UNIVERSIDADE D
COIMBRA

Hugo Correia Marques

3D Registration and Mapping of Forest Environments

Master's Dissertation in MIEEC, supervised by Dr. David Bina Siassipour Portugal and presented to Faculty of Science and Technology of the University of Coimbra

October 2020

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

3D Registration and Mapping of Forest Environments

Hugo Correia Marques

Coimbra, October 2020

UNIVERSIDADE D
COIMBRA

3D Registration and Mapping of Forest Environments

Supervisor:

Dr. David Bina Siassipour Portugal

Jury:

Prof. Dr. António Paulo Mendes Breda Dias Coimbra

Prof. Dr. Cristiano Premebida

Dr. David Bina Siassipour Portugal

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, October 2020

Acknowledgements

It is common to say that big goals are not accomplished alone. When we think that all our knowledge comes from the cluster of information that we retain in research, articles, books and many others, and that this knowledge comes from other people, we understand that we are not alone in that way. However, these adventures are also an open door for great challenges, frustrations, headaches and disappointments. When everything we think we know seems not to be enough, when all our brilliant ideas seem to be just ideas, when all our effort is not paid off at that moment, we start giving up. Although we were looking for answers in in the right places, we were not getting what we wanted. Then, in a simple conversation between colleagues, in a moment of carefree with those we like, we refresh our mind and we try again with another spirit. We start to see problems in other ways, we see solutions where we had never thought of and we take our goals forward when we no longer had hope. We realize that all that we needed was to socialize, debate, laugh with someone about something that tormented us, know how to help and receive help, we just needed to realize again that we were not alone.

I would like to acknowledge the various people who have helped me, in one way or another, during these last years.

First of all, I thank my parents, who unconditionally supported me in all situations, provided me with all the best conditions and always trusted in my effort and commitment.

I thank my girlfriend, Inês, whose love, care and support were essential in this journey. Regardless of my less positive moments, she always had some way to lift my head, make me laugh and make me happy.

I thank my supervisors, Dr. David Portugal and Dr. Gonçalo Martins, for always keeping me on track, for constant and precious advices, for giving me the opportunity to embrace this enriching mission and for having taught principles, both technically and in organization,

that I will take with me for the rest of my life. I also thank Dr. João Ferreira who, with his experience and knowledge, helped me in important questions that would take a lot of time to overcome.

I also give my thanks to my colleagues and friends, namely Leonardo Esteves, Gonçalo Monteiro, Tiago Baptista, Francisco Roque, João Santos, Rafael Carvalho and David Gomes, with whom we made a true friendship group with remarkable moments, and with which we formed a mutual help team that was able to overcome the challenges that came our way.

I am grateful to Duda Andrada and André Araújo for their willingness to help and for having been my point of assistance in the most difficult moments of this work.

Special thanks to my grandparents and D. Ilídio Pinto Leandro.

Resumo

À medida que aumenta o desenvolvimento de sistemas robóticos autónomos, aumenta também o interesse em usar robôs como uma solução viável e segura para trabalhos repetitivos, complicados, difíceis e perigosos. Dada a uma alta densidade florestal presente na maioria dos continentes, os incêndios florestais tendem a ser bastante devastadores e difíceis de combater, o que coloca em risco várias vidas humanas, fauna e flora. Desenvolvimentos recentes na área da Robótica florestal permitem aos robôs percecionar ambientes florestais desconhecidos, mapeando-os e produzindo informação útil a fim de detetar material potencialmente inflamável, podendo assim agir e prevenir a ocorrência de fogos florestais.

A Localização e Mapeamento Simultâneos (do Inglês SLAM - Simultaneous localization and mapping) permite dotar uma plataforma robótica móvel com a capacidade de construir uma representação do ambiente circundante e, simultaneamente, localizar-se nele.

A proliferação de novos sensores e o aumento das suas capacidades abriram uma vasta gama de possibilidades para o mundo tecnológico. No âmbito de operações florestais com robôs autónomos, a calibração e o registo multissensorial tornam-se fundamentais para a obtenção de informação útil e consistente sobre o ambiente.

Este trabalho tem como foco a solução desses problemas direcionados aos ambientes florestais, lidando com as suas adversidades e desafios. Propomos uma solução que visa calibrar e registar com alta precisão os sensores de um kit sensorial multimodal embutido num robô florestal de grande porte, e utilizá-los para mapear em tempo real o ambiente circundante com dados úteis.

Esta solução foi validada por meio de experiências previamente guardadas no mundo real, utilizando datasets, tendo demonstrado um desempenho apropriado, tanto em precisão quanto em consistência. No final, conseguiu-se registar com precisão os sensores presentes num sistema robótico e construir um mapa 3D denso do ambiente com o correto registo dos

diferentes dados de sensores.

Abstract

As the development of autonomous robotic systems increases, so does the interest in using robots as a viable and safe solution for repetitive, complicated, difficult and dangerous works. Given the high forest density present in most continents, forest fires may be quite devastating and difficult to fight, which puts several human lives, fauna and flora at risk. Recent developments in the field of Forest Robotics allow robots to perceive unknown forest environments, mapping them and producing useful information in order to detect potentially flammable material, thus being able to act and prevent the occurrence of forest fires.

Simultaneous localization and Mapping (SLAM) addresses the problem of providing a mobile robotic platform with the ability to build a representation of the surrounding environment and simultaneously localize itself in it.

The proliferation of sensors and their increased capabilities have opened up a huge range of possibilities for the technological world. In the realm of forestry operations with autonomous robots, calibration and multisensory registration are essential to acquire useful and consistent information about the environment.

This work focuses on the solution to these problems in forest environments, addressing the challenges involved. We propose a solution that seeks to calibrate and register the sensors of a multimodal sensory kit embedded in a large heavy-duty forestry robot with high precision and use them to map the surrounding environment with useful information in real time.

The solution was validated through experiments with real-world pre-recorded datasets, and it has shown appropriate performance, both in accuracy and consistency. In the end, we were able to accurately register the sensors present in the robotic system with the developed method, and we were able to build a dense 3D map of the environment with the correct registration of the different sensor data.

"That's been one of my mantras - focus and simplicity. Simple can be harder than complex: you have to work hard to get your thinking clean to make it simple. But it's worth it in the end because once you get there, you can move mountains."

— Steve Jobs

Contents

List of Acronyms

A-LOAM	Advanced LOAM
BRIEF	Binary Robust Independent Elementary Feature
EKF	Extended Kalman Filter
EM	Expectation Maximisation
FAST	Features from Accelerated Segment Test
FOV	Field Of View
GPS	Global Positioning System
ICP	Iterative Corresponding Point
KF	Kalman Filter
LeGO-LOAM	Lightweight and Ground-OptimizedHandling Odometry and Mapping
LIDAR	Light Detection And Ranging
LOAM	LIDAR Odometry and Mapping
MoDSeM	Modular Framework for Distributed Semantic Mapping
MSER	Maximally Stable Extremal Regions
OpenCV	Open Source Computer Vision Library
ORB	Oriented FAST and Rotated BRIEF
PCBR	Principal Curvature-Based Region
PM	Perception Modules

RMSE	Root Mean Square Error
ROS	ROS: Robot Operating System
RTAB-Map	Real-Time Appearance-Based Mapping
RTK	Real Time Kinematic
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous localization and Mapping
STM	Short Term Memory
SURF	Speeded-up Robust Features
UC	University of Coimbra
V-LOAM	Visual LIDAR Odometry and Mapping

List of Figures

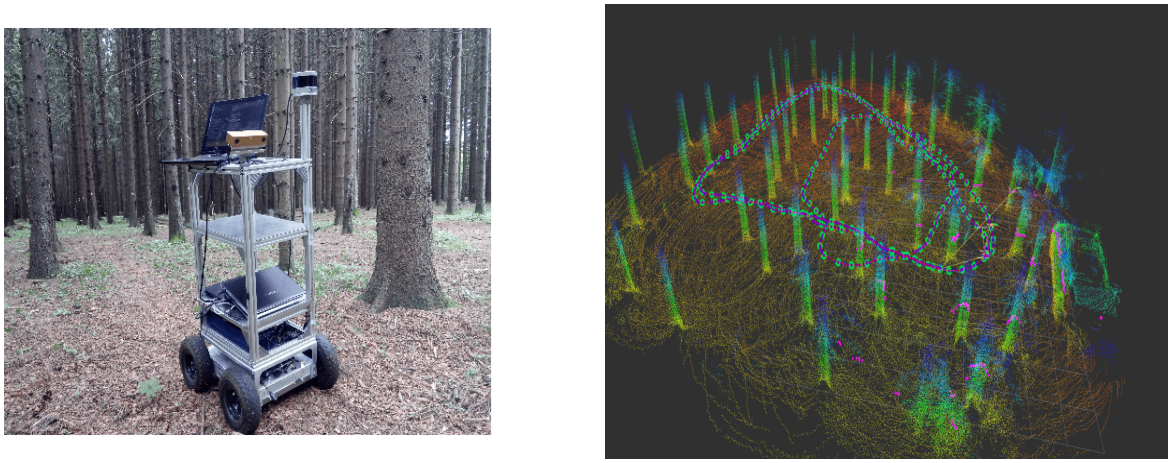
List of Tables

1 Introduction

In recent years, one of the most reported and most relevant catastrophes in different continents such as Europe, America, Africa and most recently Australia, have to do with large-scale fires. Although 84% of these fires are caused by humans, either by irresponsible burning or due to arsonists, there is still a proportion of 16% where the large amount of combustible material in forests is the main factor to start a wildfire [1]. Even in cases where these wildfires do not originate from combustible materials, early cleaning of forests can greatly reduce the wildfire spread and thus favor their combat.

The forest area on all these continents is so large (around 30% of the land area) [2] that all available human resources are not enough to prevent every outburst. Thus, specialized robots moving in a forest environment, clearing the forest, mapping the surroundings and locating possible flammable objects that could cause a large fire, are a focus area to prevent deforestation and devastating fires. Robots can be the basis of the solution because they can tackle dangerous and repetitive problems in a more effective way while safeguarding human lives. They can help us in the detection of potential issues and solve them, allow better monitoring of the forest environment by mapping less known locations, which can be determinant to assist the fire fighting organizations in order to make a more direct and effective combat. With this, many lives of professionals and civilians can be saved and fauna and flora may be preserved.

With this in mind, accurately mapping the environment with the help of a robotic system has been a much investigated area and several methods have been developed [3] [4] [5]. However, for the data to be coherent the entire system needs to be well registered, i.e. we need to know the precise relationship between all the data acquired by the different sensors. Therefore, the calibration and registration processes are extremely important for any system that uses a multimodal sensory kit such as in autonomous driving vehicles or the smartphone industry. At first, because it allows us to determine the unique parameters of all devices and to know the relationship between the real world and the data produced by them. Then



(a) A robot operating in a forest environment.

(b) 3D map of forest.

Figure 1.1: An example of a forest scenario where robots operate and a map that they are capable of producing. Images from [6].

because it allows us to relate all the data from different sensors to the same referential system, building a representation of the real world by useful and precise information.

For outdoor field robotics there are many challenges that indoor environments do not have. Some of the challenges include, for example, uncertain paths, non-ideal environmental conditions, lack of geometric features and depth information, presence of people, animals or even plant species that need to be preserved [6]. Mapping forests is one of the cases that regularly presents all these challenges, giving even more importance to a coherent set of data that can be easily interpreted by the robot in order to avoid human, monetary or any kind of damage. The Figure 1.1 shows some of the adversities present in forests such as the irregular paths and the high density of trees. We can also see the importance of mapping, allowing us to easily understand the morphology of the place where the robot passed and to distinguish elements like trees, paths or roads, branches, leaves and cars.

