if it were laying on checkerboard-defined plane π_B. Thus, a novel investigation has to be performed.

Also in this case we tried to fit polynomial functions with a different maximum degree and evaluated which ones better represent the data. In Figure 4.20 we report a comparison between quadratic and linear functions for two of the four sample sets used to generate the global correction map. From the figure, it is clear that linear functions are not suitable since they do not fit properly to the data. A further confirmation of this fact is visible in Figure 4.21, where the results of the calibration process varying the maximum degree of the polynomials from 1 to 4 is reported.

Moving a cloud along the z-axis is the same as moving one of the sensors along the same axis. For this reason, to avoid problems in the optimization phase (i.e. one parameter can be increased and the other decreased of the same quantity and the error evaluation does not change), for these polynomials the constant factor is set to zero.

All the calibrations reported in the following were performed treating each $\mathrm{g}_{u,v}(\cdot)$ as a quadratic function with the constant factor equal to zero.

Test Set Results

We finally estimated the global correction results on the acquired test set. For each test cloud we evaluated the distances of the points of the main plane to

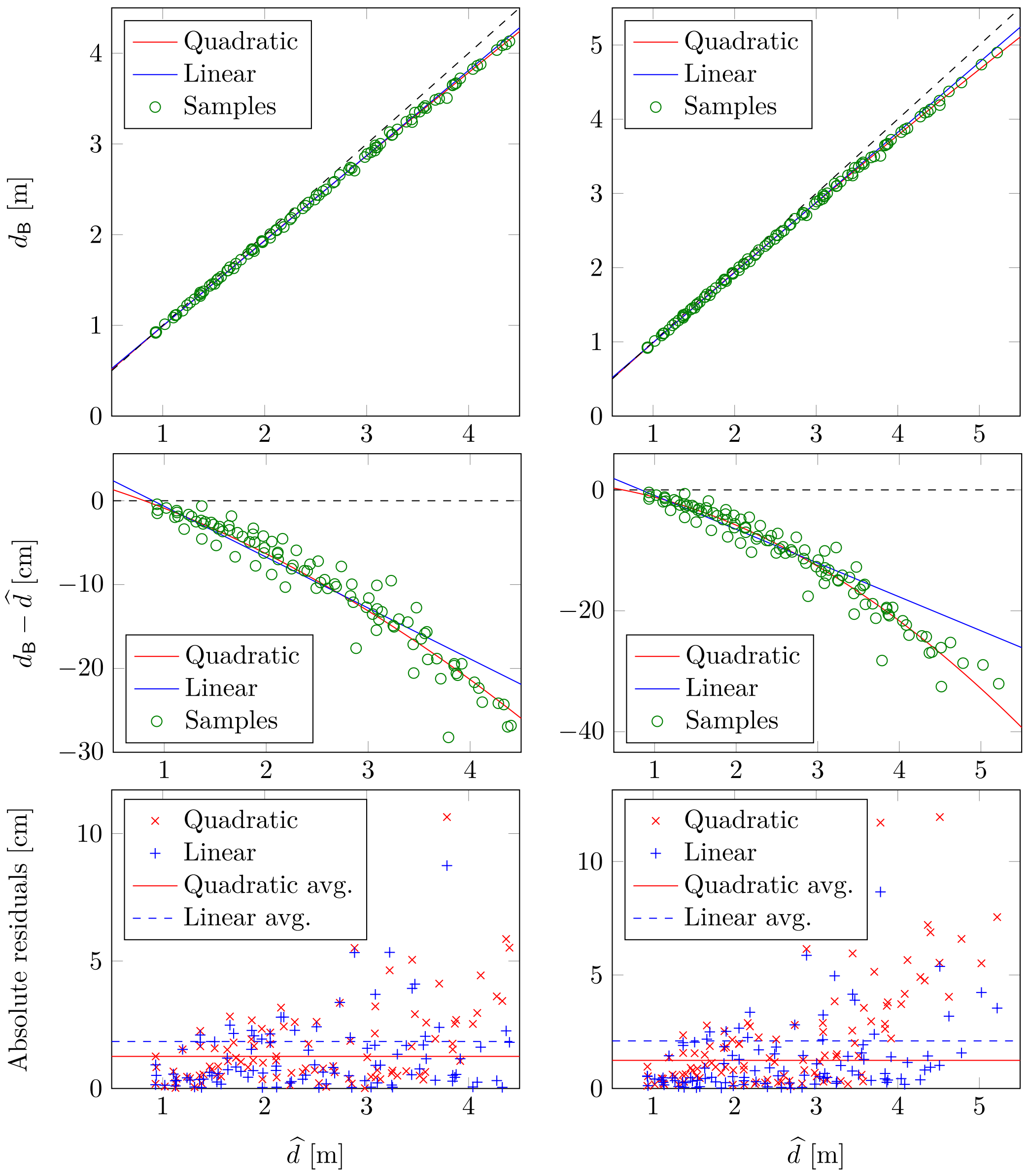


Figure 4.20: Analysis of two of the sample sets used to estimate the global correction map G. The sample set is composed by pairs $(x, y) = (\widehat{d}, d_{\mathsf{B}})$, where $\widehat{d}$ is the sensor-provided depth while d_{B} is the depth defined by the checkerboard $\mathfrak{B}$ attached on the wall. The plots at the top show the samples as well as a linear and a quadratic function that best approximate them. The plots in the middle, instead, show the difference between the samples' y and x values, i.e. $d_{\mathsf{B}} - \widehat{d}$. Finally, in the bottom plots, the residuals for the two functions are reported.

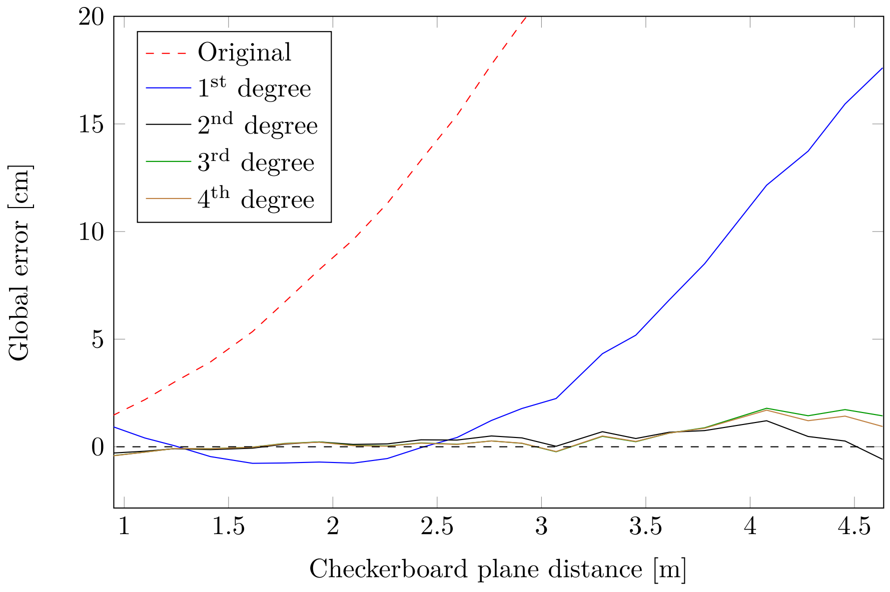


Figure 4.21: Global error when varying the maximum degree of the global correction polynomials. From the plot we can see that linear functions are not able to model (and correct) the error on the average depth estimation.

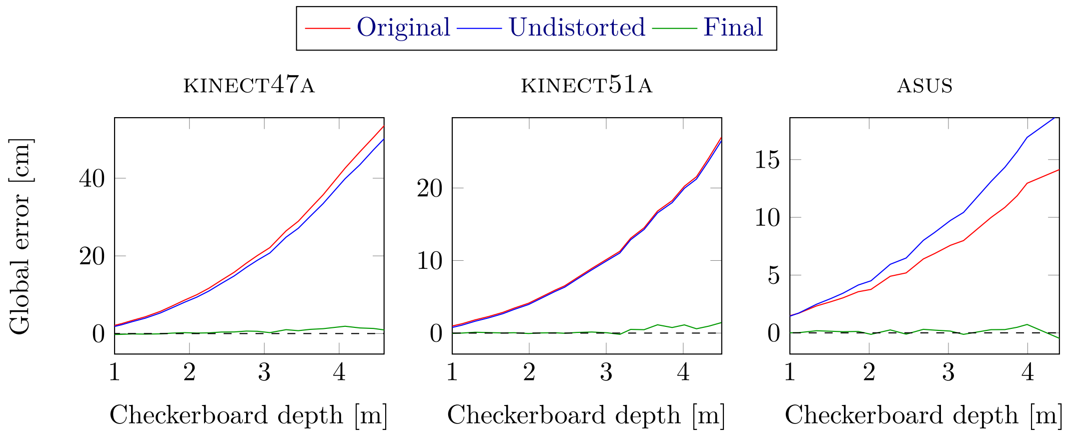
the plane defined by the checkerboard and computed their mean. The plots in Figure 4.22 show the results of such evaluation. The proposed error correction approach is working as expected: all the points are correctly translated to the right location, with respect to the checkerboard pose.

4.6.4 Final Results

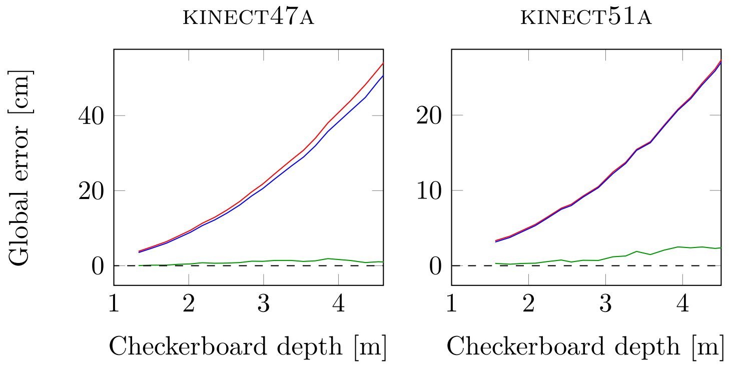
The last set of tests are meant to evaluate the results of the proposed calibration approach when dealing with real world data. To this aim, we first compare the wall average depths obtained after the calibration with the measurements given by the laser meters, then the transformations between the depth sensors and the cameras are evaluated in terms of visual results and expected values.

Depth Calibration

The first plots we are going to analyze are those in Figure 4.23. They show a quantitative evaluation of the depth error, i.e. the distance between the wall depth measured by means of the laser meters and the average depth of the points after both the undistortion and global correction phase. As clearly visible, in all the tests, the resulting planes are within a couple of centimeters from the real ones.



(a) Calibration using the device camera.



(b) Calibration using the external, high resolution camera.

Figure 4.22: Global error for the three tested depth sensors. The error is computed for the original point cloud (Original), the cloud after the undistortion phase (Undistorted), and the cloud after both the undistortion and the global error correction (Final). Moreover, to further assess the validity of the proposed approach, the results of the calibration procedure when using the device camera (a) and the external high resolution camera (b) are reported.

Table 4.2: Camera-depth sensor transform estimated for the three tested sensors when calibrated using the device camera. Both the translation $\mathbf{t} = (t_x, t_y, t_z)^\mathsf{T}$ and the rotation, represented as a quaternion $\mathbf{q} = (q_w, q_x, q_y, q_z)^\mathsf{T}$, are reported. The FACTORY line contains the factory-provided calibration parameters for the three devices.

	t_x [m]	t_y [m]	t_z [m]	q_x	q_y	q_z	q_w
FACTORY	0.025	0	0	0	0	0	1
KINECT47A	0.0237	0.0044	-0.0063	0.0034	0.0060	-0.0017	0.9999
KINECT51A	0.0276	0.0024	-0.0036	0.0025	0.0007	-0.0010	0.9999
ASUS	0.0294	-0.0040	-0.0011	0.0048	0.0059	-0.0004	0.9999

These good results confirm the soundness of our choices.

We also evaluated how much the the resulting plane is rotated with respect to the real one. To this aim we computed the angle between the normal of the plane fitted to the corrected data, and the x- and y-axis of the wall plane, i.e. $(1, 0, 0)^\mathsf{T}$ and $(0, 1, 0)^\mathsf{T}$ respectively. Let $\mathbf{n}$ be the fitted-plane normal and let $\mathbf{a}$ be the axis with respect to which the error is computed, the rotation error, e_rot, is

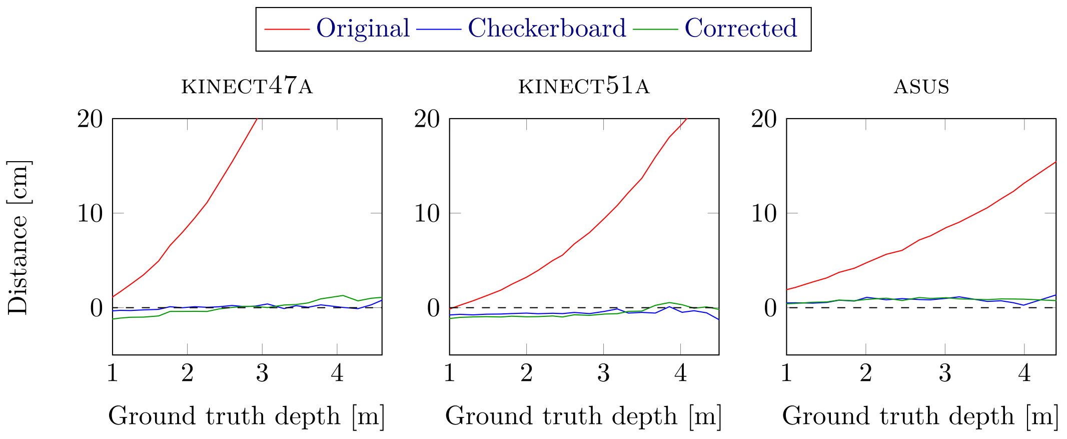
$$e_\text{rot} = \arccos(\mathbf{n}^\mathsf{T}\mathbf{a}) - \frac{\pi}{2} \ .$$

Results of the error computation for sensor KINECT51A are reported in Figure 4.24. The figure shows that the rotation about the x-axis is completely corrected. For what concerns the rotation about the y-axis, instead, the results are worse. The reason for this fact is likely to be the error on the real depth estimation. In fact, a difference of about 2 mm in the depth measures (note that this is the nominal error of the two laser meters) leads to a rotation of about 0.5°.

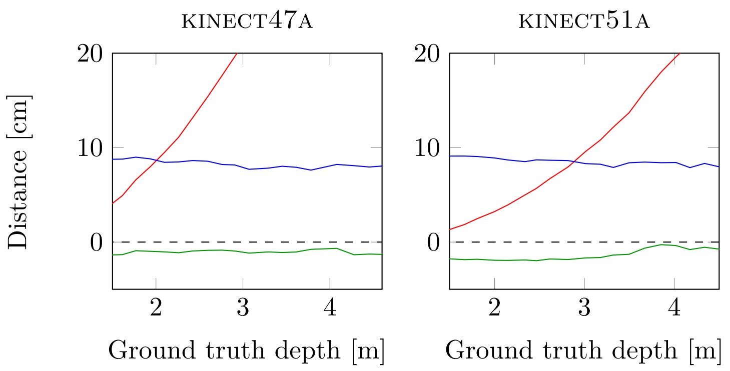
A further confirmation that the proposed approach works properly, is shown in Figure 4.25. The pictures report the clouds of Figure 4.1 after both the undistortion phase (see Figure 4.19) and the global error correction. As expected, all the clouds are now both planar and located correctly.

Camera-Depth Sensor Transform

Even if the camera-depth sensor transform ${}^{\mathcal{D}}_{\mathcal{C}}\mathbf{T}$ estimated during the optimization phase is a sort of "side effect" of the depth calibration, a good transformation can be seen as a proof of the validity of the proposed approach. In Table 4.2 the transformations that resulted from the calibration of the sensors using their device cameras, are reported. Moreover, to give a comparison metric, also the factory-provided transformation is reported. The values obtained are similar to those obtainable with other state-of-the-art calibration tools for RGB-D devices [24, 52, 53, 67].



(a) Results of the calibration using the device camera.



(b) Results of the calibration using the external, high resolution camera.

Figure 4.23: Distance between the real wall depth and the one estimated with the calibration procedure. The error is computed for the original point clouds (Original) and for the clouds after both the undistortion and the global error correction (Corrected). Moreover, the distance of the wall from the color sensor estimated using the checkerboard (Checkerboard) is reported. The error is computed using the device camera (a) as the reference camera as well as using the external high resolution one (b). Note that there is a fixed offset of less than 1 cm between the laser meters and the two Kinects (KINECT47A and KINECT51A are closer to the wall) and about 9 cm between the laser meters and the high resolution camera (the camera is farther).

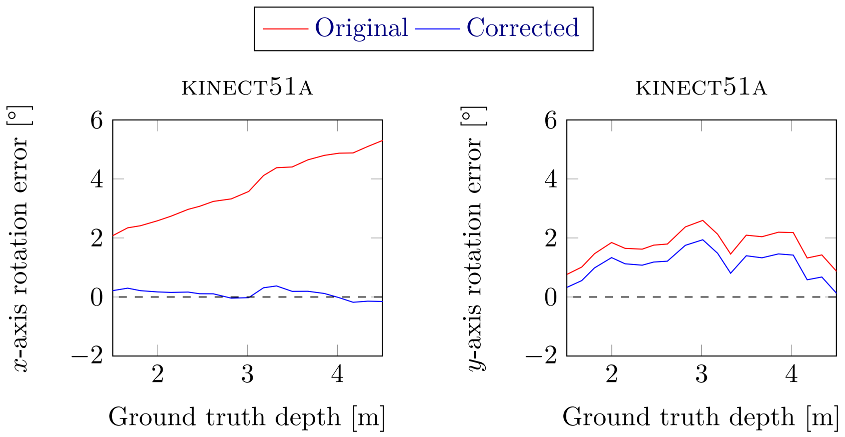


Figure 4.24: Rotation error for KINECT51A sensor. Both the rotation about the x- and y-axis are compared. The error is the angle estimated between the normal of the plane fitted to the corrected data and the theoretical wall plane x-axis and y-axis.

Finally, the visual results of this camera-depth sensor calibration are shown in Figure 4.26. Looking at the images, we can see that the data fusion obtained with the transformation computed during the calibration process are definitely better than the ones obtained with the default transformation.

4.7 ROS Package

The proposed algorithm has been implemented in C++ within the robotics framework ROS [42]. To perform the calibration, the user is asked to capture a training set using a tool provided in the package and then process the data using a different executable. For what concerns the training data collection, it is sufficient to attach a checkerboard on a large wall and get 50 to 100 images (for both the camera and the depth sensor) from different locations and orientations. Once executed, the program outputs 3 files. They contain, respectively:

- the undistortion map U;
- the global correction map G;
- the rigid displacement ${}^{\mathcal{D}}_{\mathcal{C}}\mathbf{T}$ between the two sensors.

Also a ROS node (and its threaded version, i.e. a ROS nodelet) to correct an input cloud provided by the depth sensor has been implemented. This node can be inserted transparently in a ROS pipeline between the producer node (i.e. the sensor) and the consumer node (i.e. the user program). In this way the user program does

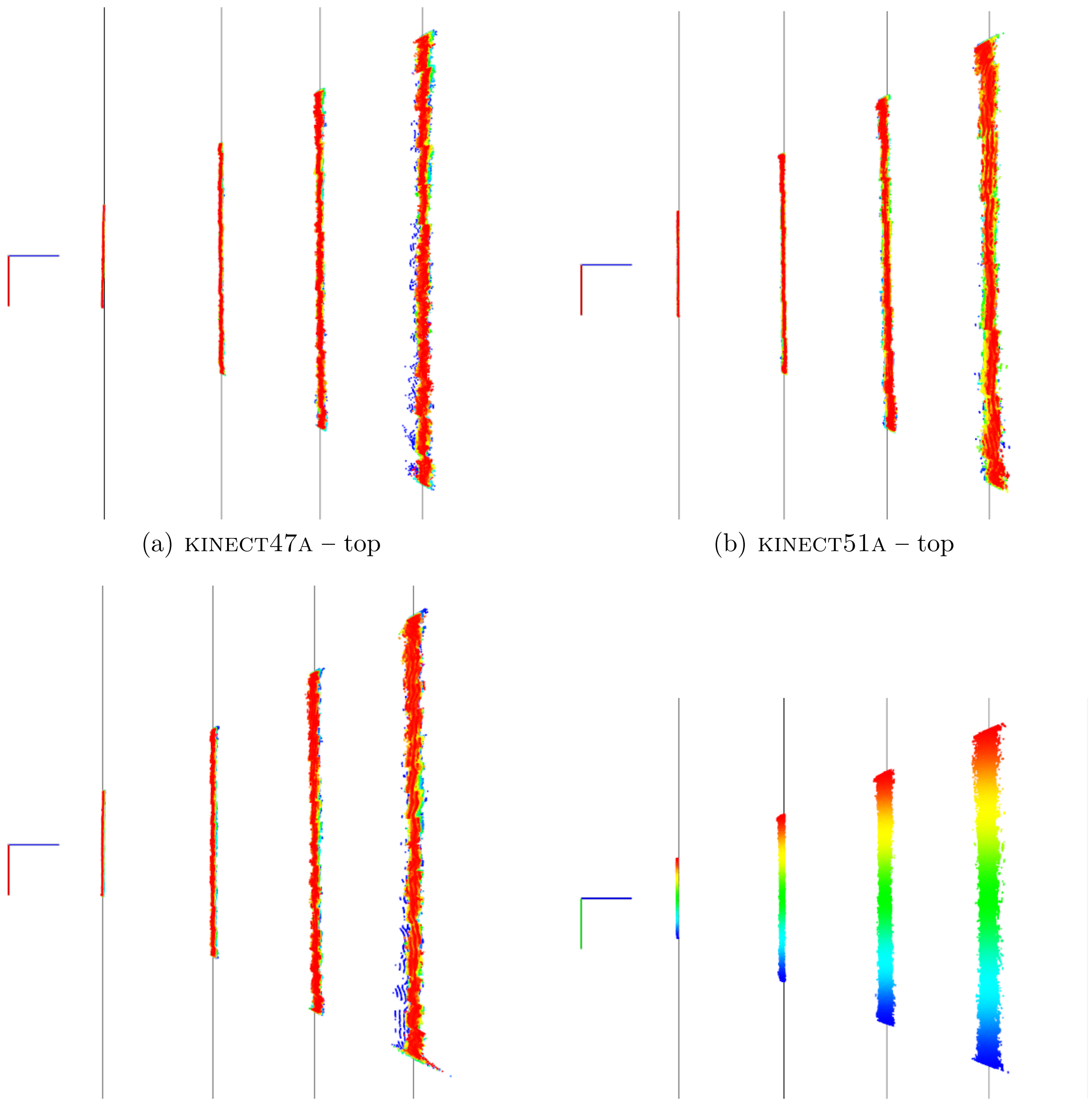
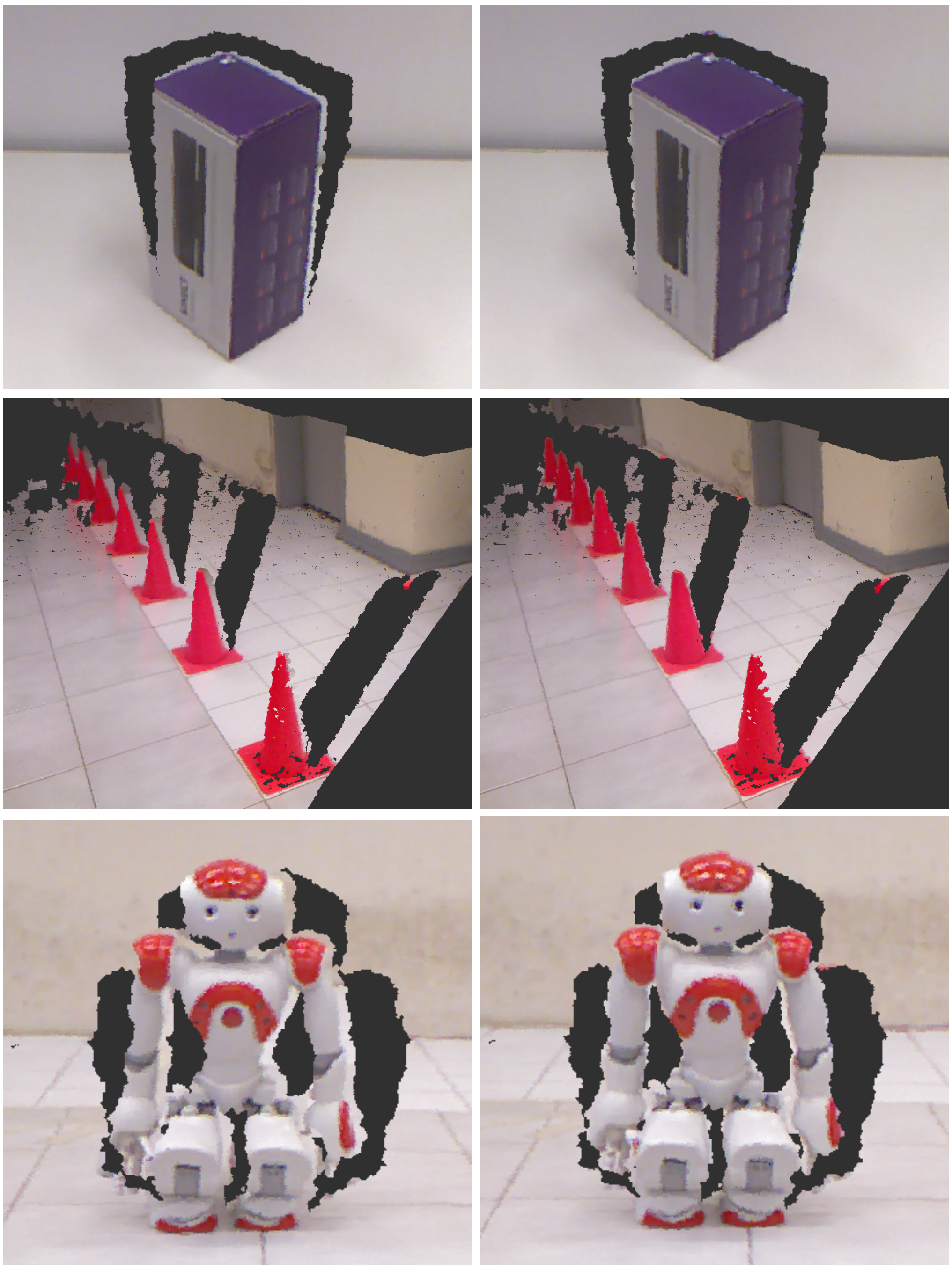

(a) KINECT47A – top
(b) KINECT51A – top
(c) ASUS – top
(d) KINECT51A – side

Figure 4.25: Top and side views of the point clouds of Figure 4.1 after the undistortion phase (Figure 4.19) and the global correction phase. Again, the gray lines show the depth measured by means of the laser meters. As we can see, every cloud is now planar as well as in the right position and correctly oriented.



(a) Default transformation (b) Computed transformation

Figure 4.26: Colored point clouds resulting from the fusion of depth and RGB data. For each cloud, the points are firstly transformed into camera coordinates, then projected into the image and colored with the RGB color of the obtained pixel. As we can see, the results obtained with the newly computed transformation (b) are definitely better than those obtained with the default one (a).

not need to be modified to take into account the calibration parameters.
The code is available as open source in a GitHub repository:

`http://github.com/iaslab-unipd/rgbd_calibration`.

Apart from ROS, its main dependencies are Eigen [21], OpenCV [9], PCL [44] and Ceres Solver [1].

4.7.1 Performance

We tested both the calibration and the correction node in terms of the execution time of the whole calibration procedure and the correction of a single input cloud, respectively. For what concerns the calibration procedure, the data collection part takes no more than 10 minutes for a hundred images. The data processing, instead, lasts about 45 minutes on a high-end consumer laptop[1]. This duration refers to a calibration that uses depth images of size 640×480 pixels, i.e. the full resolution ones. Clouds downsampled to the size of 320×240 pixels, let the calibration procedure last no more than 5 minutes, nevertheless the accuracy of the resulting parameters is not reduced.

On one side, the calibration is an operation that is performed once and so the execution time is not critical. On the other side, the execution time of the correction node is critical, since the data generated by a depth sensor, typically with a frequency of 30 Hz, must be elaborated in real-time. For this reason, we tested the performance of 3 different implementations of the correction algorithm:

- a standard CPU implementation;
- a parallel CPU implementation exploiting the OpenMP directives;
- a parallel GPU implementation using CUDA.

The results are reported in Figure 4.27. Clearly the GPU implementation outperforms the CPU ones, but dedicated hardware is needed. Going deeper into details, most of the time spent for elaborating the cloud on the GPU (about 95%) is dedicated to the copy of the data to and from the GPU memory.
Anyway, all the implementations are able to correct the clouds in real-time.

4.8 Conclusions

In this chapter we presented a novel method to calibrate a structured-light depth sensor. The proposed calibration procedure only requires the user to collect

[1]CPU: Intel Core i7-4700MQ, RAM: 16GB, SSD, GPU: NVidia GTX 750M.

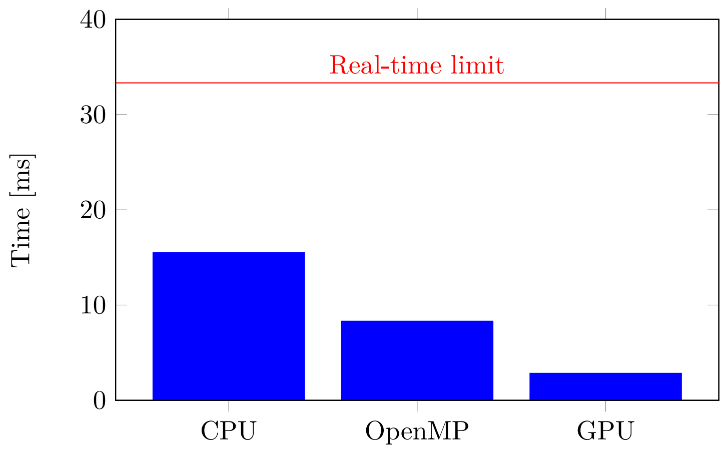


Figure 4.27: Time comparison between the 3 different implementations of the correction node. The red line indicates the time limit to elaborate point clouds at 30Hz.

data in a minimally structured environment (e.g., a wall with attached a checkerboard) and, afterwards, let the software process the data. We proposed to model the depth sensor error by means of two different components, a distortion error and a global, systematic error. The distortion error is modeled using a per-pixel parametric undistortion map, estimated in the first stage of the algorithm. The depth systematic error along with the camera-depth sensor alignment are estimated in the second stage of the algorithm, inside a robust optimization framework [1].

A detailed set of tests on both the input data and the results of the calibration showed that the proposed approach is able to correctly recover the shape of the framed scene. Moreover, the RGB clouds obtained from the the fusion of RGB and depth data show that even the rigid-displacement between the two sensors is correctly estimated. In fact, reported results show the possibility to improve the accuracy of a low cost RGB-D sensor with a very simple, yet human-friendly, procedure.

The main drawback of the proposed method is its high dependency on good intrinsic parameters for the sensor used during the calibration procedure. We noticed that bad parameters lead to estimate a wrong depth sensor-camera transformation. In our tests, also the global correction functions, sometimes, did not result as good as expected. Future work includes a better estimation of the depth sensor intrinsics (focal lengths and central point) in the optimization procedure.

Chapter 5

Calibration of a Depth Sensor-Camera Network

Robotic systems and camera networks consist of many heterogeneous vision sensors, mainly cameras and depth sensors. The estimation of the poses of all such sensors with respect to a unique, consistent world frame, is a challenging and well-known problem. As a matter of fact, a good calibration of these sensors can be a useful starting point for several applications in the computer vision field (e.g. 3D mapping, people recognition and tracking [6, 36], microphone calibration for audio localization [30]) as well as in many robotics applications (e.g. simultaneous localization and mapping (SLAM) applications, grasping and manipulation).

However, even if most of the time a good calibration is mandatory for the success of the application, there are still no tools that permit to easily calibrate multiple vision sensors together in a uniform way. In fact, most of the existing tools are for specific applications or specific sensors (e.g. stereo cameras); there are only few methods developed to simultaneously calibrate an heterogeneous sensor network. As stated by Le et al. [29], the most followed approach is to divide the sensors into pairs and calibrate each pair independently, even using different algorithms for each one.

Our idea, i.e., the one behind the development of this package, is to go beyond this calibration technique, and develop an easy-to-use and easy-to-extend calibration package for ROS, such that users can add their own sensor types, their own error functions and perform the calibration. Moreover, since calibration is a time-consuming task, a fast procedure would be a very useful tool, especially when the involved sensors need often to be moved – and therefore re-calibrated.

The package addresses the problem of calibrating networks composed by cameras and depth sensors like the one in Figure 5.1. The approach we followed is an extension of the classical single camera calibration procedure [70]: users are

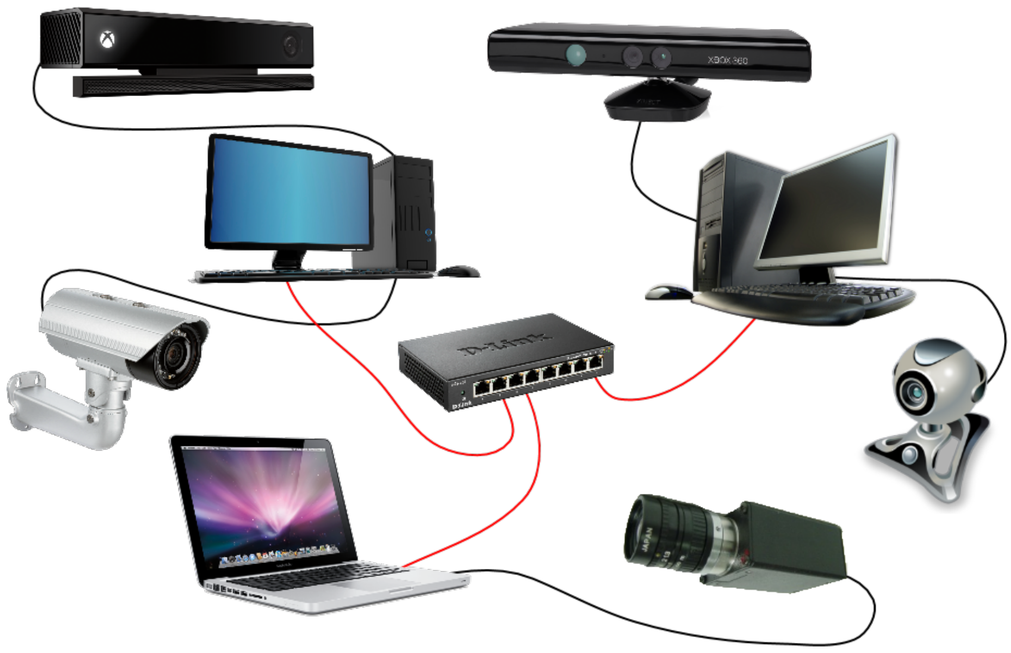


Figure 5.1: Example of sensors in a PC network that the proposed package aims to calibrate.

asked to move a checkerboard pattern in front of each camera and depth sensor and, as soon as any of the sensors see the checkerboard, the calibration starts. Then, whenever the pattern is visible by at least two sensors simultaneously, a constraint is added to the calibration problem. This process goes on until all the sensors are connected to the others. At the end, all the data are processed inside an optimization framework that improves the quality of the initial estimation.