Light Detection And Ranging (LIDAR) have been receiving a lot of attention towards new technologies. Its usefulness and versatility are the main factors to be entering large markets such as mobile phones and the automobile industry. The high precision point clouds, which are not easily affected by adverse conditions, provide complement to technologies like 3D reconstruction, autonomous driving, 3D mapping, calibration and registration. It is evident understandable that they are essential devices in any robotic system that intends to autonomously map complex unstructured environments, just like ours. The advantages of LIDARs stem from the provision of detailed 3D Point Clouds, which can be manipulated and processed while also providing useful data, namely distances, intensity of light reflections

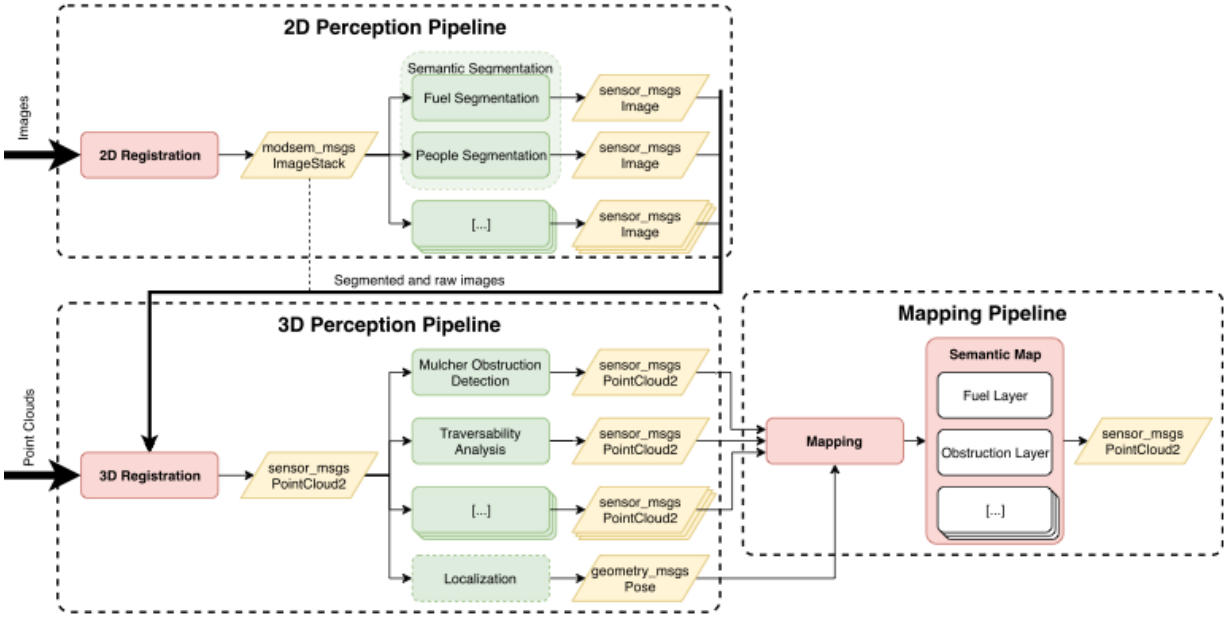


Figure 1.2: An overview of the SEMFIRE framework data flow.

and the geometry of the environment that often allows to distinguish types of objects. The interest in this type of data promoted the development of many works that produce point clouds through camera images.

SEMFIRE[1] is a collaboration project between the University of Coimbra (UC), Ingeniarius Ltd and Sfera Ultimate Ltd. In order to push forward the development of field robotics, SEMFIRE focuses on providing an autonomous robot with the ability to operate in the forest to clear flammable objects. Among others, the robot should be able to map and navigate within the forestry environment where it operates, detect flammable material and generally support human forest maintenance teams. SEMFIRE implies cooperative perception between a large heavy-duty UGV and a team of lightweight UAVs, where each member contributes to the global knowledge of the system by sharing and cooperatively processing data and percepts from one another, combining the sensory abilities, perspectives and processing power of various agents to achieve better results. The Modular Framework for Distributed Semantic Mapping (MoDSeM) proposed in [7] aggregates all this information to create a semantic map that can be shared among all system agents. Within the existing perception architecture developed in the context of the SEMFIRE project, that is illustrated in Figure 1.2, the work described in this dissertation tackles 3D registration and mapping issues which are represented in the red blocks.

The main goals of this work are:

[1] `https://semfire.ingeniarius.pt/`

1. Calibrate and register a set of cameras and sensors embedded in a robot sensor kit.

2. Build a 3D map in real time using data from different sensors such as 3D LIDARs, RGB-D or monocular cameras, with useful information about the real world environment.

This Dissertation is organized in five main chapters. In Chapter 2 we cover basic knowledge and fundamentals of registration, mapping, calibration, projection and ROS, which are essential for understanding the entire document. We also do an overview of the most popular and used approaches for both registration and mapping; in Chapter 3 we present the methods and techniques that we used for calibration and registration of pairs of sensors, including LIDAR with RGB-D cameras and LIDAR with monocular cameras. Following the pairwise calibration results, we also present the method used to merge LIDAR and camera data; then in Chapter 4 we present the apparatus used and the detailed experimental validation. We discuss the 3D mapping method used and the rationale for choosing that approach. Then results and their validation are presented; finally, in Chapter 5 we reflect about the advantages and handicaps of our methods, as well as on possible future work.

2 Background and State of the Art

2.1 Registration

2.1.1 What is Registration?

Registration is the process of finding the relation between different types of data or devices into a single coordinate system. This process is essential for complex systems because it enables the production of a robust and coherent result through data combination that comes from different sensors. It is a process widely used in object recognition, object reconstruction, mapping and localization [8].

Registration is an area of much study and development which still has several problems and challenges to overcome [9] [10]. One of them is the high variety of devices and data types which have completely different features and specifications, making it very hard to find viable ways to relate and merge all of them together.

In most environments, practically all members (like trees, stones, roads, buildings) are rigid bodies and so their shape and size remains unchanged throughout the time. This members can be related through geometric transformations that are called *affine transformations* [11]. Affine transformations preserve lines and parallelism but not necessarily distances and angles. However, they also preserve the euclidean distance between every pair of points. With that, we can say that two images of the same environment, taken in different scenarios and by different cameras, have a very high possibility of having matching pairs of points which allow us to estimate the transformation that relates both images. This factor is very important for registration as it makes it possible to match two images through easily detected images characteristics like contours, boundaries, colors and corners of the same rigid body.

Briefly, the main goal of registration is to find the rotation and translation matrices that minimizes the distance between corresponding data from different sources. We can consider

it as a spatial transformation problem, therefore it is possible to define the equations that relate both sources by knowing, a priori, some control points from the reference source and the same points in another source.

For registration we do not only need to know points in different sources, we also need to understand how these sources perceive the environment around. Devices such as cameras and LIDAR are the most common when it comes to obtaining data that allows to know, interact and react with the scenario in which they are inserted. The challenge to be able to perceive the surrounding environment as accurately as possible begins by first knowing the internal characteristics of the devices we work with. These characteristics give us the possibility to convert between types of data (for example 3D points to 2D), to get to know the external relationship between the devices and to create new data that merge different types into one much more complete.

LIDAR is a laser-based sensing technology that determines the distance objects are in the environment by emitting the laser signal and calculating the time it takes to return to the device. Due to being a sensor that captures data in a direct way, its internal characteristics only concern its field of view and maximum range. On the other hand, although the camera also produces data using light rays, the camera has lens that takes all the light rays bouncing around and uses glass to redirect them to a single point creating a image. Therefore, as the data is not obtained directly, the relationship of the image pixels with the real world points are also not directly related. This relationship can be known through the characteristics of the camera, commonly known by intrinsic parameters. The intrinsic parameters are essential to be able to use images as a coherent and accurate solution to perceive the environment and can be obtained through a process known as camera calibration.

2.1.2 Camera Calibration

Camera calibration is the process of estimating the lens and image sensor parameters, the intrinsic parameters. Intrinsic parameters define the relation between the world 3D points and the image plane 2D points and they can be used to correct lens distortion, measure the size of an object in world units, determine the location of the camera in the scene and to know how far objects are from the camera. These applications are widely used in computer vision and robotic systems for navigation, 3D reconstruction and mapping.

Intrinsic calibration

Each camera has its own lens that influences the relation between the image and the world. Intrinsic calibration gives us intrinsic parameters that allows us to make the correct relation between the points of the world and the points of the image, surpassing the effects of the lenses [12]. Intrinsic parameters do not depend on the scene or on the position of the camera in the world so, once calculated, the parameters do not change as long as features such as focus and zoom are not changed. This parameters are represented by a 3x3 matrix called *camera matrix* with the following representation:

$$K = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

where:

- f_x,f_y are the focal lengths expressed in pixel units.
- (c_x,c_y) is the point of intersection of the camera optical coordinate system with the image plane, called *principal point*. Ideally it is the center of the image.
- γ represents the skew coefficient between the image x and y axis. It is non-zero if the image axes are not perpendicular but, for most cases, they are perpendicular and skew is 0.

Camera lenses can also affect the image-world relation through the distortion effect. The distortion is a deviation from the rectilinear projection of the light rays in the image. Without distortion, the straight lines remain straight in the image. However, due to the lenses, the light is spread outside the straight line which causes the image to be distorted or blurred. Distortion and blurred effects, also called optical aberrations, depend on the type and nature of the aberration.

There are two types of distortion: *radial* and *tangential* [13]. The radial distortion happens when light rays bend more near the edges of a lens than they do at its optical center, and the smaller the lenses the greater the radial distortion. There are three different types of radial distortion: *negative radial distortion*, also known as “pincushion”; *positive radial distortion*, known as "barrel" as well; and finally the *complex distortion* or "mustache", which is a combination of the previous two.

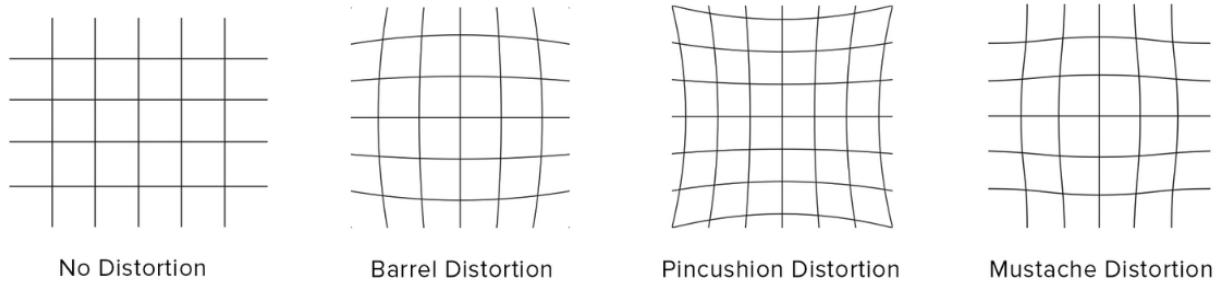


Figure 2.1: Types of radial distortion [14].

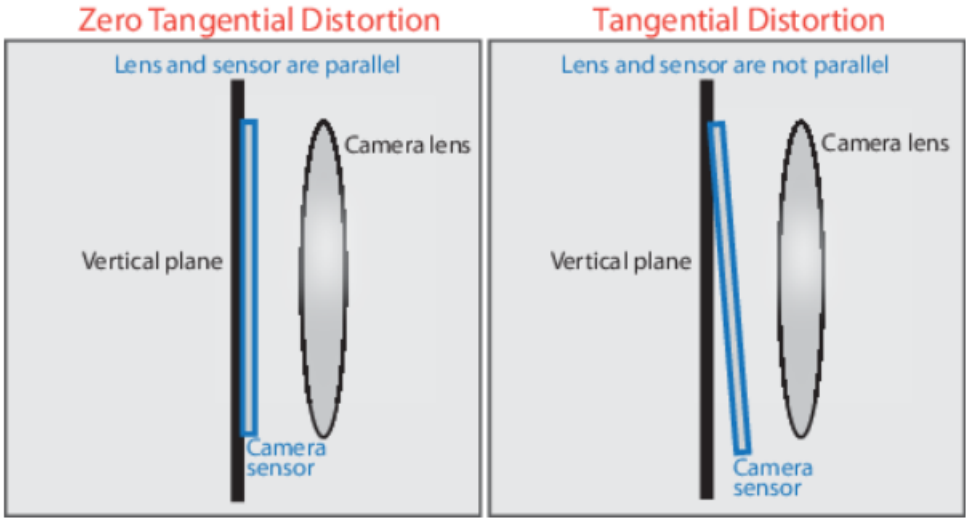


Figure 2.2: Example of tangential distortion [15].

The Figure 2.1 illustrates the effects of the these types of distortion when compared to an image with no distortion. The tangential distortion occurs when the lens is not parallel to the image plan. This is illustrated in Figure 2.2.

The effects of radial and tangential distortion are defined through distortion coefficients. Due to the effects of distortion, the pixel coordinates suffer variations in relation to the coordinates obtained through rectilinear projection. These coefficients are used to make the correct adjustment from old pixel coordinates to new corrected pixel coordinates [16]. There are six radial coefficients $(k_1, k_2, ..., k_6)$ and two tangential coefficients (p_1, p_2).

For the radial distortion, the new coordinates are calculated as:

$$x_{corrected} = x\frac{1 + k_1r^2 + k_2r^4 + k_3r^6}{1 + k_4r^2 + k_5r^4 + k_6r^6}$$
$$y_{corrected} = y\frac{1 + k_1r^2 + k_2r^4 + k_3r^6}{1 + k_4r^2 + k_5r^4 + k_6r^6} \tag{2.2}$$

As for tangential distortion, it is calculated as follows:

$$\begin{aligned} x_{corrected} &= x + [2p_1xy + p_2(r^2 + 2x^2)] \\ y_{corrected} &= y + [p_1(r^2 + 2y^2) + 2p_2xy] \end{aligned} \tag{2.3}$$

where $r^2 = x^2 + y^2$.

Extrinsic calibration

In a environment, any object can be represented by a coordinate system that defines its center and how it is oriented. The combination of the position and orientation of an object give us the object pose. Consider a scenario where two acquisition devices co-exist. The object pose information given by each device is related to its own pose in the world. These two poses will be provided without any information of the relationship between the two independent devices which perceive the object. Without knowing the relative pose between each device in our system, the perceived environment can become ambiguous, incorrect and confusing. Extrinsic calibration is the solution to relate all devices to each other or even to a common coordinate system (often called world origin or just world), making all data accurate and viable as they are related to a common point in the world [17].

Extrinsic calibration is represented by its extrinsic parameters, which are formed by two matrices: *translation* and *rotation*. The translation matrix, t, defines relative positions of each frame and is expressed as a 1x3 matrix. The rotation matrix, R, is a 3x3 matrix that rotates corresponding axes of each frame into each other. However, it is possible to write both matrices as one, forming a compound transformation called homogeneous transforms that are denoted by a 4x4 matrix as follows:

$$[R|t] = \begin{bmatrix} r11 & r12 & r12 & t_x \\ r21 & r22 & r23 & t_y \\ r31 & r32 & r33 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R_{33} & t_{13}^T \\ 0 & 1 \end{bmatrix} \tag{2.4}$$

This calibration is not only applicable to cameras but to any pair of devices from which it is possible to compute the translation and rotation that differentiate their coordinate systems. Extrinsic calibration between a pair of LIDAR is also very common and there are techniques that calculate the spatial relationship through the point clouds that both produce. We also take a look at them later in this document.

Projection

Data can be differentiated in spatial dimensions between 2D and 3D. There are many cases in which we have data with different dimensional spaces and we want to relate them, such as wanting to relate data from a range finder (3D) with the a camera image (2D). However, relating data that does not have the same dimensional space can be complex and to overcome these challenge we turn to projection.

Projection is the process of converting 3D points to 2D. A scene view by the camera is formed by the projection of the 3D world points into the image plane. This process is also called perspective or even projection perspective transformation [18].

The camera matrix can be extended with a further column becoming a 3x4 matrix which is known as projection matrix:

$$P = \begin{bmatrix} f'_x & \gamma & c'_x & T_x \\ 0 & f'_y & c'_y & T_y \\ 0 & 0 & 1 & T_z \end{bmatrix} \tag{2.5}$$

Despite being an extension of the camera matrix the parameters can be slightly different. It is known that the real optical center is somewhere inside the camera and that the 3D coordinate system, which we use in the calculations, refers to that optical center. However, in reality, they may not be quite the same. Due to this, the optical center may differ in a roto-translation transformation from the coordinate system, and the parameters difference is usually due to this transformation. The fourth column differs for monocular and stereo cameras. For a stereo pair, this column relates the position of the second camera optical center with the first camera's frame. As both cameras are in the same stereo image plane we assume $T_z = 0$ and we also assume that the first camera always has the initial conditions $T_x = T_y = 0$. For the second camera of a horizontal stereo pair, generally the one on the right, $T_y = 0$ and $T_x = -f'_x \cdot d$, where d is the distance between both cameras[1]. For monocular cameras, we define it as if it is just the reference camera in a stereo pair, i.e., $T_x = T_y = T_z = 0$.

In this way, we can work with homogeneous points which is the most common notation in robotics and rigid body transforms. We use $[x, y, w]$ for 2D points and $[x, y, z, w]$ for 3D points where, initially and by convention, $w = 1$. Thus, if any point is scaled by a factor w, all parameters must be multiplied or divided so that w remains 1.

[1] `https://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/CameraInfo.html`

With both intrinsic and extrinsic parameters we are able to project world points into a image plane [19]. We will consider the situation where we use a monocular camera and so it is sufficient to use the K matrix. The projection is done by the following equation:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f'_x & \gamma & c'_x \\ 0 & f'_y & c'_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r11 & r12 & r12 & t_x \\ r21 & r22 & r23 & t_y \\ r31 & r32 & r33 & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.6}$$

which can be written using the above-mentioned notations:

$$s \begin{bmatrix} u \\ v \\ w \end{bmatrix} = K[R|t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.7}$$

where:

- s is an arbitrary scale factor.
- (X,Y,Z) are the coordinates of a 3D point in the world coordinate system.
- (u,v) are the coordinates of the point that was projected to the 2D image plane, in pixels.
- K is the camera matrix.
- $[R|t]$ is the matrix of extrinsic parameters which translates coordinates of a point (X,Y,Z) to a fixed coordinate system in respect to the camera, typically the optical coordinate system.

This expression is applicable to cases where the distortion effect is negligible. For cases with distortion, the points undergo a small change in their coordinates.

First, we will look at the projection in more detail. The calculation of one 3D world point for the desired frame is done with the extrinsic parameter matrix, as shown in the equation 2.8. Then, as we use homogeneous coordinates, we need to divide all coordinates by the factor w, as shown in equations 2.9. Finally, the image points (u,v) are determined

using the camera's intrinsic parameters, following equations 2.10.

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} + T \tag{2.8}$$

$$\begin{aligned} x' &= x/w \\ y' &= y/w \\ z' &= z/w \end{aligned} \tag{2.9}$$

$$\begin{aligned} u &= f_x * x' + c_x \\ v &= f_y * y' + c_y \end{aligned} \tag{2.10}$$

However, *with distortion*, the points of the world seen by the camera are not so directly related into the image plane. As discussed in this chapter the pixel coordinates are adjusted, so the equation 2.9 is extended to:

$$\begin{aligned} x'' &= x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y'' &= y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + [p_1(r^2 + 2y^2) + 2p_2 xy] \end{aligned} \tag{2.11}$$

and the correct pixel coordinates are:

$$\begin{aligned} u &= f_x * x'' + c_x \\ v &= f_y * y'' + c_y \end{aligned} \tag{2.12}$$

2.1.3 Multi-sensor registration

In this subsection, we present different types and methods for multi-sensor registration, and some of their advantages and challenges. Since registration is based on the relationship between two different sensors and data sets, there are many methods that aim to make that relationship as accurate as possible. The methods are practically applicable to all types of image and point clouds. Although these relationships are relatively limited to the same type of data (image-image or point cloud-point cloud), it is possible to transform data into other

types. Transforming point clouds into images by projecting 3D points onto an image plane or transforming an image into a point cloud are a very common example of this. Therefore, it is also possible to register a camera and a LIDAR through two point clouds or through two images, despite initially not having the same type of data.

We can separate the methods into two groups: *Manual* and *Automatic* [20] [21]. Manual registration is a method where the processes of identification and matching image points are done with human help. It is common to use this type of registration to match point clouds because there are several software that allow us to easily obtain accurate point information from those point clouds. With these points the problem becomes a simple spatial relationship between corresponding points that can be obtained through mathematical optimization methods. Automatic registration are methods in which the detection and matching of points is done without human help. For these methods it is essential to detect objects that are easily related between different images, the so-called *features*.

We can also distinguish the methods into two major classes: *Image* [3] and *Point cloud* registration [22] [23].

Image Registration

Image registration is an automatic or manual procedure which tries to find corresponding points between two or more images and align them spatially to minimize a desired error.

Image registration can be divided into two types: *Image to Image* and *Image to Map*. In Image to Image, two images are aligned so that, when fused or integrated, the pixels correspond to the same object. In Image to Map the images are "stitched" so that they can build a map with their information, maintaining their spatial relations. In other words, we do not intend to find the direct relationship between the images to fuse corresponding pixels, but we do intend to calculate the spatial transforms which align a set of images to a common observational frame, usually the base of the robotic system [3].

Due to the diversity of images to be registered and the different types of resources, it is not possible to define a universal method. Even so, most image registration methods have four fundamental steps [24]:

- Feature detection: distinctive features (closed-boundary regions, edges, contours, line intersections, corners, etc.) are manually or automatically detected. For further processing, these features can be represented by their point representatives (like centers of gravity and distinctive points) which are called control points.

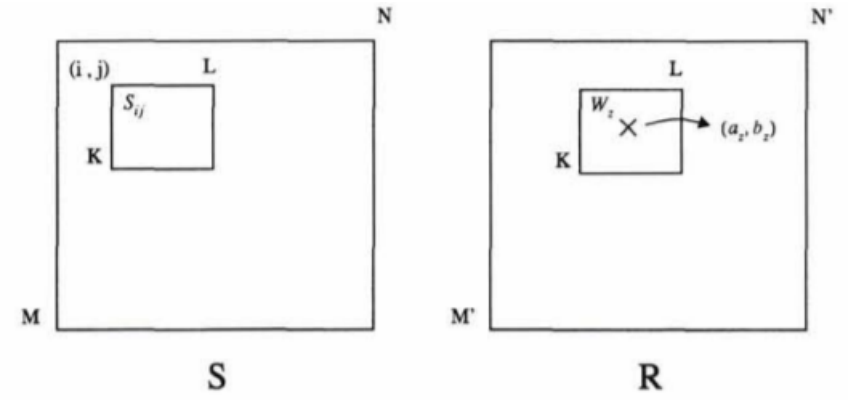


Figure 2.3: Area-based registration and demonstration of windows and control points [25].

- Feature matching: the correspondence between the features detected in the sensed image and those detected in the reference image is established.
- Transform model estimation: The type and parameters of the functions that aligns the sensed image with the reference image are estimated. This functions are known as the mapping functions.
- Image resampling and transformation: the mapping functions obtained in the previous step are used to transform the sensed image.

With the evolution of automation in registration, the feature detection step gradually stopped being made by users or experts and two main fields of feature understanding began to stand out: *Area* and *Feature based* registration.