The proposed package improves the state of the art in various ways. First of all, differently from the rest of the existent tools, the calibration algorithm is distributed in the network. That is, part of the data process (the features extraction) is performed on the PCs directly connected to the sensors, the remaining part (the optimization), instead, is performed on a central PC. This results in a calibration tool that can deal with a dozen sensors easily. In fact, the bandwidth usage is kept very low since most of the data are elaborated locally instead of being transmitted over the network.

Then, with respect to the state of the art, the calibration is computed using an established and reliable optimization framework: Ceres Solver [1]. This framework allows to calibrate the network in an on-line fashion: the calibration is performed/refined as soon as new data are acquired.

The remainder of the chapter is organized as follows. In Section 5.1 we review some of the existing works about the calibration of camera and depth sensor networks. We detail the proposed calibration approach for camera-only networks in Section 5.2, while its extension to networks composed of both cameras and depth sensors is described in Section 5.3. In Section 5.4 we define the package architecture and show how to configure and use the provided nodes. In Section 5.5 we report results of both simulations and real-world tests, while in Section 5.6 we give a brief overview of an open source project, OpenPTrack. Finally, in Section 5.7, we draw some conclusions.

5.1 Related Work

In the robot vision field, RGB cameras have been a key technology for the development of visual perception. In the last few years, the introduction of RGB-D sensors contributed deeply to the advancement of data fusion in practical applications. Auvinet et al. [3] proposed a new method for calibrating multiple depth cameras for body reconstruction using only depth information. Their algorithm is based on plane intersections and the NTP protocol for data synchronization. The calibration achieves good results: even if the depth error of the sensor is 10 mm, the reconstruction error with 3 depth cameras is, in the best case, less than 6 mm. However, they have to manually select the plane corners and, above all, they only deal with depth sensors.

Another approach to solve the calibration problem for a network of cameras and depth sensors, is the one proposed by Le and Ng [29]: they proposed to jointly calibrate groups of sensors. More specifically, the groups were composed by a set of sensors able to provide a 3D representation of the world (e.g. a stereo camera, an RGB camera and a depth camera, etc.). First of all they calibrated the intrinsics of each sensor, secondly they calibrated the extrinsic parameters of each group, then they calibrated the extrinsic parameters of each group with respect to all the others. Finally the calibration parameters were refined in one optimization step. Their experiments show that this method not only reduces the calibration error, but also requires a little human intervention. An advantage of having groups that output 3D data is that the same calibration object can be used to calibrate a group with respect to all the others, regardless of the sensor type. Also, a joint calibration does not accumulate errors like a calibration based on sensor pairs do. The main drawback of their approach is that they always need to group sensors beforehand in order to have 3D data outputs.

Finally, Furgale et al. [18], recently developed a similar ROS package, `Kalibr`[1], that tackles the spatio-temporal calibration of multi-sensor systems composed of

[1]`https://github.com/ethz-asl/kalibr`

cameras and an IMU. To the best of our knowledge, this is the work most similar to ours, even though it does not deal with depth sensors.

5.2 Camera-only Network Calibration

Definition 5.1 (Network Calibration). *Let* $\mathsf{S} = \{\mathfrak{S}_1, \mathfrak{S}_2 \dots \mathfrak{S}_N\}$ *be a set of sensors in a network. The calibration problem consists in finding, for each sensor* $\mathfrak{S}_i \in \mathsf{S}, i = 1 \dots N$*, its pose* ${}^{\mathcal{W}}\mathcal{S}_i$ *with respect to a common reference frame* $\mathcal{W}$*, namely the* world.

To solve the calibration problem (Definition 5.1) for a network composed by cameras, we use a *checkerboard pattern*. Firstly, we estimate the camera poses with respect to a common reference frame with a direct method. Then, these poses are refined taking advantage of all the data acquired, in an optimization step.

5.2.1 Pose Estimation

So, let $\mathfrak{C}_1$ and $\mathfrak{C}_2$ be two different cameras, with reference frame $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. Let $k \in \mathbb{N}$ be an *acquisition step*, i.e., a progressive number that is incremented each time the checkerboard is moved to a different location and an acquisition for every camera is triggered at the same instant, and let ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(k)}$ be the checkerboard pose at step k. Then, if both cameras see the checkerboard, we can compute ${}^{\mathcal{C}_1}_{\mathcal{B}}\mathbf{T}^{(k)}$ and ${}^{\mathcal{C}_2}_{\mathcal{B}}\mathbf{T}^{(k)}$ using (2.5).

Starting from these two poses (and the fact the the checkerboard is in the same location), we can estimate the pose of one camera with respect to the other ${}^{\mathcal{C}_1}_{\mathcal{C}_2}\mathbf{T}$ with a closed formula

$$ {}^{\mathcal{C}_1}_{\mathcal{C}_2}\mathbf{T} = {}^{\mathcal{C}_1}_{\mathcal{C}_2}\mathbf{T}^{(k)} = {}^{\mathcal{C}_1}_{\mathcal{B}}\mathbf{T}^{(k)} \cdot {}^{\mathcal{C}_2}_{\mathcal{B}}\mathbf{T}^{(k)^{-1}} \quad . \tag{5.1} $$

Then, recalling that affine transforms can be chained

$$ {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T} = {}^{\mathcal{B}}_{\mathcal{X}}\mathbf{T} \cdot {}^{\mathcal{X}}_{\mathcal{A}}\mathbf{T} \ , $$

and that

$$ {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T} = {}^{\mathcal{A}}_{\mathcal{B}}\mathbf{T}^{-1} \ , $$

we can find a solution to our calibration problem for a network composed by N cameras: we just need to estimate (or set) the pose ${}^{\mathcal{W}}_{\mathcal{C}_i}\mathbf{T}$ of one camera $\mathfrak{C}_i$ with respect to the world reference frame $\mathcal{W}$ and move the checkerboard around until every camera pose is computed, using any of the aforementioned equations.

To estimate the pose of a camera $\mathfrak{C}_i$ with respect to the world reference frame $\mathcal{W}$, first of all we must know whether we need to define a world reference frame $\mathcal{W}$ in the environment or not. Actually, if it really does not matter where such reference frame is, any camera reference frame $\mathcal{C}_i$ can be set as the world, that is $\mathcal{W} = \mathcal{C}_i$ (or equally ${}^{\mathcal{W}}_{\mathcal{C}_i}\mathbf{T} = \mathbf{I}_4$) for one $i \in \{1, \dots, N\}$. Otherwise the pose can be set manually: ${}^{\mathcal{W}}_{\mathcal{C}_i}\mathbf{T} = \mathbf{W}$ for some transformation matrix $\mathbf{W}$ and $i \in \{1, \dots, N\}$; or estimated by moving the checkerboard to the desired position and setting ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(k)} = \mathbf{I}_4$, for some $k \in \mathbb{N}$.

At this stage we have good estimates of the sensor poses, however, due to errors in the measurements, usually

$$ {}^{\mathcal{C}_1}_{\mathcal{C}_2}\mathbf{T}^{(k)} \neq {}^{\mathcal{C}_1}_{\mathcal{C}_2}\mathbf{T}^{(l)} $$

for two different steps k and l. Therefore we must perform an optimization step to refine the estimated camera poses, such that the error is reduced as much as possible.

5.2.2 Optimization

Taking a step back to the acquisition part, we can organize the calibration data in a matrix, like the one in Figure 5.2. Following the *bundle adjustment* approach [57], we refine both the camera poses ${}^{\mathcal{W}}_{\mathcal{C}_i}\mathbf{T}$, $i = 1 \dots N$ and the checkerboard poses ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(k)}$, $k = 1 \dots K$. Indeed, even if we perfectly know the pose of every camera with respect to the world, the pose of a checkerboard $\mathfrak{B}$ estimated at step k using two different cameras, say $\mathfrak{C}_i$ and $\mathfrak{C}_j$, is likely to be different:

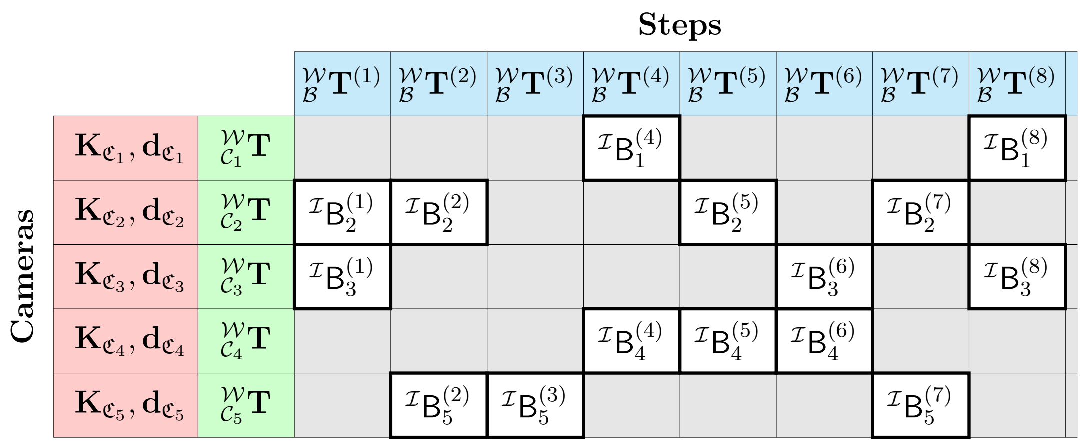
$$ {}^{\mathcal{W}}_{\mathcal{C}_i}\mathbf{T}^{(k)} \cdot {}^{\mathcal{C}_i}_{\mathcal{B}}\mathbf{T}^{(k)} \neq {}^{\mathcal{W}}_{\mathcal{C}_j}\mathbf{T}^{(k)} \cdot {}^{\mathcal{C}_j}_{\mathcal{B}}\mathbf{T}^{(k)} \quad . $$

To achieve a satisfying solution, a good candidate to be minimized is the reprojection error defined in (2.6).

Suppose the error on the corner estimation in the images provided by each camera $\mathfrak{C}_i$ is distributed as a Gaussian $\mathcal{N}(0, \sigma^2_{\mathfrak{C}_i})$; typical values for $\sigma_{\mathfrak{C}_i}$ are 0.5 or 1. Then, the complete error function $E_{\mathfrak{C}}$ that let us refine the sensor poses is

$$ \begin{aligned} E_{\mathfrak{C}} &\triangleq \sum_{k=1}^{K}\sum_{i=1}^{N} u_{ik} \cdot \frac{1}{\sigma^2_{\mathfrak{C}_i}} \cdot \mathrm{e}_{\mathrm{r}_{\mathfrak{C}_i}} \left({}^{\mathcal{W}}_{\mathcal{C}_i}\mathbf{T}^{-1} \cdot {}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(k)}, {}^{\mathcal{B}}\mathsf{B}, {}^{\mathcal{I}}\mathsf{B}^{(k)}_i \right) \qquad (5.2) \\ &= \sum_{k=1}^{K}\sum_{i=1}^{N} u_{ik} \cdot \frac{1}{\sigma^2_{\mathfrak{C}_i}} \cdot \sum_{(r,c)\in\mathbb{I}_{\mathfrak{B}}} \left\| {}^{\mathcal{I}}\mathbf{b}^{(k)}_{i,r,c} - \mathrm{r}_{\mathfrak{C}_i}\left({}^{\mathcal{W}}_{\mathcal{C}_i}\mathbf{T}^{-1} \cdot {}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(k)} \cdot {}^{\mathcal{B}}\mathbf{b}_{r,c} \right) \right\|^2 \quad , \end{aligned} $$

where u_{ik} is an indication function equal to 1 if camera $\mathfrak{C}_i$ sees the checkerboard at step k (otherwise it is 0).



Steps

Cameras

		${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(1)}$	${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(2)}$	${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(3)}$	${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(4)}$	${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(5)}$	${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(6)}$	${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(7)}$	${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(8)}$
$\mathbf{K}_{\mathfrak{C}_1}, \mathbf{d}_{\mathfrak{C}_1}$	${}^{\mathcal{W}}_{\mathcal{C}_1}\mathbf{T}$				${}^{\mathcal{I}}\mathsf{B}_1^{(4)}$				${}^{\mathcal{I}}\mathsf{B}_1^{(8)}$
$\mathbf{K}_{\mathfrak{C}_2}, \mathbf{d}_{\mathfrak{C}_2}$	${}^{\mathcal{W}}_{\mathcal{C}_2}\mathbf{T}$	${}^{\mathcal{I}}\mathsf{B}_2^{(1)}$	${}^{\mathcal{I}}\mathsf{B}_2^{(2)}$			${}^{\mathcal{I}}\mathsf{B}_2^{(5)}$		${}^{\mathcal{I}}\mathsf{B}_2^{(7)}$	
$\mathbf{K}_{\mathfrak{C}_3}, \mathbf{d}_{\mathfrak{C}_3}$	${}^{\mathcal{W}}_{\mathcal{C}_3}\mathbf{T}$	${}^{\mathcal{I}}\mathsf{B}_3^{(1)}$					${}^{\mathcal{I}}\mathsf{B}_3^{(6)}$		${}^{\mathcal{I}}\mathsf{B}_3^{(8)}$
$\mathbf{K}_{\mathfrak{C}_4}, \mathbf{d}_{\mathfrak{C}_4}$	${}^{\mathcal{W}}_{\mathcal{C}_4}\mathbf{T}$				${}^{\mathcal{I}}\mathsf{B}_4^{(4)}$	${}^{\mathcal{I}}\mathsf{B}_4^{(5)}$	${}^{\mathcal{I}}\mathsf{B}_4^{(6)}$		
$\mathbf{K}_{\mathfrak{C}_5}, \mathbf{d}_{\mathfrak{C}_5}$	${}^{\mathcal{W}}_{\mathcal{C}_5}\mathbf{T}$		${}^{\mathcal{I}}\mathsf{B}_5^{(2)}$	${}^{\mathcal{I}}\mathsf{B}_5^{(3)}$				${}^{\mathcal{I}}\mathsf{B}_5^{(7)}$	

Figure 5.2: Matrix view of the calibration data. Each row is associated to a camera $\mathfrak{C}_i$ and contains both the camera parameters $\mathbf{K}_{\mathfrak{C}_i}$ and $\mathbf{d}_{\mathfrak{C}_i}$, and the camera estimated pose ${}^{\mathcal{W}}_{\mathcal{C}_i}\mathbf{T}$. The columns are instead associated to the steps and contain the poses of the checkerboard at every step k, namely ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(k)}$. A cell (i, k) contains the corners locations (in pixels) ${}^{\mathcal{I}}\mathsf{B}_i^{(k)}$ of the checkerboard at step k in the image provided by camera $\mathfrak{C}_i$, if the checkerboard is visible.

5.2.3 Additional Constraints

In one of our first applications of this calibration algorithm, OpenPTrack (see Section 5.6), we needed to calibrate a camera network in a big room against the floor, i.e. extract the floor equation and set the world frame somewhere on it. We decided to estimate the floor coefficients during the calibration procedure, exploiting the fact that positioning a checkerboard on the floor would have allowed us to define the plane equation as well as the world reference frame. However, since the room was quite big and the checkerboard far from every camera, the results were not satisfactory: the plane had often a non-negligible rotation with respect to the real one. To overcome this issue, printing a bigger checkerboard was not a viable solution. Instead, we imposed that two or more checkerboards were lying on the same plane and added the geometrical constraints to the error model. In fact, if we fix the plane π on which a checkerboard can move, the checkerboard pose can be defined by a 2D transform ${}^{\mathcal{P}}_{\mathcal{B}}\mathbf{T}_2$ with respect to the reference frame of plane π, namely $\mathcal{P}$.

So, let define a plane by means of its reference frame ${}^{\mathcal{W}}_{\mathcal{P}}\mathbf{T}$, such that ${}^{\mathcal{P}}\mathcal{X}$ and ${}^{\mathcal{P}}\mathcal{Y}$ are on the plane and ${}^{\mathcal{P}}\mathcal{Z}$ is its normal, as depicted in Figure 5.4. The pose of a checkerboard lying on π at step k is

$$
{}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(k)} = {}^{\mathcal{W}}_{\mathcal{P}}\mathbf{T} \cdot {}^{\mathcal{P}}_{\mathcal{B}}\mathbf{T}_2^{(k)}, \tag{5.3}
$$

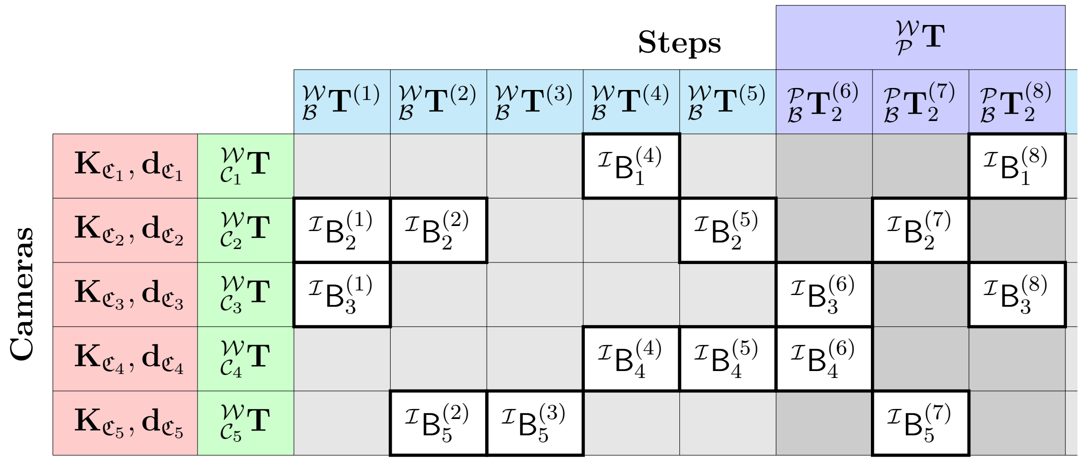

Steps
${}^{\mathcal{W}}_{\mathcal{P}}\mathbf{T}$
${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(1)}$ ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(2)}$ ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(3)}$ ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(4)}$ ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(5)}$ ${}^{\mathcal{P}}_{\mathcal{B}}\mathbf{T}_2^{(6)}$ ${}^{\mathcal{P}}_{\mathcal{B}}\mathbf{T}_2^{(7)}$ ${}^{\mathcal{P}}_{\mathcal{B}}\mathbf{T}_2^{(8)}$
Cameras
$\mathbf{K}_{\mathfrak{C}_1}, \mathbf{d}_{\mathfrak{C}_1}$ ${}^{\mathcal{W}}_{\mathcal{C}_1}\mathbf{T}$ ${}^{\mathcal{I}}\mathsf{B}_1^{(4)}$ ${}^{\mathcal{I}}\mathsf{B}_1^{(8)}$
$\mathbf{K}_{\mathfrak{C}_2}, \mathbf{d}_{\mathfrak{C}_2}$ ${}^{\mathcal{W}}_{\mathcal{C}_2}\mathbf{T}$ ${}^{\mathcal{I}}\mathsf{B}_2^{(1)}$ ${}^{\mathcal{I}}\mathsf{B}_2^{(2)}$ ${}^{\mathcal{I}}\mathsf{B}_2^{(5)}$ ${}^{\mathcal{I}}\mathsf{B}_2^{(7)}$
$\mathbf{K}_{\mathfrak{C}_3}, \mathbf{d}_{\mathfrak{C}_3}$ ${}^{\mathcal{W}}_{\mathcal{C}_3}\mathbf{T}$ ${}^{\mathcal{I}}\mathsf{B}_3^{(1)}$ ${}^{\mathcal{I}}\mathsf{B}_3^{(6)}$ ${}^{\mathcal{I}}\mathsf{B}_3^{(8)}$
$\mathbf{K}_{\mathfrak{C}_4}, \mathbf{d}_{\mathfrak{C}_4}$ ${}^{\mathcal{W}}_{\mathcal{C}_4}\mathbf{T}$ ${}^{\mathcal{I}}\mathsf{B}_4^{(4)}$ ${}^{\mathcal{I}}\mathsf{B}_4^{(5)}$ ${}^{\mathcal{I}}\mathsf{B}_4^{(6)}$
$\mathbf{K}_{\mathfrak{C}_5}, \mathbf{d}_{\mathfrak{C}_5}$ ${}^{\mathcal{W}}_{\mathcal{C}_5}\mathbf{T}$ ${}^{\mathcal{I}}\mathsf{B}_5^{(2)}$ ${}^{\mathcal{I}}\mathsf{B}_5^{(3)}$ ${}^{\mathcal{I}}\mathsf{B}_5^{(7)}$

Figure 5.3: Matrix view of the calibration data with a plane constraint ${}^{\mathcal{W}}_{\mathcal{P}}\mathbf{T}$. The columns associated to the checkerboards constrained to lie on the plane now contain the checkerboard pose with respect to the plane ${}^{\mathcal{P}}_{\mathcal{B}}\mathbf{T}_2^{(k)}$.

where the 2D transform ${}^{\mathcal{P}}_{\mathcal{B}}\mathbf{T}_2^{(k)}$ is wrapped into a 3D one to perform the matrix multiplication

$${}^{\mathcal{P}}_{\mathcal{B}}\mathbf{T}_2^{(k)} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & t_x \\ \sin(\theta) & \cos(\theta) & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

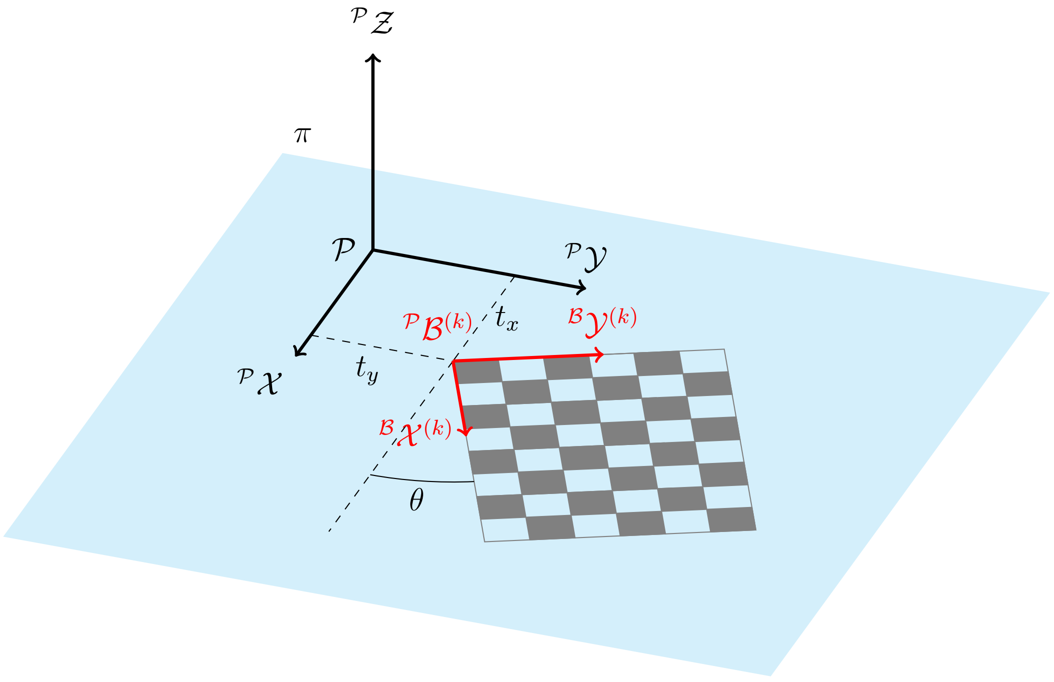
We can now substitute (5.3) into (5.2) to refine both the plane and the checkerboard pose. An overview of the data matrix with the constraint added is depicted in Figure 5.3.

5.3 Extension to a Depth Sensor-Camera Network

5.3.1 Pose Estimation

In Section 5.2 we have presented a calibration procedure for networks composed of cameras. Such procedure works well with cameras, but how can we calibrate a network composed by both cameras and depth sensors?

A depth sensor $\mathfrak{D}$ provides a point cloud reflecting the shape of the scene in the sensor field of view. Obviously, we cannot directly estimate the checkerboard pose using the checkerboard corners (as we did for calibrating a camera network) because they are not visible in the depth images. Instead, we can exploit the 3D



${}^{\mathcal{P}}\mathcal{Z}$
π
$\mathcal{P}$
${}^{\mathcal{P}}\mathcal{Y}$
${}^{\mathcal{P}}\mathcal{B}^{(k)}$
t_x
${}^{\mathcal{B}}\mathcal{Y}^{(k)}$
${}^{\mathcal{P}}\mathcal{X}$
t_y
${}^{\mathcal{B}}\mathcal{X}^{(k)}$
θ

Figure 5.4: Reference frames of the plane π and a checkerboard $\mathfrak{B}$ lying on it.

data to extract the pattern plane and perform a plane-to-plane calibration [58].

Provided we have already estimated the checkerboard pose ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(k)}$ at step k, we can define the checkerboard plane ${}^{\mathcal{W}}\pi_{\mathfrak{B}}^{(k)}$ using three non-collinear corners. We can estimate also the pattern plane ${}^{\mathcal{D}}\pi_{\mathfrak{B}}^{(k)}$ from the point cloud, using a RANSAC-based [16] plane fitting algorithm. Then, let $\{k_1, \ldots, k_n\}$, with $n \geq 3$, be the intersection between the steps in which depth sensor $\mathfrak{D}$ sees the checkerboard and those in which the checkerboard pose has been computed, and let the plane equations be of the form $\mathbf{n}^\mathsf{T} \cdot \mathbf{x} - d = 0$. Following [58], we define

$$
\begin{aligned}
{}^{\mathcal{W}}\mathbf{N} &\triangleq \left({}^{\mathcal{W}}\mathbf{n}_{\mathfrak{B}}^{(k_1)} \ldots {}^{\mathcal{W}}\mathbf{n}_{\mathfrak{B}}^{(k_n)}\right)^\mathsf{T} & \qquad {}^{\mathcal{W}}\mathbf{d} &\triangleq \left({}^{\mathcal{W}}d_{\mathfrak{B}}^{(k_1)} \ldots {}^{\mathcal{W}}d_{\mathfrak{B}}^{(k_n)}\right)^\mathsf{T} \\
{}^{\mathcal{D}}\mathbf{N} &\triangleq \left({}^{\mathcal{D}}\mathbf{n}_{\mathfrak{B}}^{(k_1)} \ldots {}^{\mathcal{D}}\mathbf{n}_{\mathfrak{B}}^{(k_n)}\right)^\mathsf{T} & \qquad {}^{\mathcal{D}}\mathbf{d} &\triangleq \left({}^{\mathcal{D}}d_{\mathfrak{B}}^{(k_1)} \ldots {}^{\mathcal{D}}d_{\mathfrak{B}}^{(k_n)}\right)^\mathsf{T}
\end{aligned}
$$

and compute the depth sensor pose

$$
{}^{\mathcal{W}}_{\mathcal{D}}\mathbf{T} = \begin{pmatrix} {}^{\mathcal{W}}_{\mathcal{D}}\mathbf{R} & {}^{\mathcal{W}}_{\mathcal{D}}\mathbf{t} \\ \mathbf{0}^\mathsf{T} & 1 \end{pmatrix}
$$

as

$$
\begin{aligned}
{}^{\mathcal{W}}_{\mathcal{D}}\mathbf{t} &= \left({}^{\mathcal{W}}\mathbf{N}^\mathsf{T} \cdot {}^{\mathcal{W}}\mathbf{N}\right)^{-1} \cdot {}^{\mathcal{W}}\mathbf{N}^\mathsf{T} \cdot \left({}^{\mathcal{W}}\mathbf{d} - {}^{\mathcal{D}}\mathbf{d}\right) \quad , \\
{}^{\mathcal{W}}_{\mathcal{D}}\mathbf{R} &= \mathbf{V} \cdot \mathbf{U}^\mathsf{T} \quad ,
\end{aligned}
$$

where $\mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^\mathsf{T}$ is the SVD decomposition of ${}^{\mathcal{D}}\mathbf{N}^\mathsf{T} \cdot {}^{\mathcal{W}}\mathbf{N}$.

5.3.2 Optimization

For what concerns the error function, we calculate it as a sort of distance between the plane defined by the checkerboard ${}^{\mathcal{W}}\pi^{(k)}_{\mathfrak{B}}$ at step k, and the plane fitted to the checkerboard depth data of sensors $\mathfrak{D}_j$, $j = 1, \ldots, M$, i.e. ${}^{\mathcal{D}_j}\pi^{(k)}_{\mathfrak{B}}$. So, let $\mathrm{p}_\pi(\mathbf{x})$ be the *line-of-sight* projection of a point $\mathbf{x}$ onto plane π as described in Section 4.4.1, and let the error on the depth measurements of sensor $\mathfrak{D}_j$ be distributed as a Gaussian $\mathcal{N}\left(0, \sigma^2_{\mathfrak{D}_j}(z)\right)$ with mean 0 and variance dependent on the depth value z (see Chapter 4). We define the error function $E_{\mathfrak{D}}$ as

$$E_{\mathfrak{D}} \triangleq \sum_{k=1}^{K}\sum_{j=1}^{M} u_{jk} \cdot \sum_{(r,c)\in\mathbb{I}_{\mathfrak{B}}} \frac{1}{\sigma^2_{\mathfrak{D}_j}\left({}^{\mathcal{D}_j}z^{(k)}_{r,c}\right)} \cdot \left\| {}^{\mathcal{D}_j}\mathbf{b}^{(k)}_{r,c} - \mathrm{p}_{{}^{\mathcal{D}_j}\pi^{(k)}_{\mathfrak{B}}}\left({}^{\mathcal{D}_j}\mathbf{b}^{(k)}_{r,c}\right) \right\|^2 \tag{5.4}$$

where

$${}^{\mathcal{D}_j}\mathbf{b}^{(k)}_{r,c} = {}^{\mathcal{W}}_{\mathcal{D}_j}\mathbf{T}^{-1} \cdot {}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T}^{(k)} \cdot {}^{\mathcal{B}}\mathbf{b}_{r,c} \quad ,$$

and u_{jk} is an indication function equal to 1 if sensor $\mathfrak{D}_j$ sees the checkerboard at step k (otherwise it is 0), and ${}^{\mathcal{D}_j}z^{(k)}_{r,c}$ is the z component of ${}^{\mathcal{D}_j}\mathbf{b}^{(k)}_{r,c}$.

5.4 ROS Package

The presented calibration approach has been implemented in C++ within the ROS framework [42] and is available with an open source license in a GitHub repository: `https://github.com/iaslab-unipd/calibration_toolkit`. An example of how to install the package and perform a calibration is reported in Appendix A. The calibration procedure is based on the assumption that all the sensors are already calibrated, i.e. all their intrinsic parameters are known. For what concerns cameras, it just needs the intrinsic parameters to be passed together with the images. For what concerns depth sensors, instead, the package expects the data it receives to be already corrected, since an established way of correcting the data does not exist yet.

5.4.1 Architecture

The main purpose of the here-presented package is to allow the calibration of all the sensors (for now cameras and Kinect-like depth sensors) within a ROS network, no matter where they are. That is, suppose to have a set of sensors *distributed* in a PC network as in Figure 5.1, the typical approach for the calibration procedure

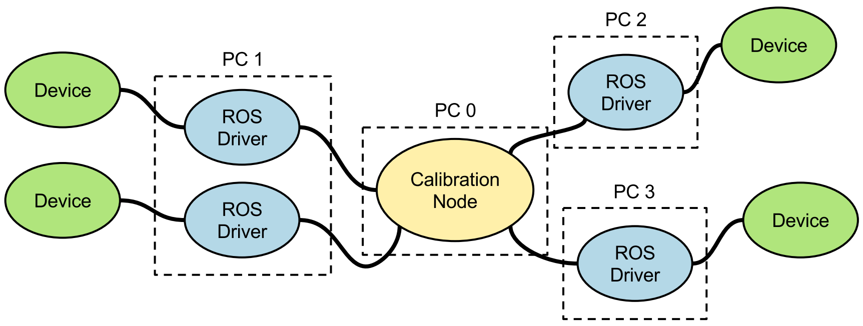


Figure 5.5: Typical approach of network calibration algorithms. All the driver nodes are directly connected to a central node that performs the computation. The bandwidth usage is high.

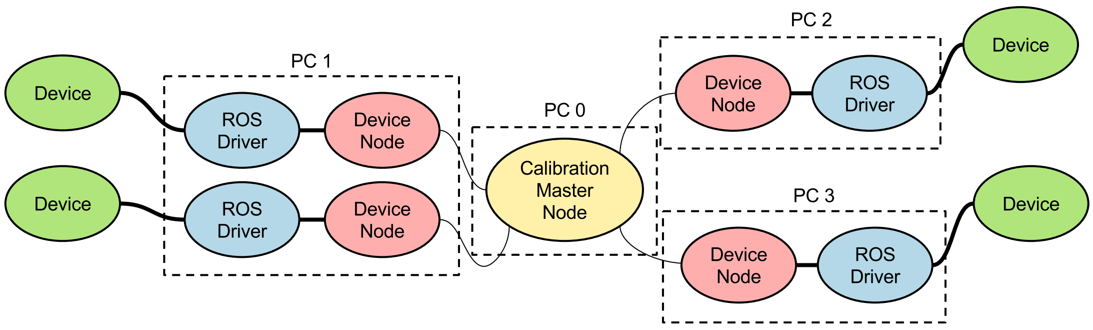


Figure 5.6: Our approach to the network calibration procedure. The data provided by a device are elaborated by a node on the same PC and only the necessary calibration features are sent to the calibration node. Here, the thin lines that connect the master node and the device nodes mean that the quantity of data over the network is limited, with respect to the quantity of data of the typical approach (Figure 5.14).

is to develop a *calibration node* that grabs the data generated by all the sensors, elaborate them and then estimate the rigid displacement of the sensors in the scene (Figure 5.14).