Area based methods are used for cases where details that can be easily detected are not predominant, so they take advantage of features like colors or gray levels. One of the challenges of this methods is that they cannot be applied directly to all cases due to the differences of gray levels that can vary from sensor to sensor, causing uncertainty and serious errors. In area based registration the identification of control points is made by comparing small windows of points, with the same size, on the reference image and on the sensed images. Thus, each window W_z of the sensed image is compared to a sub-image (Si, j) of the reference image and, when the window that best matches both is found, the information of the window's centers and the sub-image's center is stored as control points. With these control points it is possible to calculate the parameters of the transformation that relates both images, and then allows to determine the spatial transformation between the two cameras that acquired the images [25]. The area-based registration technique is illustrated by the figure 2.3.

Feature based methods are applied when there are features sufficiently strong and easy to detect. These methods usually make use of feature extraction algorithms that prioritize more local structural and image intensity information. Wide variations in sets of points are good carriers of useful information, where are included edges, corners, blobs, contours, surfaces and points of intersection [26]. Feature detection and feature matching are two fundamental processes for both method accuracy and autonomy. Many methods that mainly aim to detect edges, corners and blobs have been developed. The most common feature detectors are Canny, Sobel, Harris and Laplacian of Gaussian. Approaches such as FAST (Features from Accelerated Segment Test) [27], SIFT (Scale-Invariant Feature Transform) [28], SURF (Speeded-up Robust Features) [29] [30], BRIEF (Binary Robust Independent Elementary Features) [31] and ORB (Oriented FAST and Rotated BRIEF) [32] have gained a lot of attention due to their ability to use these feature detection techniques with great precision and with reduced processing time.

Table 2.1: Feature detection methods and types of features that each is able to detect [33] [34].

Feature detectors	Edge	Corner	Blob
Canny	Yes	No	No
Sobel	Yes	No	No
Harris and Stephens	Yes	Yes	No
Laplacian of Gaussian	No	Yes	Yes
Difference of Gaussians	No	Yes	Yes
FAST	No	Yes	Yes
SUSAN	Yes	Yes	No
Level Curve Curvature	No	Yes	No
Determinant of the Hessian	No	Yes	Yes
MSER	No	No	Yes
PCBR	No	No	Yes

Point cloud Registration

Point cloud registration is the process of finding a spatial transformation that aligns two point clouds. Point clouds can be 2D or 3D and are widely used in robotic systems for mapping. There are several devices that can produce point clouds such as LIDAR and

RGB-D cameras, therefore the point cloud correspondence is a commonly used method to estimate the spatial transformation of both data and devices. They are also very useful for registration with cameras that can not produce point clouds (like monocular cameras) as it is possible to turn a 3D point cloud into a 2D one and relate it to 2D images. With this relationship, we are able to combine color information of the image with the depth data of a ranger finder, making it possible to make a 3D map with different types of useful data [22].

Matching point clouds can be done through a rigid or a non-rigid transformation. A rigid transformation is defined as a transformation that does not change the distance between any two points, so typically it consists of translation and rotation. On the other hand, non-rigid transformations do not guarantee that the distance between each two points is maintained since scale and shear transformations are applied. In most cases in robotic systems, point clouds can be related through a simple rigid transformation since, in the environment in which they are located, the objects do not suffer deformations thus maintaining the distance between their points. Therefore, we will only refer methods that estimate this rigid transformation.

The methods for matching points are mostly based on the iteration of finding the closest pairs of points until all points match correctly. A well-known example that describes this idea is the ICP (Iterative Corresponding Point) [35] [36]. In general, ICP matches the closest point in the reference point cloud to the sensed one, estimates the rotation and translation transformations by applying a root mean square point optimization, transforms the points with this transformations and iterates these steps again until all the points are match correctly. In the figure 2.4 are represented two point clouds that differ by a transformation and, through ICP, it was possible to match the points of both and estimate the rotation and translation that relates them.

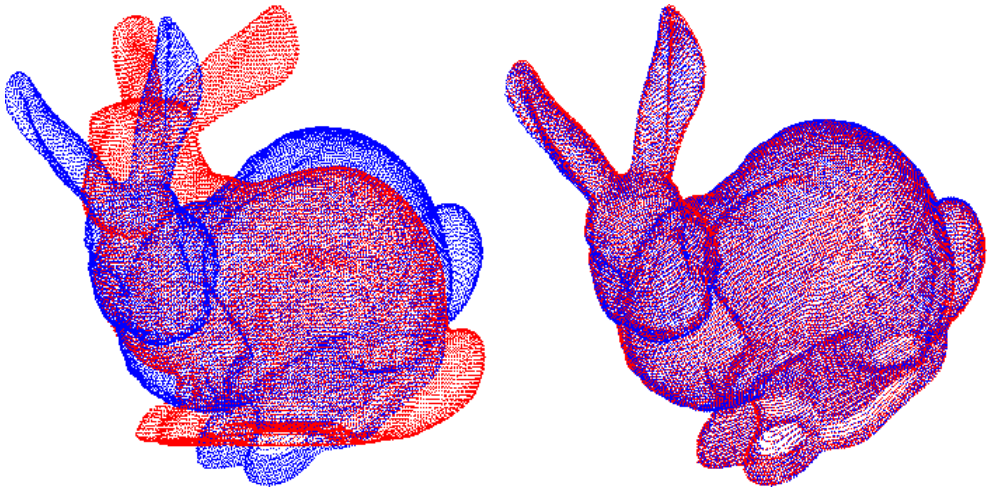


Figure 2.4: Example of matching point clouds with ICP [37].

2.2 Mapping

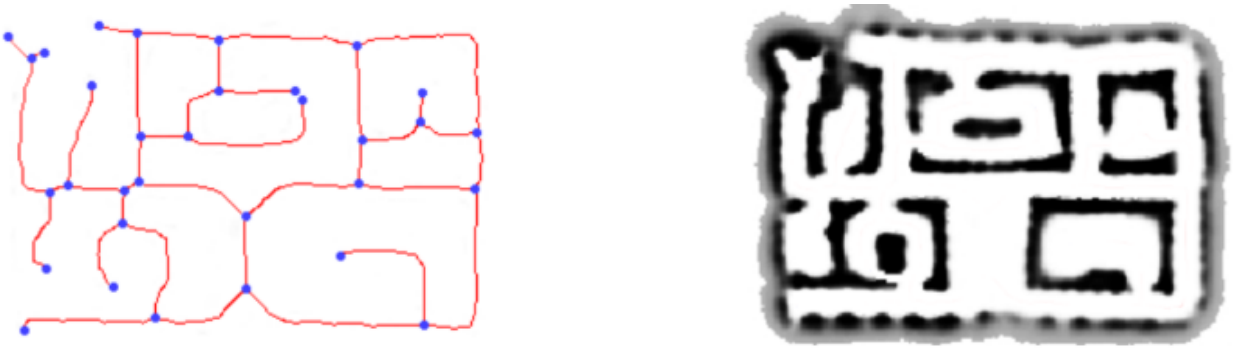
2.2.1 What is Mapping?

Mapping is the process of creating a map with the intention to act on it. A map is a representation of an area showing physical features like buildings, roads or any other element of the environment. There are two main categories: *metric* (or grid based) and *topological* maps. The Figure 2.5 illustrates the difference between them.

Metric maps represent the environment through evenly-spaced tables where each grid cell represents a location in space. Occupancy grids are one of the most popular type of maps because they are easily built for large scale locations and the grid geometry directly matches the environment geometry, so the position and orientation of the robot can be easily determined. The downside of grid-based maps is that they take up huge space and have a high computational cost because they need to capture a huge amount of points in the world under analysis [38].

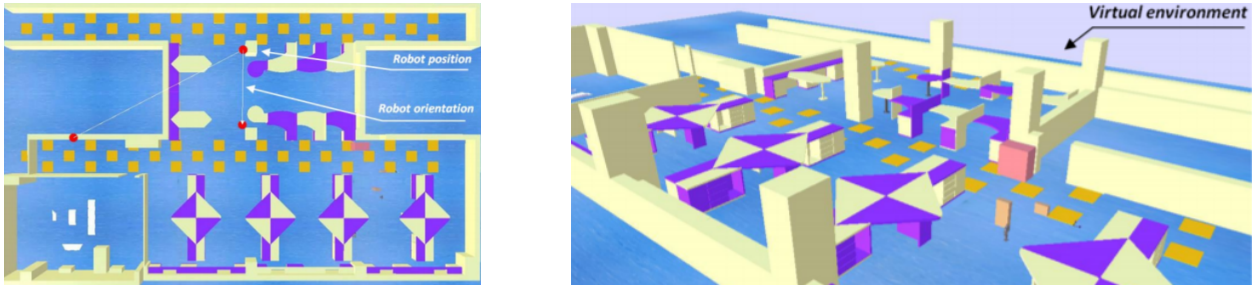
On the other hand, topological maps represent the map by a graph, based on landmarks and their connectivity. In this method the robot's position is determined relatively to the model through the landmarks. Each node corresponds to a specific location and the existence of an edge connecting them indicates the possibility of navigating. However, topological maps have difficulty distinguishing very identical locations, so if the robot goes through two similar locations it cannot quickly determine if it is the same or a different location than before [39].

Maps can also differ in 2D and 3D. For 2D maps, objects are represented in two dimensions, length and width. 3D maps represent length, width and depth, allowing the perception



(a) Topological (b) Metric

Figure 2.5: A comparison between metric and topological map, images from [40] with the author's permission.



(a) 2D (b) 3D

Figure 2.6: Overview of 2D and 3D maps [41].

of relief. These differences can be seen in the Figure 2.6.

The process of creating maps cannot be separated from the localization process, and an incorrect estimation of the location causes errors that are incorporated into the map. This is a well-known problem that we call Simultaneous Localization and Mapping (SLAM) [42] [43]. SLAM has some complexity because the localization and mapping processes may be under unfavorable conditions, such as errors, uncertainty, ambiguity and noise, which do not make it easy to develop a coherent solution for both tasks.

2.2.2 Classical Methods and Solutions for Mapping

Sensor and environment uncertainty have always given a challenge to the development of robotic navigation and mapping methods. The main solutions to overcome the sensors and environment ambiguity are probabilistic techniques due to their ability to handle modeled uncertainties. In this section, we overview the most successful probabilistic algorithms [44], including *Kalman Filter*, *Expectation Maximisation* and *Particle Filter algorithms*, *LOAM*

and *RTAB-Map*.

Filtering techniques

In filtering techniques data is processed, incorporated into the filter's system and then discarded. This techniques are usually additive, in the sense that they add data to the map as the robot explores its surroundings.

Bayes Filters, in robotics, are the basis for most classical solutions to SLAM problems and it allows robots to continuously update the most likely position within a coordinate system based on the most recently acquired sensor data. The Bayes filter represents t as time, s_t as the robot's pose, m_t the map known over time and $x_t = \{s_t, m_p\}$ as the complete state, which is normally the robot's position in the map. Also, sensor data (images, range measurements, etc) are denoted as z, and control signals as u, which are the signals that determine the robot's motion. With that, it aims to continuously estimate a probability distribution that calculate the probability of the robot being in a certain position x at a time t knowing the sensor's data history [45]:

$$p(s_t, m_t | z^t, u^t) = p(x_t | z^t, u^t) \tag{2.13}$$

Then by integrating the set of observations, that arrives over time, we can get a posterior probability over the maps and robot poses [46]:

$$p(x_t | z^t, u^t) = \eta p(z_t | x_t) \int p(x_t | u_t, x_{t-1}) p(x_{t-1} | z^{t-1}, u^{t-1}) dx_{t-1} \tag{2.14}$$

where η is a normalizing constant.