This approach is clearly non-scalable: more sensors means more bandwidth yet more computational power needed. To overcome this problem, we propose a different, distributed, architecture that allows to both drastically reduce the bandwidth usage and distribute the computational cost over the network: we separate the data analysis from the calibration procedure. As depicted in Figure 5.6, there are now two sorts of calibration nodes: *device nodes* and the *master node*. A device node is responsible for elaborating the data provided by its device; it extracts the corners from every image, creates a message and sends it to the master node that executes the calibration procedure.

Recalling that ROS device drivers typically stream data at a defined framerate, with the proposed architecture we are able to work in a request-reply way. It is the master node that asks for new data when needed. Note that, since ROS service calls are blocking, the communication relies on the `actionlib` stack.

5.4.2 Device Node

Node Parameters

A device node is responsible for getting the data from a device and, upon request, elaborate and send the results back to the master node. We first distinguish between the word *device* and *sensor*: a device is an item that can be connected to a PC, while a sensor is the item whose pose will be estimated in the calibration procedure. For example, a Kinect is a device composed by two different sensors: the RGB camera and the IR camera/depth sensor.

From a ROS perspective, a device node subscribes to the image and/or depth topics of the device sensors and keeps listening to an *action topic* until a request from the master node arrives. When a request is received, the last images/clouds are processed and the extracted calibration features are packed into a message and sent back to the master.

To explain how to configure and run a device node, suppose we have to create a launch file for a Kinect v1. We first need to set the name of the device and its serial in case two or more Kinects are launched on the same PC:

```xml
<?xml version="1.0"?>
<launch>
  <arg name="device_name"   default="kinect1" />
  <arg name="device_serial" default="#1" />
```

Then, to avoid conflicts between nodes launched from different PCs, we group everything inside the namespace `$ROS_PC_NAME`:

```xml
  <group ns="$(env ROS_PC_NAME)">
```

Now we include the launcher for the Kinect driver. The argument `camera` let us define the namespace for all the topics published by the driver as well as the reference frames in the Kinect messages: here we add the `_driver` suffix just to avoid confusion.

```xml
    <include file="$(find openni_launch)/launch/openni.launch">
      <arg name="camera" value="$(arg device_name)_driver" />
      <arg name="device_id" value="$(arg device_serial)" />
      <arg name="publish_tf" value="false" />
    </include>
```

The last node we need to add is our device node. The package `multisensor_calibration` already provides the node as a binary called `device_node`. So, we rename it to match our current device, paying attention that, for communication reasons, the node name needs the suffix `_node`.

```
<node pkg="multisensor_calibration" type="device_node"
      name="$(arg device_name)_node" output="screen">
```

We then define the Kinect sensors that we want to calibrate. They have to be defined within the `~device` namespace, in particular:

`name` – sets the device name, only for logging purposes;

`sensors` – defines the list of sensors to calibrate, it is divided into:

- **`intensity`** – the sensors that will be treated as pinhole cameras;
- **`depth`** – the sensors that will be treated as depth sensors;

`<sensor>` – sets, for each sensor defined in `sensors`:

- **`frame_id`** – its unique frame id;
- **`error`** – the error polynomial $\sigma_{\mathfrak{D}}(z)$ [*depth sensors only*], where:
  - **`min_degree/max_degree`** – the minimum and maximum degree of the polynomial (in the example below $\sigma_{\mathfrak{D}}(z) = c_0 \cdot z^0 + c_1 \cdot z^1 + c_2 \cdot z^2$);
  - **`coefficients`** – the polynomial coefficients c_i;

`transforms` – defines the known transforms between the sensors:

- **`<sensor>`** – the child sensor;
  - **`parent`** – the parent sensor, that is, the frame to which the transform is defined;
  - **`translation`** – the translation between the two sensors;
  - **`rotation`** – the quaternion defining the rotation between the two sensors.

```
  <rosparam param="device" subst_value="true">
    name: "$(arg device_name)"
    sensors:
      intensity: ["rgb"]
      depth: ["depth"]
    rgb:
      frame_id: "/$(env ROS_PC_NAME)/$(arg device_name)/rgb"
    depth:
          frame_id: "/$(env ROS_PC_NAME)/$(arg device_name)/
             depth"
          error:
            min_degree: 0
            max_degree: 2
            coefficients: [0.0, 0.0, 0.0035]
        transforms:
          depth:
            parent: "rgb"
            translation: {x: -0.025, y: 0.0, z: 0.0}
            rotation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}
      </rosparam>
```

Finally, we have to connect the device node to the topics published by the driver. We use the *remapping* feature of ROS to set the sensor nodes listen to the right topics. Both for depth sensors and cameras, the default topics they listen to are of the form `~/device/<sensor>/<topic type>`, where `<topic type>` is `image` for either images or depth images, `camera_info` for the camera calibration parameters and `cloud` for the point clouds:

```
      <remap from="~device/rgb/image"
             to="$(arg device_name)_driver/rgb/image_color"/>
      <remap from="~device/rgb/camera_info"
             to="$(arg device_name)_driver/rgb/camera_info"/>
      <remap from="~device/depth/cloud"
             to="$(arg device_name)_driver/depth/points" />
      <remap from="~device/depth/camera_info"
             to="$(arg device_name)_driver/depth/camera_info" /
                >
    </node>
  </group>
</launch>
```

5.4.3 Master Node

Node Parameters

Let's take a look to the launch file for the master node to see the parameters it needs to run. The launch file for the master node is simple:

```
<?xml version="1.0"?>
<launch>
  <arg name="network_file" default="$(find
     multisensor_calibration)/conf/network.yaml" />
  <arg name="checkerboard_file" default="$(find
     multisensor_calibration)/conf/checkerboard.yaml" />
  <group ns="$(env ROS_PC_NAME)">
```

```
    <node pkg="multisensor_calibration" type="master_node"
          name="master_node" output="screen">
      <param name="network_file" value="$(arg network_file)" />
      <rosparam command="load" file="$(arg checkerboard_file)"
         />
    </node>
  </group>
</launch>
```

Firstly, we need to set the file containing the network description (`network_file` parameter) to let the master node know which are the sensors that are being calibrating. The network configuration is expected to be in a *yaml* file of the form:

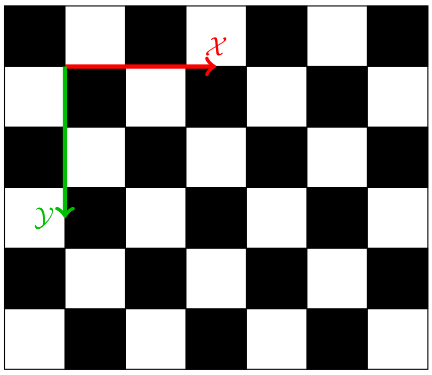
```
# Network configuration
network:
  - pc: "<ROS_PC_NAME_1>"
    devices: ["<DEVICE_NAME_1>", "<DEVICE_NAME_2>"]
  - ...
  - pc: "<ROS_PC_NAME_N>"
    devices: ["<DEVICE_NAME_1>", ..., "<DEVICE_NAME_N>"]
```

typically stored in the `multisensor_calibration/conf` directory of the master PC. Here, the `pc` parameters must match the names previously given to the PCs via the `export` command (see Appendix A), while the strings in the `devices` array are the device names. They have to match the first part of a device node, that is, all but the suffix `_node`. Note that each device node is expected to be reachable in the network using the PC name as a namespace. As an example, the device node `camera` that runs in the PC named `Gemini`, is expected to be a node called `/Gemini/camera_node`.

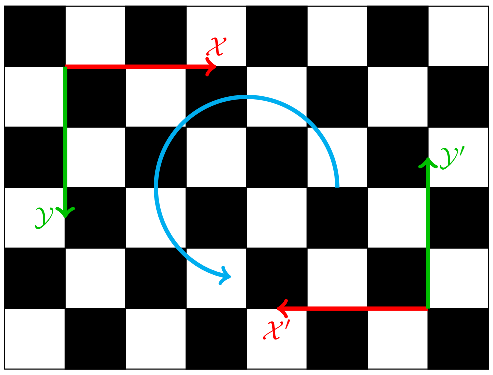
The second parameter is `checkerboard_file`. It must be a *yaml* file and contain the checkerboard pattern specifications:

```
# Checkerboard configuration
checkerboard:
  cols: <internal corners along the x dimension>
  rows: <internal corners along the y dimension>
  cell_width: <cell size along the x dimension in meters>
  cell_height: <cell size along the y dimension in meters>
```

Because of symmetry (see Figure 5.7), it is mandatory that one of `cols` and `rows` is odd and the other even, no matter which. Otherwise two different sensors can assign to the same checkerboard two different reference frames, invalidating the calibration procedure. In fact, according to our experience, the OpenCV [9] corner detector starts enumerating the corners from one of the black corner-cells (if present), in row-major order. We rely on this order to set the reference frame of the checkerboard: the $\mathcal{X}$-axis along the columns and the $\mathcal{Y}$-axis along the rows.



(a) A 5 × 6 checkerboard.



(b) A 5 × 7 checkerboard.

Figure 5.7: The reference frame of a checkerboard is located internally with respect to one of the black corner-cells (if present), according to the checkerboard size, that is, with the $\mathcal{X}$-axis along the columns and the $\mathcal{Y}$-axis along the rows. (a) The reference frame of a 5×6 checkerboard has only one possible location. (b) here, due to a rotational symmetry, the reference frame of the 5×7 checkerboard can be positioned in two different locations, leading to pose estimation problems.

Services and Messages

It is possible to interact with the master node with messages and services. The most important topic the node is listening on is `~acquisition`. Publishing a message on such topic would result in a data acquisition. Note that, the message queue on the node is set to 1, so even if the node receives multiple requests while still elaborating previous data, only the last one will be taken into account. Note also that `rostopic pub` permits to publish messages at a defined rate with the flag `-r <rate>`. Users must pay attention to use such feature since the sensors may be not perfectly synchronized. In fact, if the data are acquired while the pattern is still moving, the resulting calibration might be wrong.

The node is also listening to an action topic named `~action`. It is used to send special calibration commands in form of strings. For now the only two commands accepted are `begin plane` and `end plane`. The former command tells the calibration algorithm that, from that instant on, the checkerboards are all lying on the same plane (see Section 5.2.3). The latter, instead, makes the algorithm go back to its standard behavior.

Finally, to get the results of the calibration, the master node offers a service called `~get_results`. It can be invoked with an empty request and returns a vector of messages of type `calibration_msgs/ObjectPose`. In the future we envision to improve this service with some options like:

- ask for the poses of checkerboards and/or planes;
- set a fixed transform between the world and a sensor in the request, the response will be filled with the poses transformed according to it.

5.5 Experiments

We tested our algorithm both in simulation and in real-world scenarios. In the latter case, we also compared the results with those obtained with a state-of-the-art calibration toolbox.

5.5.1 Simulation

We simulated two different scenarios:

1. N aligned Kinects (camera + depth sensor) at a fixed distance (Figure 5.8(a)).
2. N Kinects in a circle of a known radius (Figure 5.8(b)).

We supposed that the location of a corner $\mathbf{b}$ in an image is estimated with a normally distributed error with mean $\mu_{\mathfrak{C}} = 0$ and standard deviation $\sigma_{\mathfrak{C}} = 0.5$ pixels along both axes. For each pixel (u, v) in the depth image with a ground truth depth value d, we assumed the error to be normally distributed with mean $\mu_{\mathfrak{D}} = 0$ and a standard deviation $\sigma_{\mathfrak{C}}(z) = 0.0035 \cdot z^2$ meters[2].

At the end of every simulation we computed both the translation error

$$e_{\text{tra}} \triangleq \frac{1}{2 \cdot N - 1} \sum_{i=2}^{2 \cdot N} \|\mathbf{t}_i - \hat{\mathbf{t}}_i\| \ , \tag{5.5}$$

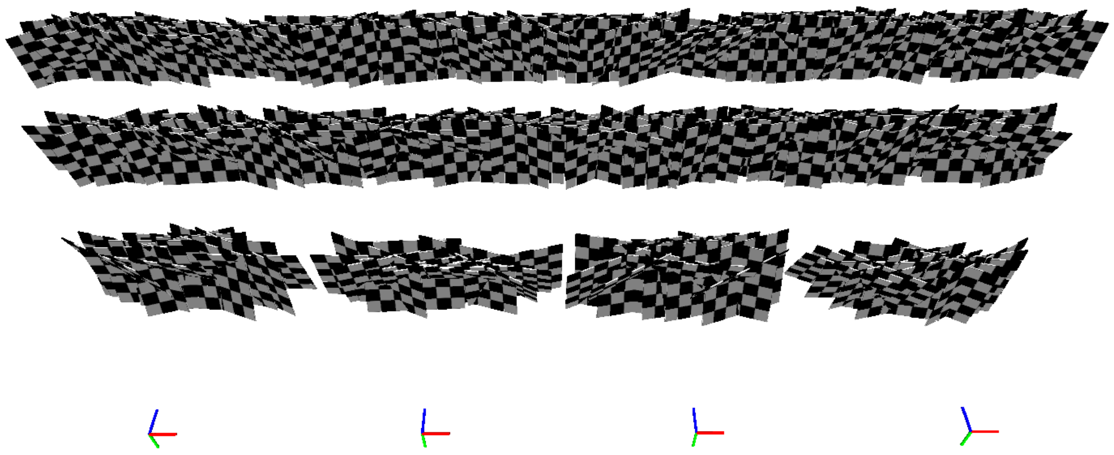
and the rotation error

$$e_{\text{rot}} \triangleq \frac{1}{2 \cdot N - 1} \sum_{i=2}^{2 \cdot N} 2 \cdot \arccos(|{\mathbf{q}_i}^{-1} \cdot \hat{\mathbf{q}}_i|) \ . \tag{5.6}$$

$\mathbf{t}_i$ and $\mathbf{q}_i$ are respectively the estimated translation and rotation (expressed as a quaternion) of sensor i while $\hat{\mathbf{t}}_i$ and $\hat{\mathbf{q}}_i$ are the real ones. All the sensor poses are referred to sensor 1 that we considered the world reference frame. Recalling that each Kinect is composed by a camera and a depth sensor, we had $2 \cdot N - 1$ sensors overall for which to calculate the two errors.

Figure 5.9 shows the translation error e_{tra} computed both varying the number of sensors N and the distance d between them. As we expected, the closer the

[2]http://wiki.ros.org/openni_kinect/kinect_accuracy


(a) N aligned Kinects at a fixed distance.

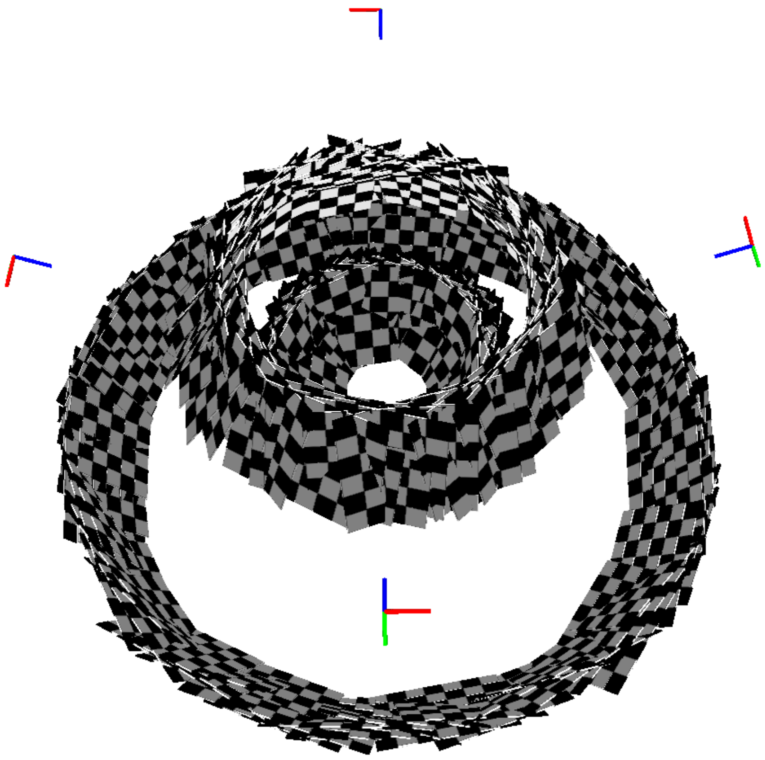

(b) N Kinects in a circle of a fixed radius.

Figure 5.8: The two scenarios we simulated for testing the calibration procedure. The checkerboard was moved along a defined path and rotated of a random angle about a random axis after each translation.

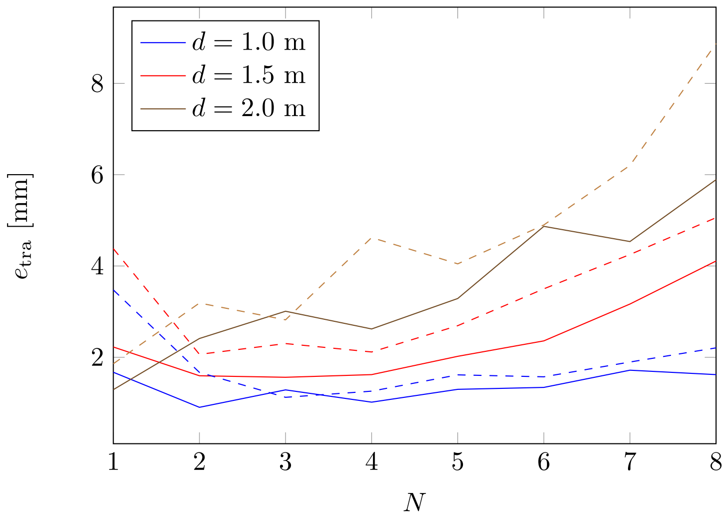


Figure 5.9: Translation error e_{tra} computed both varying the number of sensors N and the distance d between them. Continuous lines report results of tests with $60 \cdot N$ checkerboards, while dashed lines report results of tests with $30 \cdot N$ checkerboards.

sensors, the better the calibration. In fact, when the distance is small there are more checkerboards in the overlapping field of view of neighboring sensors, and therefore there are more (strong) constraints for the pose estimation.

In Figure 5.10, instead, the translation error $\mathrm{e}_{\mathrm{tra}}$ of the sensor poses when sensors are in a circle is shown. In this case, an increase in the number of sensors makes the error decrease, while an increase in the radius of the circle makes the error increase. Results about the rotation error are not reported as they are very small, usually less than 0.1°.

5.5.2 Real Tests

Quantitative Results

To prove the validity of our approach, we also performed some tests in a real scenario. We measured the real poses of the sensors by means of a motion capture system (Figure 5.11) and compared these measures with the pose estimations obtained with our package and with the ones obtained with a state-of-the-art Matlab toolbox [61]: `amcctoolbox`. The comparison is made without taking into account the depth sensors, since the `amcctoolbox` is not meant to deal with such kind of sensors. Moreover it's worth to notice that such package estimates both the intrinsic and the extrinsic parameters of all the cameras, while our does not.

Results from 6 different test scenarios are provided in Table 5.1. Test 1, 2 and

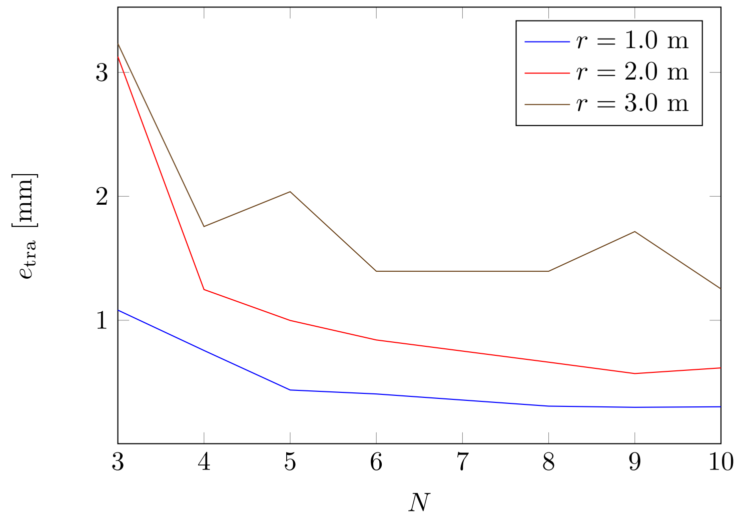


Figure 5.10: Translation error e_{tra} computed both varying the number of sensors N and the radius r of the circle.

3 are grouped together since they have been performed on the same scene, same for test 4 and 5.
For each test the table reports:

1. STEPS: The number of acquisition steps.
2. AMCC: The average translation error (5.5) after the calibration with the `amcctoolbox`.
3. OUR_D: The average translation error after the calibration with our algorithm using as intrinsic parameters the default ones provided by ROS.
4. OUR_A: The average translation error after the calibration with our algorithm using as intrinsic parameters the ones provided by `amcctoolbox`.
5. OUR_BA: The average translation error after the calibration with our algorithm using as intrinsic parameters the best ones provided by `amcctoolbox` (actually, the ones calculated during test 1).
6. MUTUAL: The average translation error between the two calibrations with the same intrinsic parameters.

Looking at the results, we can state that when using the same intrinsic parameters, the camera poses estimated by the two algorithms do not differ that much,

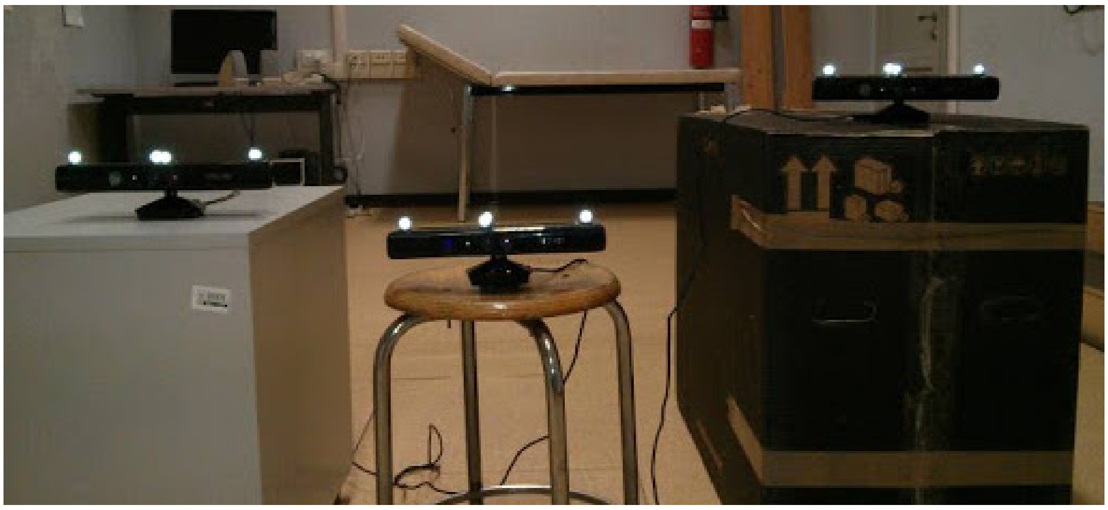


Figure 5.11: One of the test scenarios. Four markers are placed on each kinect in order to recover their locations by means of a motion capture system.

Table 5.1: Comparison between the calibration parameters estimated with `amcc-toolbox` and with our package. Highlighted are the best results obtained for each configuration.

		translation error [mm]				
TEST	STEPS	AMCC	OUR__D	OUR__A	OUR__BA	MUTUAL
1	130	16.755	16.230	**13.205**	**13.205**	11.5761
2	106	**10.371**	17.592	16.493	13.542	7.5705
3	104	**10.319**	18.118	12.512	13.949	6.0740
4	112	45.938	**29.029**	50.483	32.269	11.0807
5	101	37.628	**28.777**	43.043	32.489	8.3238
6	101	26.739	**22.744**	26.253	27.474	7.4885

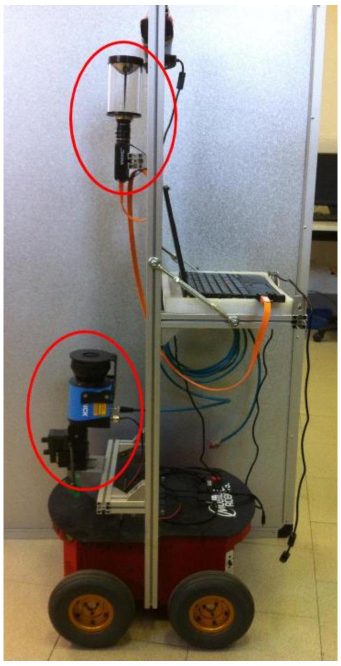


Figure 5.12: The robotic platform used in our calibration test. At the bottom you can see the laser range finder on the pan-tilt unit, at the top the omnidirectional camera.

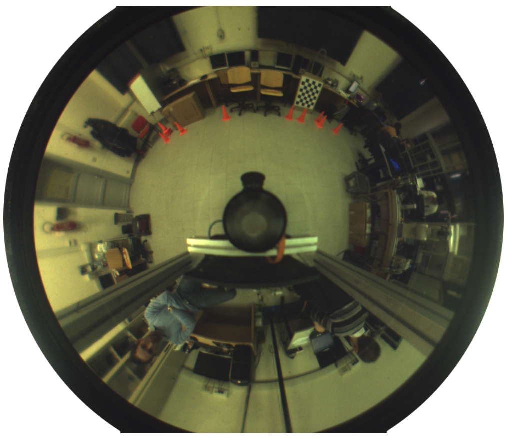
the difference between the two is more or less 1 cm (MUTUAL). Actually, the results highly depend on the intrinsic parameters estimation: keeping them constant (OUR_D and OUR_BA) makes the results more stable.

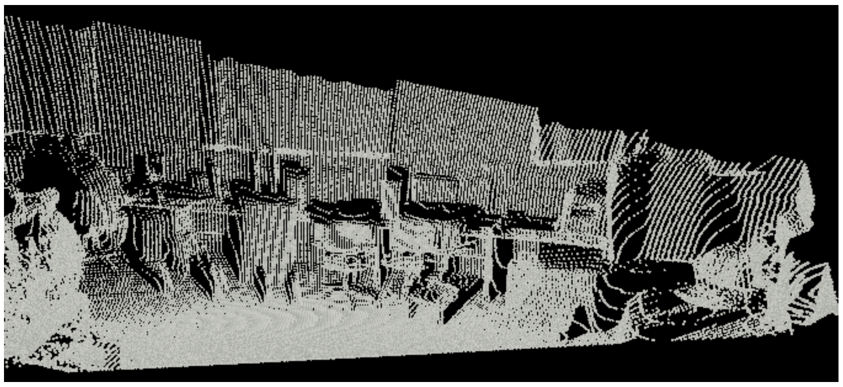
What makes the difference between our algorithm and the others, is the computational time On a modern quad-core laptop[3], we were able to perform the whole calibration procedure during the data acquisition: 2 minutes in total. We spent more than 20 minutes to perform the calibration with the `amcctoolbox`.

Qualitative Results

During the development of the toolbox, several real tests were made with different sensor types (Microsoft Kinect, Asus Xtion Pro Live, standard and omnidirectional cameras, etc.). One of the most challenging test we performed was the calibration between an actuated laser range finder (see Chapter 3) and an omnidirectional camera. We arranged a robotic platform (Figure 5.12) with a Sick LMS-100 laser range finder mounted on a Directed Perception pan-tilt unit (PTU46) and a omnidirectional camera above. We then performed our calibration and obtained the results visible in Figure 5.13. Despite the number of images and

[3]CPU: Intel i7-4700MQ, RAM: 12 GB.


(a) Omnidirectional image.


(b) 3D scan obtained from the actuated laser.

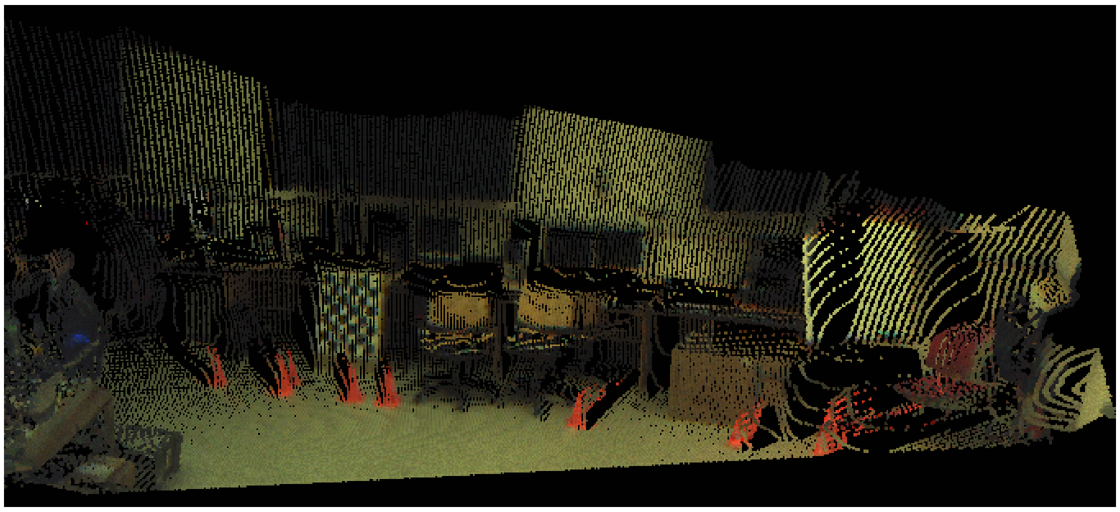

(c) Fusion of RGB and depth data.

Figure 5.13: Results of the calibration between a laser range finder and an omnidirectional camera. RGB (a) and depth (b) data are fused together to form (c).

Figure 5.14: The OpenPTrack logo.

3D scans was around 10, the results were really satisfactory.

5.6 OpenPTrack

OpenPTrack [38] is an open source project launched in 2013 to create a scalable, multi-camera solution for person tracking.

With the advent of commercially available consumer depth sensors, and continued efforts in computer vision research to improve multi-modal image and point cloud processing, robust person tracking with the stability and responsiveness necessary to drive interactive applications is now possible at low cost. But the results of the research are not easy to use for application developers.

The creation of OpenPTrack was sparked by the belief that a disruptive project was needed for artists, creators and educators to work with robust real-time person tracking in real-world projects.

OpenPTrack is developed in collaboration with UCLA REMAP[4] and Open Perception[5]. The project contains numerous state-of-the-art algorithms for RGB and/or depth tracking, and has been created on top of ROS, to support the addition and removal of different sensor streams on-line.

The network geometry in OpenPTrack is estimated with a user-friendly calibration procedure based on the one presented in this chapter. The only difference with the here-presented calibration is that only the intensity data is used to estimate the sensor poses, i.e. the depth sensors are not calibrated.

This project can be seen as a testbed for both the software architecture and algorithm we described in this chapter.

5.6.1 Calibration Procedure

The calibration procedure is meant to work in networks composed of Microsoft Kinect (v1 and v2), Mesa SwissRanger and stereo cameras. Unlike Kinect, the SwissRanger is not coupled with a color camera. For this reason, its intensity images are used for the calibration. In Figure 5.15, color images from three Kinect

[4]http://remap.ucla.edu/

[5]http://www.openperception.org/

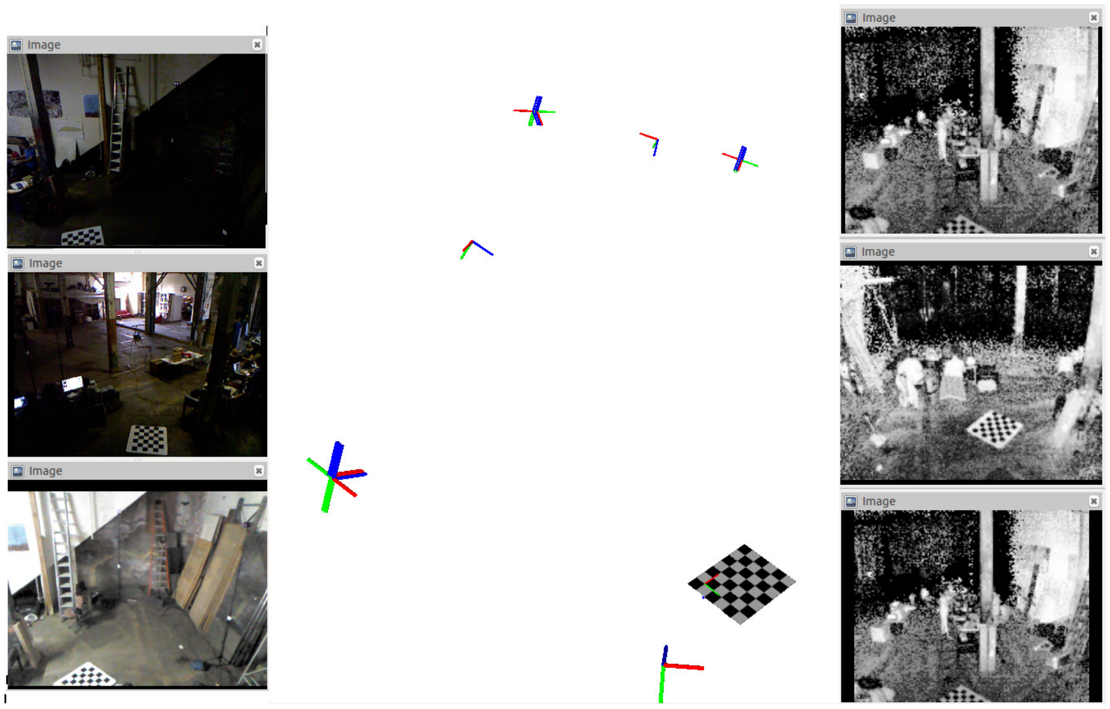


Figure 5.15: Extrinsic calibration results for a network composed of three Kinect v1 (images on the left) and three SwissRangers SR-4500 (images on the right).

v1 and equalized intensity images from three SR-4500 are shown together with the output of the network calibration procedure. The pose of every sensor is visualized as a reference frame and is positioned with respect to the ground reference frame, i.e. the world.

Another example of calibration performed using the OpenPTrack procedure is visible in Figure 5.16. In this case the network is composed of ten sensors: seven Kinect v1, two SR-4500 and a stereo pair.