The specific case where the Bayes filter follows a Gaussian distribution is called *Kalman Filter* (KF). Most of the time we assume that the map is static and that the maps and poses depend on the previous ones. However, this relationship may not be linearly dependent and to solve this problem we apply a Taylor series to the nonlinear function. This approach with such modifications is called the *Extended Kalman Filter* (EKF). The downside is that, when EKF is applied to the SLAM problem, it restricts many applications because it requires data to be modeled as geometric shapes [47] [48].

Expectation Maximisation Methods complement the Kalman Filter approaches as they manage to solve the correspondence problem, in complex and ambiguous environments, by using expectation and maximisation (EM) techniques. The expectation step generates multiple hypotheses about the robot's path in a single map and the maximisation step uses

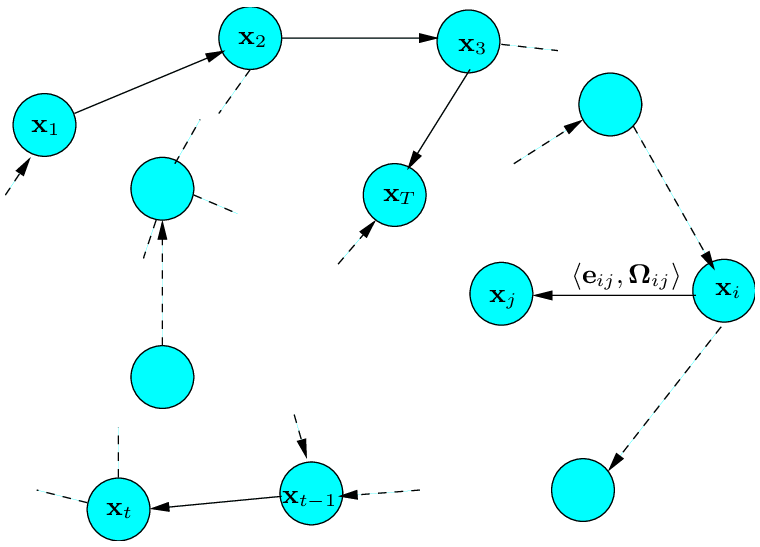


Figure 2.7: A graphical representation of the constraint network used by graph-based SLAM [50].

the correspondences between the sensory data and features in the environment to create the final features that will result in the most likely map. The final features will also help to maintain a correct estimation of the robot's path through further iterations and the incorrect features gradually disappear. As it is a more selective method in the data to be used, the general notion of uncertainty, used in Kalman filters, is not maintained. This means that the algorithm can get stuck during iterative map correction processes due to a lack of data related to previous iterations. The algorithm is also computationally expensive to guarantee an accurate map, which can be a problem for systems that depend on real-time processing.

Particle filters have also been introduced as a solution to SLAM problems because they can deal with non-Gaussian distributions commonly seen in noise, motion and sensory features. In this approach, each particle of the filter carries an individual possible map solution and it can generally be assumed that the denser parts of each group, i.e., groups of solutions that are very close to each other in a certain space, indicate a correct estimation. Since the best results with this method are with the largest number of particles and as each particle has a considerable computational cost, the challenge is to determine the perfect balance between low computational cost and high precision [49].

Graph-Based Methods

As the name implies, this method makes it possible to reformulate mapping and localization problems into a graph-related problem, the so well-known Graph-based SLAM. The name is often misleading as it makes it appear that it is related to the way the map is constructed

but is actually related to the way the data is represented and optimized. As shown in the Figure 2.7, every node in the graph corresponds to a robot pose and nearby poses are connected. $\Omega_{i,j}$ represents the information matrix of the measurements and $e_{i,j}$ is the error which depends on the difference between the expected observation, usually the relative transformation between the two nodes, and the real observation. Graph-based methods works with all the data gathered while maintaining a sense of general uncertainty in a graph form, that represent places in the environment, and how these places are connected, keeping a complete history of past poses [50]. This an important characteristic of these techniques as they make good use of past poses to define local frames of reference, which makes it a good tool for rendering range scans into a map. When these methods were initially developed, the map generation was often done only after all data was gathered because the computational demand is high. However, with the growth of computing power, it became possible to do this technique in real time.

In graph-based approaches, a node represents a particular robot pose while edges encode either odometry or loop-closure constraints. For each edge, they represent a transformation matrix (rotation and translation) as well as a covariance matrix, with the uncertainties based on these transformations. Then, an optimization algorithm is applied to reduce nonlinear loop closure and odometry constraints as much as possible in order to determine the most likely map [51]. *Loop closure* consists in detecting when the robot has returned to a known past location and thus determine the drift between the real and computed position. This allows us to drastically reduce misleading errors on the estimated trajectory. *Odometry* lets us estimate the robot own position and its variations over time using information provided by motion sensors. This method is sensitive to errors due to the integration of speed measurements over time and to the environments issues because, as it is normally used on wheeled robots, they have a high probability of wheel slippage. Optimizations that use loop closure and odometry simultaneously are called *pose-graph optimization*.

RTAB-Map (Real-Time Appearance-Based Mapping) is a RGB-D Graph SLAM approach based on a global Bayesian loop closure detector [52] [53]. It is a graph based SLAM technique because it determines how likely a new image comes from a previous or a new location via a loop closure process [54] [55]. When a good assumption is accepted, these new images are added and optimized by a graph optimizer. The approach in [56] consists in four steps: extraction of visual features from the images and then match those features with another ones from previous images. After that, they get a set of 3D point correspondences between any

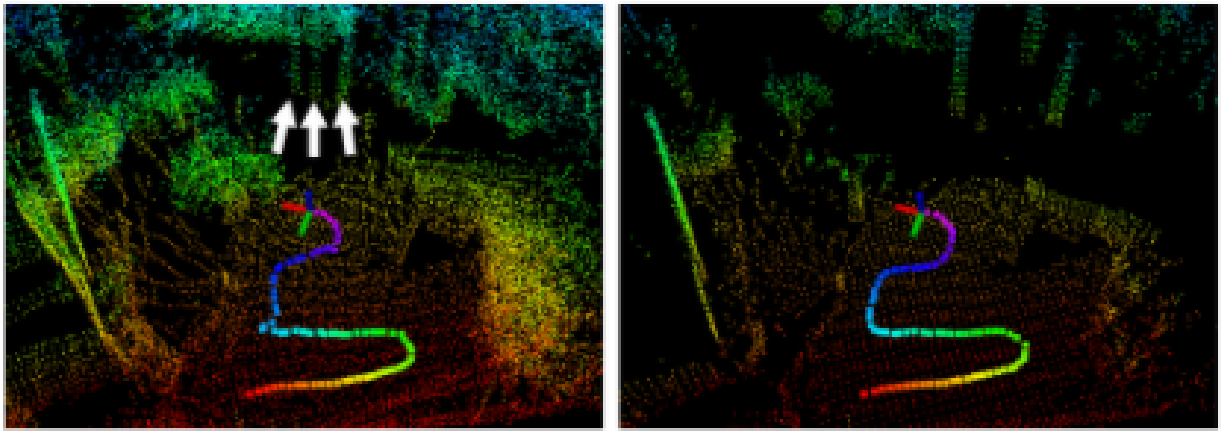
[2] `http://introlab.github.io/rtabmap/`

Figure 2.8: An example of a RTAB-Map representation[2].

two frames and, finally, they can estimate the relative transformation that relate the frames. One of the biggest strengths of these technique is its compatibility with a wide variety of devices, namely a handheld Kinect, a stereo camera or a 3D LIDAR for 6DoF mapping, or a laser rangefinder for 3DoF mapping. The Figure 2.8 illustrates a map example with the RTAB-Map approach.

LOAM stands for LIDAR Odometry and Mapping in Real-time. Essentially, this means that this technique uses range measurements captured by a LIDAR to locate itself in an environment and build a map of it. Although it gives us results with little computational complexity and with extremely accurate measurements, these measurements are received at different times and may cause errors in motion estimation and in the registration of the point cloud. Laser scanning enables a wide range of applications that can be obtained with great accuracy without the need for perfect conditions as it is not influenced by environmental conditions. The weaknesses of laser scanning are the need of many scans in different locations to get a dense result, it takes some time to complete a full scan, it needs a common coordinate system and it may require additional hardware like planar and sphere targets, making the whole system more expensive [57].

In [59], the main idea focuses on dividing SLAM complexity into two algorithms: one does high-frequency visual odometry to estimate motion and the other uses LIDAR odometry at a lower frequency to refine motion estimation and remove distortion in the point cloud. Thus, they combine a monocular camera with a 3D LIDAR and named this method as *V-LOAM* (Visual LIDAR Odometry and Mapping). Despite V-LOAM presenting interesting results for environments that do not provide data overload, as it calculates the transformation be-



(a) LOAM (b) LeGO-LOAM

Figure 2.9: Maps obtained from LOAM and LeGO-LOAM techniques. In the LOAM map, three arrows represent the same tree which shows uncertainty and ambiguity. The same does not happen with LeGO-LOAM where only one tree is represented and the path is also more accurate [58].

tween two sets of point clouds through edges and planar points, for forest environments these results may be substandard due to the large amounts of vegetation and trees that this environments usually have, therefore the data to be processed is high. Improved techniques like *A-LOAM* (Advanced LOAM)[3] and *LeGO-LOAM* (Lightweight and Ground-Optimized Handling Odometry and Mapping) [60] aim to make a cleaner and simpler LOAM method, both using identical processes to LOAM but without redundancies and complicated mathematical processes. LeGO-LOAM focuses on processing data from overloaded environments with the presence of a ground plane in its segmentation and optimization steps. The work in [58] shows the results of LeGO-LOAM and makes a strong comparison between the classic LOAM and the improved method, as it can be seen in figure 2.9.

[3] `https://github.com/HKUST-Aerial-Robotics/A-LOAM`

Table 2.2: An overview of above mentioned mapping methods.

	Based on	Computational cost	Map accuracy	Notes
Particle filters [49]	Particle density	Depends on the number of particles	Depends on the number of particles	Very good results imply high computational cost Can deal with non-Gaussian distributions which are the most frequent cases due to noise
EKF-SLAM [47] [48]	Gaussian distributions of Bayes filters	Depends on the number of particles	Depends on the number of particles	Coherent results when the heading uncertainty stays "small", i.e. it can not have disproportionately large jumps in the vehicle pose update Requires data to be modeled as geometric shapes
Graph SLAM [50]	Loop closure and Odometry	Medium	Medium	Limited computational resources may have an impact on mapping results
LOAM [57]	Range measurements given by a 6DOF LIDAR	Low	High	Very good for accurate localization within the centimeter-level, may not be feasible due to the lack of simple geometric shapes on the environment
A-LOAM (footnote 3)	Ceres Solver and Eigen library for code structure simplification	Low	High	At high speeds this method starts to fail at ICP for point cloud registration
LeGO-LOAM [60] [58]	Process data from non-structured environments like forests	Low	Very High	Maintains good results in high speed processing Requires a ground plane in the segmentation and optimization steps

2.3 Software and Harware

2.3.1 ROS: Robot Operating System

In science and technology, the validation of theoretical methods must be conducted through detailed tests and analysis. The robotic community supports this aspect, since the approaches proposed are usually compiled in code and complemented by documentation of the functionality. This allows the developed approach to be retested and improved by other users reducing the time it takes to prepare a solution.

Robot Operating System (ROS) is a software framework that provides just that. ROS is mostly open source and developed work is commonly available for other users to test, reuse and possibly improve, encouraging collaborative robotics software development. It is a collection of tools, libraries, and conventions that aim to simplify the development of complex and robust code for robots.