5.7 Conclusions

We have presented a ROS package that lets users calibrate networks composed by cameras and depth sensors. A checkerboard pattern is used to estimate the relative poses between sensors and everything is optimized with a state-of-the-art non-linear least squares solver [1]. The choice of using ROS as the developing framework gives our implementation lots of advantages with respect to, for example, Matlab-based toolboxes [61]. With ROS, the data provided by the sensors can be used immediately to perform an online calibration and, especially, ROS allows

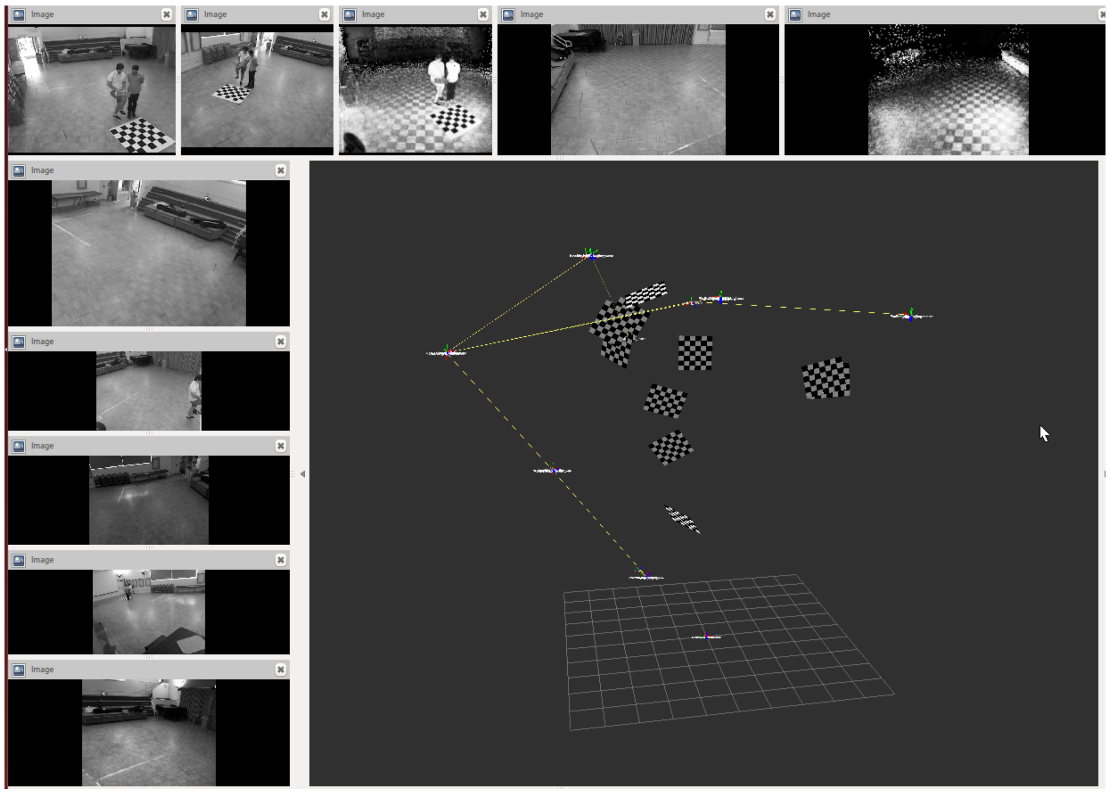


Figure 5.16: Calibration results with ten sensors: seven Kinect v1, two SR-4500 and a stereo pair.

the architecture to be easily distributed.

The proposed approach has been tested both in simulation and in real world scenarios and, provided the intrinsic parameters of the sensors are well estimated, the results are comparable to those given by a state-of-the-art toolbox. One of the main drawbacks of the presented approach is that it highly depends on the intrinsic parameters provided.

From the implementation point of view, we think that in the future, it will be really useful to provide a tool to assess the quality of the obtained calibration, in order to let users verify if the obtained results are reliable or not. Another important improvement will be at code level. In our idea, the package will be part of a *calibration ecosystem* for ROS, where contributors can implement their own calibration algorithms or contribute to the existing ones by adding, for example, new sensors. Hence, it will be necessary to define some standard interfaces between nodes and classes to lower the entry barrier for new developers and let the ecosystem grow.

Chapter 6

Conclusions

Robots are coming. What we – developers and researchers – can do, is to enhance their reliability and world awareness by creating novel applications and algorithms, and by improving the existing ones. In particular, since robots need to interact with the environment and with people, they need to perfectly know where they are and what is happening around them. To this end, applications like SLAM and object/people tracking become fundamental. Such applications are typically based on vision and depth sensors such as laser scanners and depth cameras: high quality data is mandatory to have good results.

In Chapter 3 we proposed a novel, minimal solution to recover the axis of rotation of an actuated 2D laser range finder. An actuated 2D laser scanner allows us to have a reliable 3D view of the environment at a low cost. We showed that 4 checkerboards are sufficient to obtain a good 3D reconstruction of the scanned scene and, in Chapter 5, that it is possible to combine these data with a color image to obtain an RGB-D cloud of the whole scene.

In Chapter 4 we described a calibration procedure for Kinect-like depth sensors. We firstly analyzed the error on the depth measurements of such kind of sensors and then derived a two-step algorithm for reducing this error. In the first step, a flat wall is used to compute, for each pixel, an undistortion function to recover the real shape of the framed scene. In the second step, instead, an RGB camera and a checkerboard are used in conjunction with the depth sensor to estimate both a correction map for the wrong average depth and the relative pose between camera and depth sensor. Tests against the real shape (a plane) and distance (measured by means of two laser meters), showed that the described approach is able to correctly deform a point cloud and make it match the expected one. Moreover, the results of the pose estimation were coherent with the factory-provided information and similar to those of other state-of-the-art approaches.

The main drawback of the presented approach is that a large flat wall is needed to compute the correction maps. As soon as possible we will compare our calibration results with the ones obtained with similar applications. In the future we can envision to perform the calibration using natural scenes as in [56]. Finally, it is worth to notice that the presented algorithm has been released as an open source ROS package[1].

In Chapter 5 a calibration package for networks of depth sensors and cameras is described. The main novelties of this package are its distributed architecture, that allows the calibration of a dozen sensors (at least) in real-time, and a non-linear-optimization-based continuous refinement of the estimated sensor poses. Real-world as well as simulation tests demonstrated that the presented approach correctly estimates all the sensor poses. The main drawback is that camera and depth sensor intrinsic parameters need to be estimated beforehand to let the system work. The package has been released as open source in a GitHub repository[2] and as part of a bigger project, OpenPTrack[3], an open source project to create a scalable, multi-camera solution for person tracking. At the moment, the presented application deals with standard cameras and Kinect-like depth sensors. In the future we envision to improve its capabilities by adding new sensors: 2D laser scanners (for which a new calibration algorithm has to be implemented), ToF cameras and, why not, robot-specific sensors such as IMUs.

In conclusion we can state that the here-presented works have demonstrated that it is possible to improve the data provided by depth sensors, even though much needs to be done still.

[1] `https://github.com/iaslab-unipd/rgbd_calibration`
[2] `https://github.com/iaslab-unipd/calibration_toolkit`
[3] `http://openptrack.org`

Appendices

Appendix A

Walk-through: Calibration of a Depth Sensor-Camera Network

A.1 Package Dependencies

The calibration package is freely available cloning the GitHub repository:

https://github.com/iaslab-unipd/calibration_toolkit.

The package is developed in C++11 and apart from ROS [42], the main dependencies of the software are Eigen 3.2 [21], OpenCV 2.4 [9], PCL 1.7 [44] and Ceres Solver 1.10 [1], a library to solve non-linear least squares problems.

Most of these libraries are available in the Ubuntu 14.04 repositories and easily installable. Unfortunately, Ceres Solver 1.10 is missing. Actually, in Ubuntu 14.04 it is possible to install version 1.8 of Ceres Solver, contained in the apt package `libceres-dev`, however, because of some bugs (in the library), our code does not compile. To overcome this issue, we have prepared a script to download the latest tested version and install it. Just type

```
roscd calibration_toolkit/../scripts
./install_ceres.sh
```

on a terminal, and Ceres Solver as well as its dependencies will be installed on your system.

Before beginning with the example, we need to recall that the algorithm assumes that the intrinsic parameters of all the sensors have been already estimated, and expects that each camera driver publishes its own calibration parameters within a `sensor_msgs/CameraInfo` message.

A.2 Environment Configuration

In order to allow communication between nodes in different computers, the environment variable ROS_MASTER_URI on every client PC must be set to the IP address of the PC where the master node is launched, namely the *master PC*. Additionally, the ROS_IP and ROS_PC_NAME environment variables must be set. This can be done temporarily by typing

```
export ROS_MASTER_URI=http://<MASTER_IP>:11311/
export ROS_IP=<MACHINE_IP>
export ROS_PC_NAME=<MACHINE_NAME>
```

on a terminal. Note that the PC names assigned can be whatever the user wants, not necessarily the real names of the PCs. To set them definitively, they can be appended to the .bashrc file in the user's home folder. As an example:

```
echo "export ROS_MASTER_URI=http://192.168.1.1:11311/
export ROS_IP=192.168.1.5
export ROS_PC_NAME=Phoenix" >> ~/.bashrc
```

A.3 Calibration of a Network of Two Cameras

Suppose that all the PCs are configured as explained in Section A.2, and that we want to calibrate a network of two cameras connected to two different PCs that are called, respectively, Phoenix and Gemini. The multi-sensor calibration is performed by running a *master node* in the master PC, in our case Lyra, and a device driver in every PC attached to a device (one driver node for each device). First of all, we have to run the ROS drivers for all our devices. As an example, for a PointGrey camera, type on a terminal:

```
roslaunch pointgrey_camera_driver camera.launch
```

Then, we have to wrap these drivers in our calibration environment so that they can communicate with the master node in Lyra. To this aim, we run on both Phoenix and Gemini:

```
roslaunch multisensor_calibration camera_node.launch \
camera_name:=camera image_topic:=/camera/image \
camera_info_topic:=/camera/camera_info
```

where image_topic and camera_info_topic are the real topics on which the camera drivers publish their data. If everything is fine, we will see in our terminal:

```
[/Gemini/camera_node] All messages received.
[/Gemini/camera_node/get_device_info] Service started.
[/Gemini/camera_node/extract_checkerboard] Action server
    started.
```

At this point, `Phoenix` and `Gemini` are working as expected: we can move to `Lyra`. We create a file called `network.yaml` in the directory `multisensor_calibration/conf`, open our favorite text editor, and fill the file with the names of two PCs and the correspondent cameras as below:

```
# Network configuration
network:
  - pc: "Phoenix"
    devices: ["camera"]
  - pc: "Gemini"
    devices: ["camera"]
```

Then, we define the calibration pattern, i.e. the checkerboard, that we will use during the calibration procedure. We create a file named `checkerboard.yaml` in the directory `multisensor_calibration/conf` and fill it with our checkerboard parameters:

```
# Checkerboard configuration
checkerboard:
  cols: 6
  rows: 5
  cell_width: 0.12
  cell_height: 0.12
```

We can now start the calibration procedure. We type:

```
roslaunch multisensor_calibration master_node.launch
```

on a terminal. If all the nodes are launched correctly we'll have an output similar to the one below:

```
[/Lyra/master_node] Connected to [/Phoenix/camera_node/
   get_device_info] service.
[/Lyra/master_node] Connected to [/Gemini/camera_node/
   get_device_info] service.
[/Lyra/master_node] Connected to [/Phoenix/camera_node/
   extract_checkerboard] action server.
[/Lyra/master_node] Connected to [/Gemini/camera_node/
   extract_checkerboard] action server.
[/Lyra/master_node] Getting device infos...
[/Lyra/calibration] Sensor [/Phoenix/camera] added.
[/Lyra/calibration] Sensor [/Gemini/camera] added.
[/Lyra/master_node] Initialization complete.
```

We can then start the data acquisition phase: we take the checkerboard and move it around letting all the sensors see it. To acquire the images from every sensor, we publish an empty message on the topic `/Lyra/master_node/acquisition`:

```
rostopic pub /Lyra/master_node/acquisition std_msgs/Empty -1
```

Note that ROS let us publish a message at a fixed rate, we just need to substitute `-1` with the desired rate `-r <rate>`. In this case, we must pay attention not to move the checkerboard too quickly, to avoid blur and calibration errors due to the non perfect time-synchronization of the sensors.

We can monitor the whole calibration procedure via Rviz (Figure A.1). It is sufficient to set the `world` frame as the fixed frame, add the *tf* view and a *marker* view on topic `/Lyra/master_node/markers` and every time the checkerboard is detected or a sensor pose is estimated we will see it on the screen. In Figure A.1 some screenshots acquired during the calibration are shown. Before terminating the calibration, we position the checkerboard on the floor and publish:

```
rostopic pub /Lyra/master_node/action std_msgs/String "begin
   plane" -1
```

From now on, the calibration algorithm assumes that all the checkerboards are lying on the same plane (see Section 5.2.3). We perform some acquisitions with the checkerboard lying on the floor and then publish an `end plane` instruction:

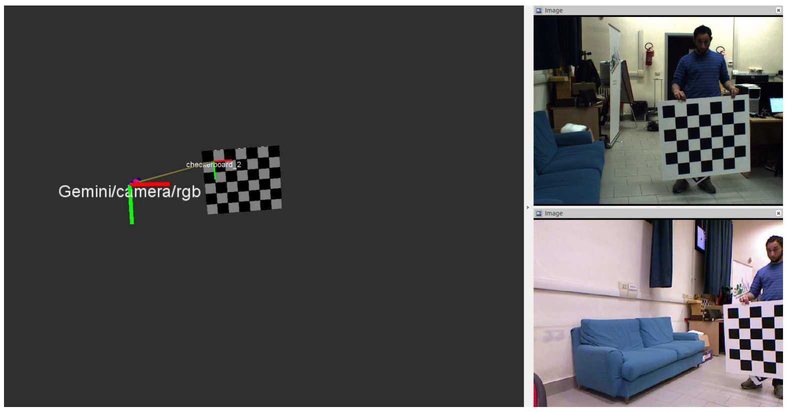
```
rostopic pub /Lyra/master_node/action std_msgs/String "end
   plane" -1
```

Finally, to get the results of the calibration, we use the service `get_results` offered by the master node. On a terminal we run:

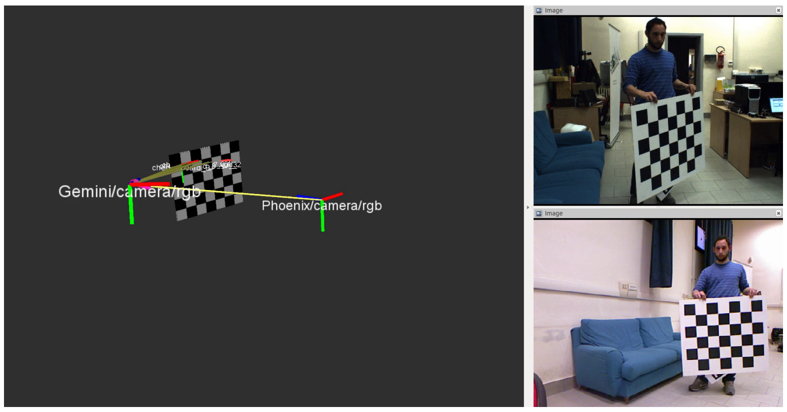
```
rosservice call /Lyra/master_node/get_results
```

and get the estimated poses, similar to the ones below:

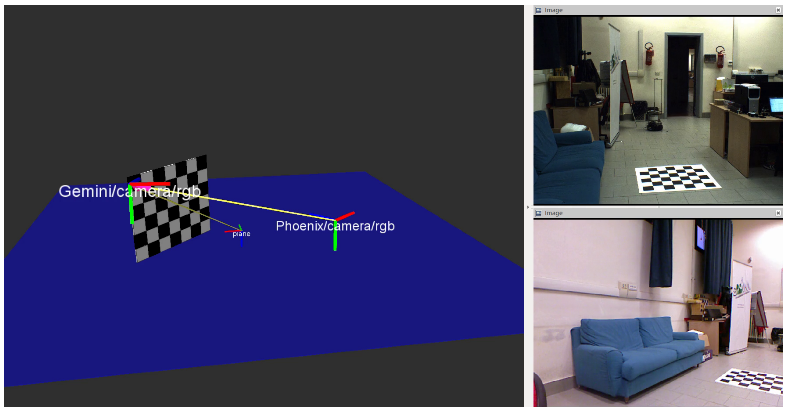
```
poses:
  - frame_id: "/world"
    child_frame_id: "/Gemini/camera"
    pose:
      position: {x: 0.0, y: 0.0, z: 0.0}
      orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}
  - frame_id: "/world"
    child_frame_id: "/Phoenix/camera"
    pose:
      position: {x: 1.12064, y: 0.321081, z: 0.565662}
      orientation: {x: 0.0471913, y: -0.377825, z: -0.0340694,
         w: 0.924046}
```



(a) First sensor visualized in Rviz with the checkerboard.



(b) Second sensor visualized in Rviz with the checkerboard.



(c) Plane visualized in Rviz (the checkerboard is not visualized).

Figure A.1: Screenshots acquired during the calibration procedure. (a) As soon as one sensor detects the checkerboard pattern, it becomes part of the `tf` tree and its pose is published. (b) Then, every sensor that sees the checkerboard is added to the tree. (c) If the program is asked to estimate a plane, its pose is published and can be visualized using Rviz.

Bibliography

[1] S. Agarwal, K. Mierle, et al. *Ceres Solver*. http://ceres-solver.org.

[2] H. Andreasson. "Local Visual Feature Based Localisation and Mapping by Mobile Robots". PhD thesis. 2008.

[3] E. Auvinet, J. Meunier, and F. Multon. "Multiple Depth Cameras Calibration and Body Volume Reconstruction for Gait Analysis". In: *Information Science, Signal Processing and their Applications (ISSPA), 2012 11th International Conference on*. 2012, pp. 478–483.