As described in [61], ROS is based on the exchange of messages between *ROS nodes*, which are programs that take advantage of ROS functionalities to communicate with each other. ROS nodes run independently but simultaneously, and do not need to be operating on the same device. ROS nodes can have different implementations: data access and management, mathematical functions, or any functionality possible through the languages supported by ROS (such as Python and C++). This is a very important feature because, in robotic systems, it is very common to have to process, manage and communicate data coming from several devices at the same time. ROS also allows us to abstract away the hardware complexity of many robots , therefore code is written in such a way that it is independent of the underlying robot, sensor or any hardware used[4].

ROS software is a compilation of messages, nodes, libraries, configuration files, and others. All these components together are distributed in packages that make sharing and obtaining software much easier. Usually, these packages are obtained through repositories and compiled with the CMAKE toolset, which produces executable code.

2.3.2 Ranger

The Ranger (see figure 2.10) is a 4000kg autonomous robot, based on the Bobcat T190, which is the main actor of the SEMFIRE project, being responsible for landscape maintenance in

[4] `https://www.ros.org/about-ros/`

Figure 2.10: Main sensory hub.

the forest environment.

This platform has these characteristics:

- It is able to carry all the tools (a mechanical mulcher, sensor kit, computational array and others) necessary to complete the mission;
- It is completely fly-by-wire, meaning that it encompasses electronic control mechanisms to develop remote and autonomous control routines;
- It is a well-known, well-supported machine with readily-available maintenance experts.

The ranger supports a sensor kit that has:

- A 3D 16 channels LeiShen C16 LIDAR[5];
- Five Intel RealSense D435 RGB-D Cameras[6] with five AAEON UP Board Atom for sensor data acquisition;
- One FLIR AX8 thermal camera[7];
- One Teledyne Dalsa Genie Nano C2420 multispectral camera[8];

[5] http://en.leishen-LIDAR.com/product/leida/MX/15d44ea1-94f5-4b89-86eb-f5a781b04078.html

[6] https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d415.html

[7] https://www.flir.com/products/ax8-automation/

[8] https://www.edmundoptics.eu/p/c2420-23-color-dalsa-genie-nano-poe-camera/4059/

- GPS and RTK devices[9];
- Inertial Measurement Unit[10];
- Mini-ITX computer equipped with a Geforce RTX 2060, an Intel Core i7–8700 CPU and 16GB of DDR4 RAM

The Intel Real Sense cameras consists in a pair of depth sensors, RGB sensor and a infrared projector. We have one in front to observe obstacles and objects in front of the robot, and it is useful for safety.

Perception is complemented by a FLIR AX8 thermal camera and a Dalsa Genie Nano Multispectral camera. The thermal camera combines thermal imaging with a visual camera so that it is possible to detect temperature differences. It will be mainly used to detect human personnel directly in front of the robot, i.e. in potential danger from collisions with the machine or the mulcher attachment. The multispectral camera allows us to see an object or any element of the environment at different wavelengths such as visible, infrared and ultraviolet light. With this technology, we can distinguish elements that may be identical in the visible light band but different in other bands. This way the robot can autonomously analyze the scene and detect plant material in various stages of decay. This can be very useful to detect flammable material for clearing.

[9] `https://emlid.com/reachrs/`

[10] `https://www.pololu.com/product/2740`

3 Proposed Approaches for Registration and Calibration

When sensors are assembled in a robotic system, their positions and orientations are chosen in a strategic way to make the most of all the information obtained, both individually and collectively. Therefore, initially spatial relationships between devices are of little importance, but they become increasingly essential the more complex and complete the desired output is.

It is common to apply manual registration as a quick and simple solution to understand the geometric relationship between devices. However, it is perfectly understood that manual measurements, i.e., with the use of physical tools and with human action, are subject to errors and uncertainties, not only due to the human eye and skill but also to the conditions to which the measurements are submitted. It is possible for two identical devices to be supported by a straight base, at the same height, both perfectly parallel to each other and yet the physical measurement having errors. This can happen due to several factors, one of which is the fact that we do not know perfectly the optical center of each camera or even the origin of the LIDAR coordinate system. We can see that this type of calibration and registration is acceptable for quick and not so accurate tests, but it is extremely important to develop very precise techniques that are able to minimize the difference between computed and real values, through mathematical and geometric methods. For that we can also take advantage of the devices characteristics, the surrounding environment and well-known third-party objects.

OpenCV (Open Source Computer Vision Library)[11] is an open source computer vision and machine learning software library . The library has more than 2500 optimized algorithms that are used extensively in software that uses cameras, images and point clouds. Among many other features, OpenCV allows camera calibration, object detection, face recognition,

[11] `https://opencv.org/`

point clouds production, recognition of scenes and markers related to augmented reality, find identical images in a database and track camera movements. We make use of the OpenCV tool and several of its features in this work within the software developed.

Monocular and Stereo Camera Calibration

Camera calibration is a needed step in 3D computer vision in order to extract metric information from 2D images. There is a lot of work developed in this field [62] [63] [64], starting with photogrammetry and more recently with computer vision. The existing techniques can be divided into two main categories: calibration by *photogrammetry* and *self-calibration*.

- Photogrammetric or object-based calibration is performed by making use of an object whose geometry is well known, i.e., whose dimensions and distances are well-known and unchangeable. Usually an object whose translation with respect to our sensor is accurately known is used, or an object from which we can obtain important information such as the geometric shape, lines, contours, corners and others. The calibration obtained through these objects can be done with great efficiency but they require an elaborate setup that allows us to know the surrounding environment and relate it accurately between spaces and entities in the scenario [65] [66].

- Self-calibration is a technique that does not use another object for calibration. In fact, these techniques use the movement of single camera in the scene to obtain different images and then find correspondences between these images. As these images come from the same camera, the intrinsic parameters are maintained and the information taken by the correspondences allows to recover both the intrinsic and extrinsic parameters. This approach is a more flexible method but still has a lot of room for improvement and so the results are not always reliable [67] [68].

A widely used technique in calibration processes is to combine photogrammetry and self-calibration, the former because it uses a 3D model and the latter because it uses motion from either the camera or the object. An appropriate example of that is the approach presented in [63]. Assuming that the object plane has the origin of the world coordinate system, Z is

always equal to zero:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[r_1 \;\; r_2 \;\; r_3 \;\; t]\begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = K[r_1 \;\; r_2 \;\; t]\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.1}$$

where r_i represent the ith column of the rotation matrix in $[R|t]$. Then, the homography matrix H is the matrix that relates the point of the object M to the point of the image m:

$$sm = HM \qquad with \;\; H = K[r_1 \;\; r_2 \;\; t] \tag{3.2}$$

Finally, through geometric interpretation, it is possible to calculate the intrinsic parameters of the camera:

$$\begin{aligned} c_x &= \gamma v_0/\alpha - B_{13}\alpha^2/\gamma \\ c_y &= (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \\ f_x &= \sqrt{\gamma/B_{11}} \\ f_y &= \sqrt{\gamma B_{11}/(B_{11}B_{22} - B_{12}^2)} \\ \gamma &= -B_{12}\alpha - B_{13}\alpha/\gamma \\ \lambda &= B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23}]/B_{11} \end{aligned} \tag{3.3}$$

where $B = K^{-T}K^{-1}$. Once the intrinsic parameters matrix K is known, we can compute the extrinsic parameters for each image:

$$r_1 = \lambda K^{-1}h_1, \quad r_2 = \lambda K^{-1}h_2, \quad r_3 = r_1 \times r_2, \quad t = \lambda K^{-1}h_3 \tag{3.4}$$

where $\lambda = 1/\left\|K^{-1}h_1\right\| = 1/\left\|K^{-1}h_2\right\|$ and h_i the homography matrix ith column. The rotation matrix $R = [r_1, r_2, r_3]$ computed in this way sometimes does not satisfy the properties of a rotation matrix. The best way to obtain the R matrix is through the singular value decomposition that is described later in this document. We can then recalculate the parameters assuming that they are corrupted by an equally distributed noise. For this, a maximum-likelihood estimation is used, which is based on minimizing the following function:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\left\|m_{ij} - \tilde{m}(K, R_i, t_i, M_j)\right\|^2 \tag{3.5}$$

where $\tilde{m}(K, R_i, t_i, M_j)$ is the projection of the point M_j in image i.

In the end, we can define the calibration procedure mentioned by the following steps:

- Print a pattern and attach it to a planar surface.
- Moving the model object or the camera while taking multiple images under different orientations and positions.
- Detect feature points in the image, usually corners of the pattern.
- Estimate the intrinsic parameters and extrinsic parameters by using the solution mentioned above.
- Recalculate all parameters by using Maximum-Likelihood Estimation.

The ROS Perception is an open-source repository that maintains software developed by contributors related to perception[12]. This software can be reused by the community in order to create their own software while overcoming the complexity of some well-known mathematical, geometric and graphic processes. The calibration software[13] for monocular and stereo cameras in this repository allows a complete and optimized calibration using only a chessboard and the OpenCV library. The calibration is done following the steps described above and by also using pattern and feature detection to extract the image points needed for the intrinsic parameters calculation. To calibrate our robotic system, cameras we used this calibration software, with which we obtained the dimensions of the camera images (width, height), the matrix of intrinsic parameters (K), the projection matrix (P) and the distortion coefficients (k_i and p_i).

3.1 LIDAR-Camera Calibration and Registration

The combination of cameras and LIDAR in robotic systems has increased tremendously in recent years. This allows a robotic system to perceive the surrounding scenario and thus be able to perform tasks autonomously, namely driving and mapping. The latest prototypes of cars with autonomous driving capability, using both of these devices, have brought the community's attention to their importance and usefulness. Due to this high interest, many methods and techniques have been developed to provide an accurate geometric relationship between a camera and a 3D LIDAR.

[12] `https://github.com/ros-perception`

[13] `http://wiki.ros.org/camera_calibration`

However, there are many aspects that must be taken into account when working with these two devices:

- *Intersection of field of view*: as already mentioned in previous chapters, registration is based on being able to find a relationship between well-defined features that are present in both camera and LIDAR frames. This factor will depend not only on the pose of the devices but also on the field of view of each one, as this will define the possibility of having common features.

- *Monocular or Stereo camera*: it is common for stereo cameras to produce a point cloud, computing depth values by stereo vision, however this is not possible with monocular cameras. In our context, to register a 3D LIDAR with different stereo and monocular cameras we will make use of the correspondence between 3D points and apply projection and reprojection between 3D and 2D.

ICP is an iterative method that seeks to minimize the Euclesian distance between points in two point clouds. In each iteration, the rotation and translation that best reduces the distances between each pair of points is calculated. With this, after many iterations, the result converges to the correspondence between each pair of points, and with all the rotations and translations of each iteration we can determine the final transformation. Finding the correct matches in this way can often result in unwanted results for cases with very similar features or without any reference point. However, for cases where we know some correspondences between point clouds with different frames, the Kabsch algorithm [69] is a closed form solution for it.

The software developed in this dissertation for LIDAR-Camera calibration complements Ankit Dhall *et al.* method [19]. This method is made available via a ROS package[14] accompanied with a detailed explanation on how to use it. The basic idea is to be able to make a correspondence between the LIDAR and camera 3D points, and find $[R|t]$ between both coordinate systems using the Kabsch algorithm.

Before we go any further, we discuss the most important steps while setting up and running the registration software. Boards are a very important agent for the execution of this technique because they are an easily distinguishable rigid and planar object. Also, as they are rectangular or quadrangular, they allow us to promptly define edges and corners as

[14] `https://github.com/ankitdhall/lidar_camera_calibration?fbclid=IwAR1brdsTuvcRaIm6uS-IZDrRm1ip68qT1c1cP7-RuTSvRTaTbeYU1RYMUKA`

some of the needed features for the calibration process. During the execution of the software, two windows are presented to the user. Through the filtration of the original 3D point cloud, where the limits of the filtration are given by the user in the configuration file so that it catches the points that correspond to the boards, the points are projected and visualized in a first window. In this window the user selects the points that are the closest to be the corners of the board. The other window shows quadrilaterals that should correctly mark the card's line segments and confirm that we are defining the points that we really need.