[4] F. Basso, R. Levorato, and E. Menegatti. "Online Calibration for Networks of Cameras and Depth Sensors". In: *Proceedings of the 12th Workshop on Non-classical Cameras, Camera Networks and Omnidirectional Vision (OMNIVIS)*. Hong Kong, China, 2014.

[5] F. Basso, R. Levorato, M. Munaro, and E. Menegatti. "A Distributed Calibration Algorithm for Color and Range Camera Networks". In: ed. by A. Koubaa. Studies in Systems, Decision and Control. (To appear). Springer.

[6] F. Basso, M. Munaro, S. Michieletto, and E. Menegatti. "Fast and Robust Multi-People Tracking from RGB-D Data for a Mobile Robot". In: *Proceedings of the 12th Intelligent Autonomous Systems (IAS) Conference*. Vol. 193. Jeju Island, Korea, 2012, pp. 265–276. DOI: 10.1007/978-3-642-33926-4_25.

[7] F. Basso, A. Pretto, and E. Menegatti. "Unsupervised Intrinsic and Extrinsic Calibration of a Camera-Depth Sensor Couple". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. Hong Kong, China, 2014, pp. 6244–6249. DOI: 10.1109/ICRA.2014.6907780.

[8] P. Ben-Tzvi, S. Charifa, and M. Shick. "Extraction of 3D Images Using Pitch-Actuated 2D Laser Range Finder for Robotic Vision". In: *Robotic and Sensors Environments (ROSE), 2010 IEEE International Workshop on*. 2010, pp. 1–6. DOI: 10.1109/ROSE.2010.5675302.

[9] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[10] A. Canessa, M. Chessa, A. Gibaldi, S. P. Sabatini, and F. Solari. “Calibrated Depth and Color Cameras for Accurate 3D Interaction in a Stereoscopic Augmented Reality Environment”. In: *Journal of Visual Communication and Image Representation* 25.1 (2014), pp. 227–237. DOI: 10.1016/j.jvcir.2013.02.011.

[11] D. Cox, J. Little, and D. O’Shea. *Ideals, Varieties, and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer New York, 2007. DOI: 10.1007/978-0-387-35651-8.

[12] C. Dal Mutto, P. Zanuttigh, and G. M. Cortelazzo. *Time-of-Flight Cameras and Microsoft Kinect*. Springer Publishing Company, 2012.

[13] M. Di Cicco, L. Iocchi, and G. Grisetti. “Non-Parametric Calibration for Depth Sensors”. In: *Proc. of the 13th International Conference on Intelligent Autonomous Systems. (IAS-13)*. Padova, Italy, 2014.

[14] A. Diosi and L. Kleeman. “Laser Scan Matching in Polar Coordinates with Application to SLAM”. In: *Intelligent Robots and Systems (IROS). 2005 IEEE/RSJ International Conference on*. 2005, pp. 3317–3322. DOI: 10.1109/IROS.2005.1545181.

[15] D. Fiedler and H. Müller. “Impact of Thermal and Environmental Conditions on the Kinect Sensor”. In: *Advances in Depth Image Analysis and Applications*. Ed. by X. Jiang, O. R. P. Bellon, D. Goldgof, and T. Oishi. Vol. 7854. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 21–31. DOI: 10.1007/978-3-642-40303-3_3.

[16] M. A. Fischler and R. C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Communications of the ACM* 24.6 (1981), pp. 381–395. DOI: 10.1145/358669.358692.

[17] A. Fod, A. Howard, and M. Mataric. “A laser-based people tracker”. In: *Robotics and Automation (ICRA). 2002 IEEE International Conference on*. Vol. 3. 2002, pp. 3024–3029. DOI: 10.1109/ROBOT.2002.1013691.

[18] P. Furgale, J. Rehder, and R. Siegwart. “Unified Temporal and Spatial Calibration for Multi-Sensor Systems”. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. 2013, pp. 1280–1286. DOI: 10.1109/IROS.2013.6696514.

[19] B. Furht. *Encyclopedia of multimedia*. Springer US, 2008.

[20] A. Gritti, O. Tarabini, J. Guzzi, G. Di Caro, V. Caglioti, L. Gambardella, and A. Giusti. “Kinect-based People Detection and Tracking from Small-Footprint Ground Robots”. In: *Intelligent Robots and Systems (IROS). 2014 IEEE/RSJ International Conference on*. 2014, pp. 4096–4103. DOI: 10.1109/IROS.2014.6943139.

[21] G. Guennebaud, B. Jacob, et al. *Eigen*. http://eigen.tuxfamily.org. 2010.

[22] J. Han, L. Shao, D. Xu, and J. Shotton. “Enhanced Computer Vision With Microsoft Kinect Sensor: A Review”. In: *Cybernetics, IEEE Transactions on* 43.5 (2013), pp. 1318–1334. DOI: 10.1109/TCYB.2013.2265378.

[23] D. Herrera C., J. Kannala, and J. Heikkilä. “Accurate and Practical Calibration of a Depth and Color Camera Pair”. In: *Computer Analysis of Images and Patterns*. Ed. by P. Real, D. Diaz-Pernil, H. Molina-Abril, A. Berciano, and W. Kropatsch. Vol. 6855. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 437–445. DOI: 10.1007/978-3-642-23678-5_52.

[24] D. Herrera C., J. Kannala, and J. Heikkilä. “Joint Depth and Color Camera Calibration with Distortion Correction”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 34.10 (2012), pp. 2058–2064. DOI: 10.1109/TPAMI.2012.125.

[25] G. J. Iddan and G. Yahav. “Three-dimensional Imaging in the Studio and Elsewhere”. In: *Proceedings SPIE*. Vol. 4298. International Society for Optics and Photonics. 2001, pp. 48–55. DOI: 10.1117/12.424913.

[26] C. Kerl, J. Sturm, and D. Cremers. “Dense Visual SLAM for RGB-D Cameras”. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. 2013, pp. 2100–2106. DOI: 10.1109/IROS.2013.6696650.

[27] K. Khoshelham and S. O. Elberink. “Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications”. In: *Sensors* 12.2 (2012), pp. 1437–1454.

[28] Y. M. Kim, D. Chan, C. Theobalt, and S. Thrun. “Design and Calibration of a Multi-view ToF Sensor Fusion System”. In: *Computer Vision and Pattern Recognition Workshops (CVPRW). 2008 IEEE Computer Society Conference on*. 2008, pp. 1–7.

[29] Q. Le and A. Ng. “Joint Calibration of Multiple Sensors”. In: *Intelligent Robots and Systems (IROS). 2009 IEEE/RSJ International Conference on*. 2009, pp. 3651–3658.

[30] R. Levorato and E. Pagello. "Probabilistic 2D Acoustic Source Localization Using Direction of Arrivals in Robot Sensor Networks". In: *Simulation, Modeling, and Programming for Autonomous Robots*. Ed. by D. Brugali, J. F. Broenink, T. Kroeger, and B. A. MacDonald. Vol. 8810. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 474–485. DOI: 10.1007/978-3-319-11900-7_40.

[31] G. Li, Y. Liu, L. Dong, X. Cai, and D. Zhou. "An Algorithm for Extrinsic Parameters Calibration of a Camera and a Laser Range Finder using Line Features". In: *Intelligent Robots and Systems (IROS). 2007 IEEE/RSJ International Conference on*. 2007, pp. 3854–3859. DOI: 10.1109/IROS.2007.4399041.

[32] M. Luber, L. Spinello, and K. Arras. "People Tracking in RGB-D Data with On-line Boosted Target Models". In: *Intelligent Robots and Systems (IROS). 2011 IEEE/RSJ International Conference on*. 2011, pp. 3844–3849. DOI: 10.1109/IROS.2011.6095075.

[33] J. M. McCarthy and G. S. Soh. *Geometric Design of Linkages*. Vol. 11. Interdisciplinary Applied Mathematics. Springer-Verlag New York, 2011. DOI: 10.1007/978-1-4419-7892-9.

[34] C. Mei and P. Rives. "Calibration Between a Central Catadioptric Camera and a Laser Range Finder for Robotic Applications". In: *Robotics and Automation (ICRA). 2006 IEEE International Conference on*. 2006, pp. 532–537. DOI: 10.1109/ROBOT.2006.1641765.

[35] A. Mendes, L. Bento, and U. Nunes. "Multi-target Detection and Tracking with a Laser Scanner". In: *Intelligent Vehicles Symposium*. 2004, pp. 796–801. DOI: 10.1109/IVS.2004.1336486.

[36] M. Munaro, F. Basso, and E. Menegatti. "Tracking People within Groups with RGB-D Data". In: *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*. Vilamoura, Portugal, 2012, pp. 2101–2107. DOI: 10.1109/IROS.2012.6385772.

[37] M. Munaro, F. Basso, S. Michieletto, E. Pagello, and E. Menegatti. "A Software Architecture for RGB-D People Tracking Based on ROS Framework for a Mobile Robot". In: *Frontiers of Intelligent Autonomous Systems*. Ed. by S. Lee, K.-J. Yoon, and J. Lee. Vol. 466. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2013, pp. 53–68. DOI: 10.1007/978-3-642-35485-4_5.

[38] M. Munaro, A. Horn, R. Illum, J. Burke, and R. B. Rusu. “OpenPTrack: People Tracking for Heterogeneous Networks of Color-Depth Cameras”. In: *IAS-13 Workshop Proceedings: 1st Intl. Workshop on 3D Robot Perception with Point Cloud Library*. Padova, Italy, 2014, pp. 235–247.

[39] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. “KinectFusion: Real-time Dense Surface Mapping and Tracking”. In: *Mixed and Augmented Reality (ISMAR). 10th IEEE International Symposium on*. 2011, pp. 127–136. DOI: `10.1109/ISMAR.2011.6092378`.

[40] P. Newman, D. Cole, and K. Ho. “Outdoor SLAM using Visual Appearance and Laser Ranging”. In: *Robotics and Automation (ICRA). 2006 IEEE International Conference on*. 2006, pp. 1180–1187. DOI: `10.1109/ROBOT.2006.1641869`.

[41] V. Pradeep, K. Konolige, and E. Berger. “Calibrating a Multi-arm Multi-sensor Robot: A Bundle Adjustment Approach”. In: *Experimental Robotics*. Ed. by O. Khatib, V. Kumar, and G. Sukhatme. Vol. 79. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, 2014, pp. 211–225. DOI: `10.1007/978-3-642-28572-1_15`.

[42] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. “ROS: an Open-source Robot Operating System”. In: *ICRA Workshop on Open Source Software*. 2009.

[43] C. Raposo, J. P. Barreto, and U. Nunes. “Fast and Accurate Calibration of a Kinect Sensor”. In: *3D Vision (3DV). 2013 International Conference on*. 2013, pp. 342–349. DOI: `10.1109/3DV.2013.52`.

[44] R. B. Rusu and S. Cousins. “3D is here: Point Cloud Library (PCL)”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011, pp. 1–4.

[45] D. Scaramuzza, A. Harati, and R. Siegwart. “Extrinsic Self Calibration of a Camera and a 3D Laser Range Finder from Natural Scenes”. In: *Intelligent Robots and Systems (IROS). 2007 IEEE/RSJ International Conference on*. 2007, pp. 4164–4169. DOI: `10.1109/IROS.2007.4399276`.

[46] D. Schulz, W. Burgard, D. Fox, and A. B. Cremers. “People Tracking with Mobile Robots using Sample-based Joint Probabilistic Data Association Filters”. In: *The International Journal of Robotics Research* 22.2 (2003), pp. 99–116. DOI: `10.1177/0278364903022002002`.

[47] J. M. Selig. *Geometric Fundamentals of Robotics*. Monographs in Computer Science. Springer New York, 2005. DOI: `10.1007/b138859`.

[48] J. M. Selig. “On the Geometry of Point-Plane Constraints on Rigid-Body Displacements”. In: *Acta Applicandae Mathematicae* 116.2 (2011), pp. 133–155. DOI: 10.1007/s10440-011-9634-6.

[49] J. Smisek, M. Jancosek, and T. Pajdla. “3D with Kinect”. In: *Computer Vision Workshops (ICCV Workshops), 2011. ICCVW 2011. IEEE International Conference on.* 2011, pp. 1154–1160.

[50] E. So, F. Basso, and E. Menegatti. “Calibration of a Rotating 2D Laser Range Finder using Point-Plane Constraints”. In: *Journal of Automation, Mobile Robotics & Intelligent Systems* 7.2 (2013), pp. 30–38.

[51] E. So and E. Menegatti. “A Unified Approach to Extrinsic Calibration Between a Camera and a Laser Range Finder using Point-Plane Constraints”. In: *Proceedings of the 1st International Workshop on Perception for Mobile Robots Autonomy.* 2012.

[52] A. N. Staranowicz, G. R. Brown, F. Morbidi, and G. L. Mariottini. “Practical and Accurate Calibration of RGB-D Cameras using Spheres”. In: *Computer Vision and Image Understanding* 137 (2015), pp. 102–114. DOI: 10.1016/j.cviu.2015.03.013.

[53] A. Staranowicz, G. R. Brown, F. Morbidi, and G. L. Mariottini. “Easy-to-Use and Accurate Calibration of RGB-D Cameras from Spheres”. In: *Image and Video Technology.* Ed. by R. Klette, M. Rivera, and S. Satoh. Vol. 8333. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 265–278. DOI: 10.1007/978-3-642-53842-1_23.

[54] H. Surmann, K. Lingemann, A. Nüchter, and J. Hertzberg. “A 3D Laser Range Finder for Autonomous Mobile Robots”. In: *Proceedings of the 32nd International Symposium on Robotics (ISR).* Vol. 19. 21. 2001, pp. 153–158.

[55] H. Surmann, A. Nüchter, and J. Hertzberg. “An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments”. In: *Robotics and Autonomous Systems* 45.3–4 (2003), pp. 181–198. DOI: 10.1016/j.robot.2003.09.004.

[56] A. Teichman, S. Miller, and S. Thrun. “Unsupervised Intrinsic Calibration of Depth Sensors via SLAM”. In: *Proceedings of Robotics: Science and Systems.* Berlin, Germany, 2013.

[57] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. “Bundle Adjustment – A Modern Synthesis”. In: *Vision Algorithms: Theory and Practice.* Ed. by B. Triggs, A. Zisserman, and R. Szeliski. Vol. 1883. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, pp. 298–372. DOI: 10.1007/3-540-44480-7_21.

[58] R. Unnikrishnan and M. Hebert. *Fast Extrinsic Calibration of a Laser Rangefinder to a Camera.* Tech. rep. Carnegie Mellon University, 2005.

[59] C. W. Wampler. "Locating N points of a rigid body on N given planes". In: *ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.* American Society of Mechanical Engineers. 2004, pp. 513–521.

[60] C. W. Wampler. "On a Rigid Body Subject to Point-Plane Constraints". In: *Journal of Mechanical Design* 128.1 (2006), pp. 151–158.

[61] M. Warren, D. McKinnon, and B. Upcroft. "Online Calibration of Stereo Rigs for Long-Term Autonomy". In: *Robotics and Automation (ICRA). 2013 IEEE International Conference on.* Karlsruhe, Germany, 2013.

[62] S. Wasielewski and O. Strauss. "Calibration of a Multi-Sensor System Laser Rangefinder/Camera". In: *Intelligent Vehicles '95 Symposium. Proceedings of the.* 1995, pp. 472–477. DOI: 10.1109/IVS.1995.528327.

[63] J. Weingarten. "Feature-based 3D SLAM". PhD thesis. École Polytechnique Fédérale de Lausanne, 2006.

[64] T. Whelan, M. Kaess, J. Leonard, and J. McDonald. "Deformation-based Loop Closure for Large Scale Dense RGB-D SLAM". In: *Intelligent Robots and Systems (IROS). 2013 IEEE/RSJ International Conference on.* 2013, pp. 548–555. DOI: 10.1109/IROS.2013.6696405.

[65] W. Xiang, C. Conly, C. D. McMurrough, and V. Athitsos. "A Review and Quantitative Comparison of Methods for Kinect Calibration". In: *Proceedings of the 2nd International Workshop on Sensor-based Activity Recognition and Interaction (WOAR).* Rostock, Germany, 2015. DOI: 10.1145/2790044.2790056.

[66] S. Zennaro, M. Munaro, S. Milani, P. Zanuttigh, A. Bernardi, S. Ghidoni, and E. Menegatti. "Performance Evaluation of the 1st and 2nd Generation Kinect for Multimedia Applications". In: *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME).* Torino, Italy, 2015.

[67] C. Zhang and Z. Zhang. "Calibration between Depth and Color Sensors for Commodity Depth Cameras". In: *Multimedia and Expo (ICME), 2011 IEEE International Conference on.* 2011, pp. 1–6. DOI: 10.1109/ICME.2011.6012191.

[68] L. Zhang and B. Ghosh. "Line Segment Based Map Building and Localization Using 2D Laser Rangefinder". In: *Robotics and Automation (ICRA). 2000 IEEE International Conference on.* Vol. 3. 2000, pp. 2538–2543. DOI: 10.1109/ROBOT.2000.846410.

[69] Q. Zhang and R. Pless. “Extrinsic Calibration of a Camera and Laser Range Finder (Improves Camera Calibration)”. In: *Intelligent Robots and Systems (IROS). 2004 IEEE/RSJ International Conference on*. Vol. 3. 2004, pp. 2301–2306. DOI: 10.1109/IROS.2004.1389752.

[70] Z. Zhang. “A Flexible New Technique for Camera Calibration”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.11 (2000), pp. 1330–1334.