However, the calibration software provided failed to generate the two above-mentioned windows with our multisensorial setup, and thus we could not define the needed points.

Originally, the author used a 16-layer Velodyne LIDAR (VPL-16)[15] and a ZED[16] camera. Although they are among the most used 3D LIDARs and outdoor depth cameras, we believe that there may be an issue with the software version used to develop the registration method with ROS and/or OpenCV and the version we are using, or a hard assumption on the underlying hardware used by the author of the package. In order for calibration to be done regardless of the devices used, we have developed additional software that allows the user to obtain all the necessary points. These points are key to the whole calibration process and can be calculated outside the original software to be later supplied back to the ICP registration algorithm.

Hence, the approach for LIDAR-camera calibration proposed in this dissertation follows the data flow illustrated in Figure 3.1.

[15] `https://velodynelidar.com/products/puck/`

[16] `https://www.stereolabs.com/zed/`

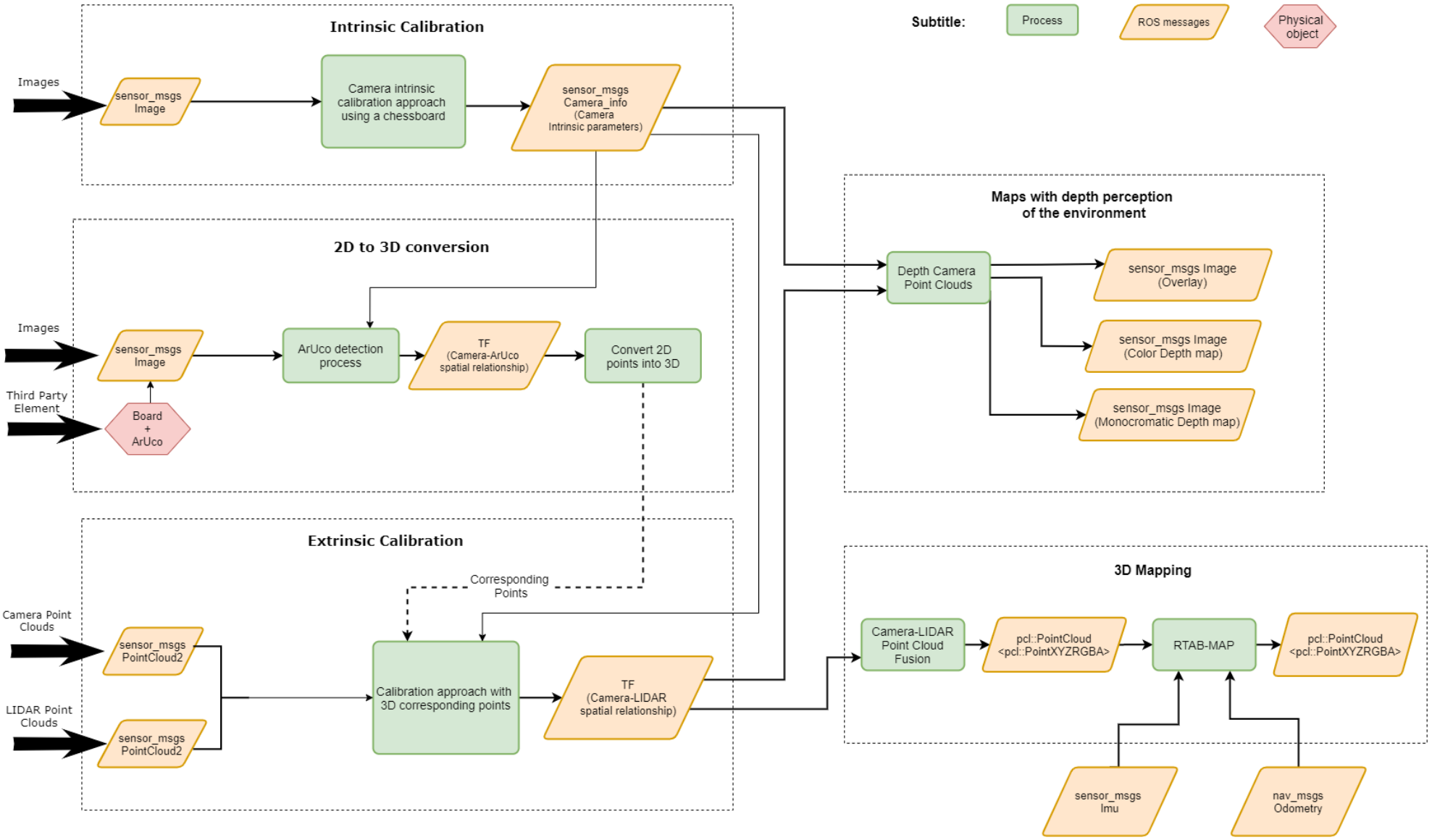


Figure 3.1: Registration and mapping data flow diagram.

3.1.1 LIDAR-RGB-D Registration

RGB-D cameras are a specific type of depth sensing devices that work in association with a RGB camera and that are able to augment the conventional image with depth information related with the distance to the sensor. These cameras can separately produce depth maps that give per-pixel depth information and an RGB image of the same environment. Then they can produce not only point clouds but also colored point clouds by calculating which color of each pixel in the image corresponds to the 3D point of the point cloud.

In ROS, there are several tools like `rviz`, `webviz`, `rqt_graph` and `ROS-mobile` that allow us to view, edit, understand and interact with different types of data. Rviz turns out to be the most intuitive and powerful tool that combines, into a single view, data such as images, point clouds, coordinate systems, geometric transformations, odometry, robot models, and also supports the visualization of techniques such as mapping, data fusion and path planning. In addition, it also allow us to make accurate measurements, view data in 360 degrees, select and restrict data, estimate a 2D pose, send a 2D navigation goal to a robot and obtain 3D coordinates just by clicking on the desired point.

By using boards that stand out from the rest of the objects we create our desired features. The boards, their edges and corners are easily detectable and recognizable by the human eye in point clouds, as we can see in Figure 3.2, so this makes the manual process of obtaining 3D points (the user selects the desired points) very reliable and with little uncertainty. Therefore, corners are used as feature points as they are easily and precisely determined. Another very important property is that by knowing the coordinates of just one point on the board and knowing the measurements of the sides of the boards a priori, we can get a sense of the coordinates of the remaining corners.

Once we know how to obtain the 3D points, we need to apply the Kabsch algorithm [69]. We define a set of corresponding points, let A be the points of one sensor, B the points of the other sensor and N the amount of points pairs:

$$A = \{a_1, a_2, ..., a_N\} \qquad B = \{b_1, b_2, ..., b_N\} \tag{3.6}$$

Calculate the centers of mass for the corresponding points, P is a group that contains these points:

$$\mu_A = \frac{1}{N} \sum_{\{i,j\} \in P} a_i \qquad \mu_B = \frac{1}{N} \sum_{\{i,j\} \in P} b_j \tag{3.7}$$

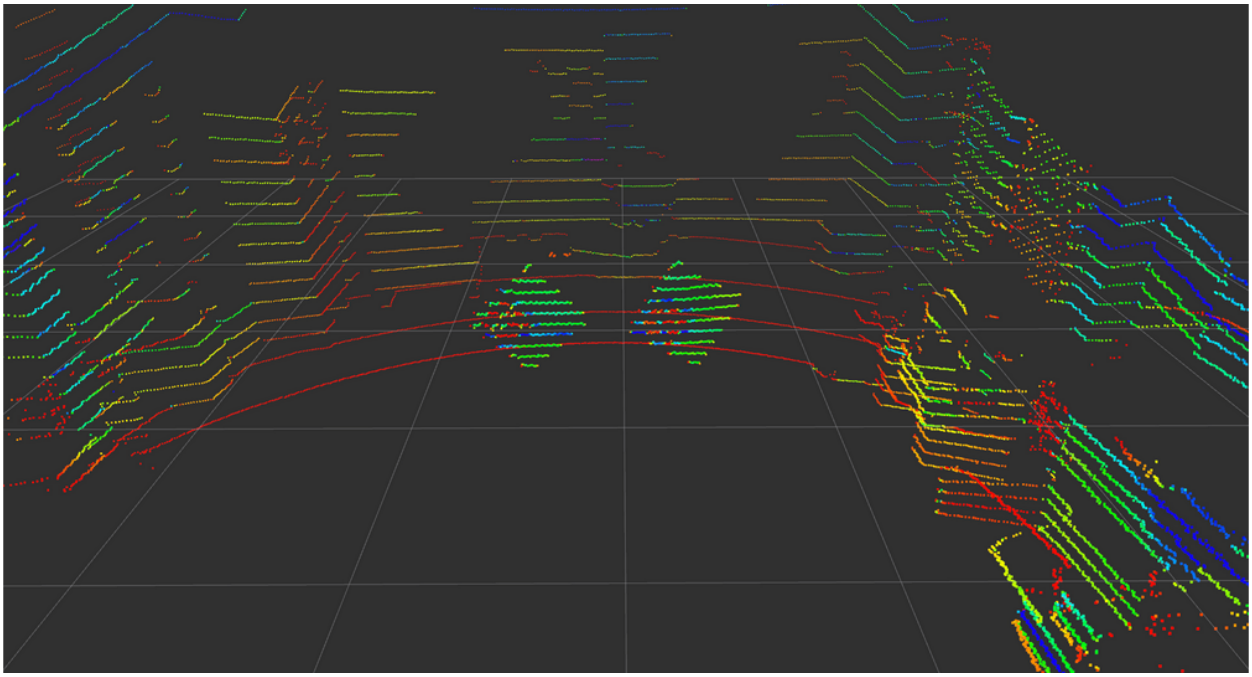


Figure 3.2: Visualization of a 3D LIDAR point cloud with the presence of boards and markers. The boards are easily distinguishable from the rest of the scene and we can clearly see the corners of the cards. The colors are indicators of the amount of light reflected by the objects [19].

Then subtract the corresponding center of mass from every point:

$$A' = \{a_i - \mu_A\} = a'_i \qquad\qquad B' = \{b_j - \mu_B\} = b'_j \tag{3.8}$$

With that, we want to get the R and t that minimizes the distance between corresponding points, which is the same as minimizing the sum of the squared errors:

$$E(R,t) = \sum_{\{i,j\} \in P} ||a_i - Rb_j - t||^2 \tag{3.9}$$

This equation can also be represented by a problem known as Orthogonal Procrustes:

$$E'(R) = \|[a'_1, a'_2, ..., a'_N] - R[b'_1, b'_2, ..., b'_N]\|_F^2 \tag{3.10}$$

The solution to this problem is equivalent to finding the nearest orthogonal matrix to a given matrix $M = A'B'^{\top}$. This cross-covariance matrix M must be calculated for each pair of corresponding points:

$$M = \sum_{\{i,j\} \in P} A'B'^{\top} \tag{3.11}$$

We can find the nearest orthogonal matrix using single value decomposition:

$$M = UDV^{\top} \tag{3.12}$$

The matrices U and V are 3×3 rotation matrices and D is a diagonal matrix. The rotation that best relates the two sets of points is:

$$R = UV^{\top} \tag{3.13}$$

However, to be a proper 3D rotation matrix it must belong to the group of rotations of a three-dimensional Euclidean space, SO(3). In this group all matrices are orthogonal 3x3 and have $det = \pm 1$. For cases where $det = -1$, the linear transformation is not preserved, instead it is reversed. To correct the matrix we use a correction matrix, C, which is a diagonal matrix such that $C = Diag(1, 1, -1)$:

$$R = UCV^{\top} \tag{3.14}$$

Now that we know the rotation between the centers of mass, the translation is a simple difference between the same centers of mass applying the rotation to one of them:

$$t = \mu_A - R\mu_B \tag{3.15}$$

3.1.2 LIDAR-RGB Registration

Unlike RGB-D cameras, monocular cameras do not, by themselves, allow us to know the depth of objects as they do not have access to stereo vision or range finders. Despite this, it is necessary to find the same corresponding 3D points as in the method with RGB-D cameras.

In order to cover the depth problem when using 2D images, markers associated with an augmented reality library called ArUco are used [70]. These markers together with the library allow us to determine the marker's pose in the environment through an image just by using the camera's intrinsic parameters. With this, we can find the 3D points in the camera frame and relate them to the LIDAR's 3D points with the same algorithm.

In this case, the boards are also necessary to be able to define the corner points as features but also to be the support for our markers. ArUcos printed on paper are glued to the boards because it allows the marker to always be well stretched and easily moved without distortions and ambiguity. In fact, as the paper is very thin, we can assume that the set formed by

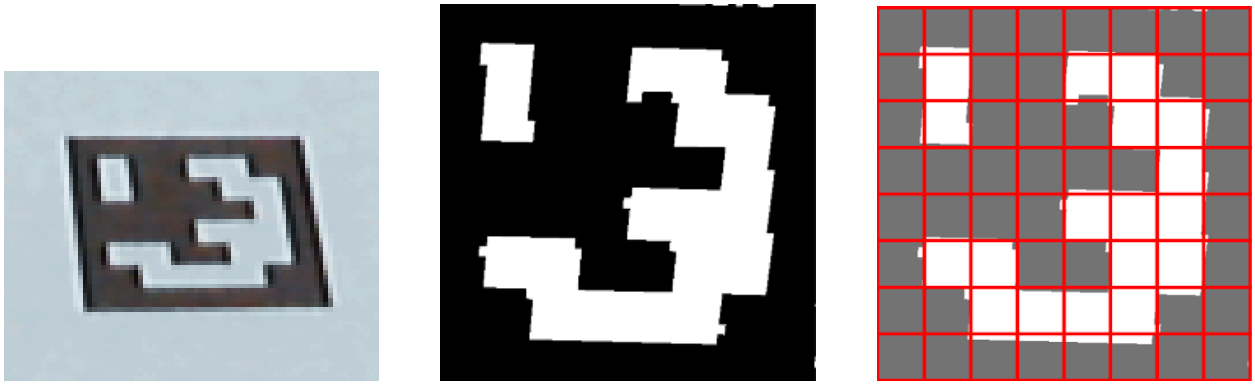


Figure 3.3: The first image is an ArUco taken from an image and in the second image we see the perspective removed. Finally, the marker is divided into a table that will be compared with the existing markers in the dictionary [71].

the board and the marker are on the same limited plane with known measures. Thus, by knowing only one point on the board, we can also determine the coordinates of the other corners very accurately using these measurements.

A dictionary of ArUco markers consists of a set of well-known markers formed through the different possible bit combinations of each square. On each cell, the mean of black and white pixels are counted to decide the bit assigned to the cell. All the bits are set to 0 or 1 depending if the mean value is higher or lower than 128, meaning it is a white or black square.

To detect a marker in a image the following steps should be done:

- We must find the corners of the marker;
- Correct the perspective distortion to obtain an image that looks as if the marker were seen from above;
- Divide the resulting image into a grid;
- Compare this grid of white and black cells with the given dictionary to find out if there is a match.

Most of the packages related to ArUcos compute these steps, some are represented in the figure 3.3. Another important aspect is that the markers must be quadrangular so that, by knowing the measurement of one side, it is enough to know the measurements of the remaining ones. These measurements are the key factor in calculating the markers depth with respect to the camera. After the respective markers are detected in the image, it is possible to know the marker's size in pixels. Then, by comparing the actual size and the

pixel size and by using the camera's intrinsic parameters, we can compute how far from the camera the marker is. Moreover, the *aruco_ros*[17] and *aruco_mapping*[18] packages accept as a parameter the size of the marker, which should be given by the user.

Now that we have understood how ArUcos allow us to relate a 2D image to an object in 3D space, let us use the packages to obtain the marker pose in the camera frame. The relative pose is given by a $[R|t]$ matrix that relates the camera frame with the marker's center. After configuring all parameters according to our setup, the pose is published in a ROS topic and we can even view the camera and marker frames in the rviz in the same 3D space.

We can apply a geometric transformation with the rotation and translation matrices in order to have the 3D points in the camera frame:

$$P_{camera} = {}^{camera}T_{board} * P_{board} \tag{3.16}$$

where P_{camera} is the 3D board point in the camera frame; ${}^{camera}T_{board}$ the $[R|t]$ between the camera and the marker; and P_{board} the 2D board point defined by manual measurements.

Although the points on the board are obtained through measurements that are normally associated with precision errors due to human action, perfectly linear objects are used to reduce these types of measurement errors.

3.1.3 LIDAR and Camera Data Merging

Now that we have addressed registration and calibration, we look into how to combine LIDAR and camera data in a single representation that gives us a more complete information. As registration allows us to know both devices data spatial relation, it becomes straightforward to merge them. By combining RGB images with point clouds, we can create useful data, namely *colored point clouds* and an *image with an overlaid sparse depth map*. The first one allows us to make use of RGB-D mapping techniques and the last one allows us to verify the accuracy of both registration and calibration.

Depth map is an image that contains information about the distance that objects in the scene are from a certain reference viewpoint. Depth maps are often related or even analogous to the terms depth buffer, Z-buffer, Z-buffering and Z-depth commonly used in 3D computer graphics and computer vision [72]. The Figure 3.4 illustrates these types of maps and the depth perception they give us.

[17] `https://github.com/pal-robotics/aruco_ros`

[18] `http://wiki.ros.org/aruco_mapping`

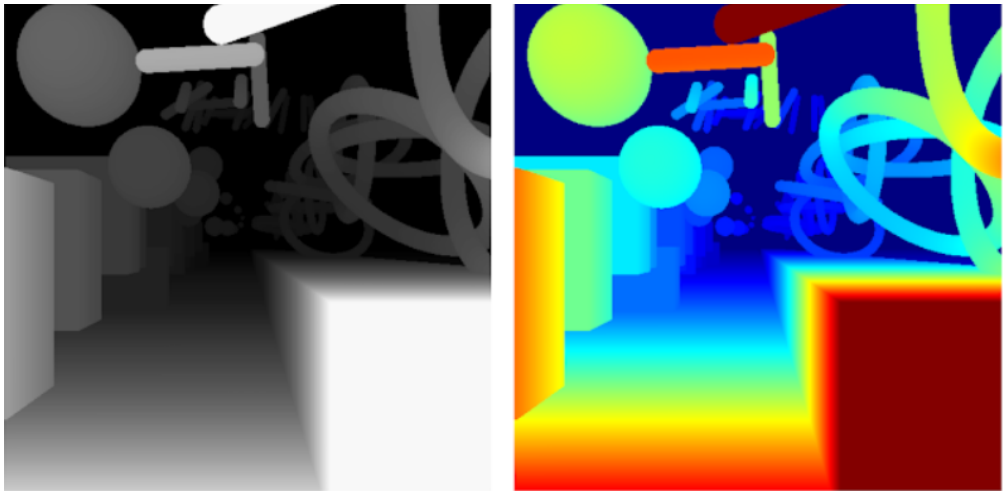
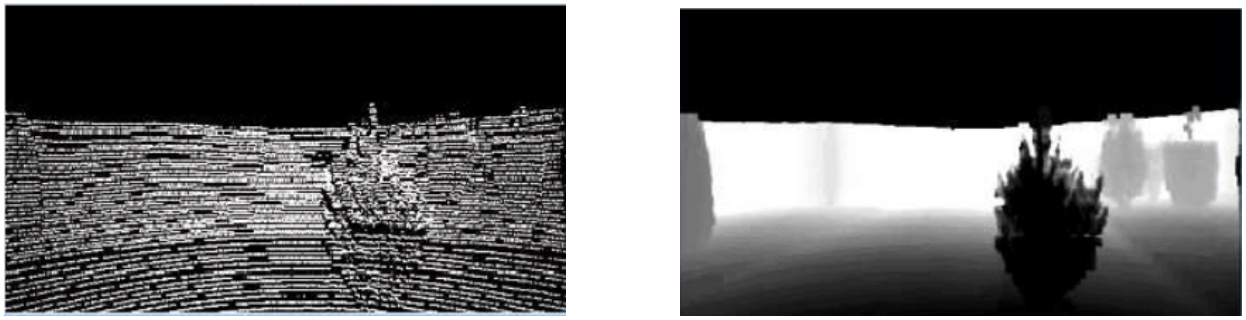


Figure 3.4: On the left a monochromatic depth map and on the right the same depth map but colored. We can see that, although they represent the same space, in the colored map the difference in depth is more noticeable [73].

There are two main ways to represent a depth map: gray level map and colorized map. The first represents the depth difference through luminosity variations, i.e., nearer surfaces are lighter and further surfaces are darker. The second represents this difference through a rainbow of colors that varies between RGB channels depending on the depth value. The two ways can also be represented in reverse, i.e., from darker to lighter and between BGR channels, respectively. Gray-level maps are used with monochrome images with only one color channel ranging from 0-255, with 255 being white, 0 being black, and intermediate values being variations between dark and light gray, hence the name. RGB depth maps have the same process but through 3 color channels, forming the different color combinations we know.

Although they are both widely used (even simultaneously), colorizing images helps the human visual system pick out detail, estimate quantitative values, and notice patterns in data in a more intuitive way. In addition, color maps can be directly related to other RGB images. However, gray level maps are simpler and with less computational power. While the color map is more used for visualization, the gray level map is more used for processing in depth assignment techniques. For instance, the RTAB-MAP approach [74] uses use depth images in gray levels to obtain the depth of each point through a normalization between the maximum depth value obtained by the device and the gray value of the point to be processed. Since there is only one color channel, normalization is sufficient to know the depth that the color really represents, then becoming a very straightforward method. With color depth maps, this process becomes more complex due to the three different color channels that



(a) Sparse depth map.

(b) Dense depth map.

Figure 3.5: Sparse and Dense depth map representations [75].

would have to be analyzed for each point.

Depth maps can be either *sparse* or *dense*, both illustrated in the figure 3.5a and figure 3.5b respectively. Sparse depth maps require a lower computational power because they usually have fewer points than dense maps, but are sufficient to be able to locate cameras in the world. Current real-time 3D reconstruction systems tend to solve the sparsity problem through SLAM, avoiding expensive dense representation. Dense maps demand greater computational power while giving a complete representation of the scenario, being often used in cases where surfaces (formed by edges, corners, blobs) are an essential factor.

Image with Sparse Depth Map

Taking advantage of LIDAR data, we can create a sparse image with the points projected in 2D. This image gives us information about the depth assigned to each pixel, i.e., the relationship between the 3D point and the pixel in the image. The 3D LIDAR provides discrete points in 16 scan layers, which are not enough to cover a whole image. Despite not generating dense data, the layers are close enough to allow to perceive entities in the scene with some clarity.

The creation of an image that merges the LIDAR projected 3D points and the points of the camera image, both in pixels, allows to visually evaluate the accuracy of the calibration. This assessment is made by observing whether the sparse points of the LIDAR are positioned in the camera image pixels that correspond correctly with the objects in the scene. For example, we check if the LIDAR points that correspond to a tree are really projected on the camera image pixels that also correspond to the same tree, and we do this for any object that is easily recognizable. The figure 3.6 demonstrates the idea described using a person as a distinguishable element. We can conclude that the LIDAR-Camera calibration is accurate because the depth map points (the projected LIDAR points) corresponding to the person

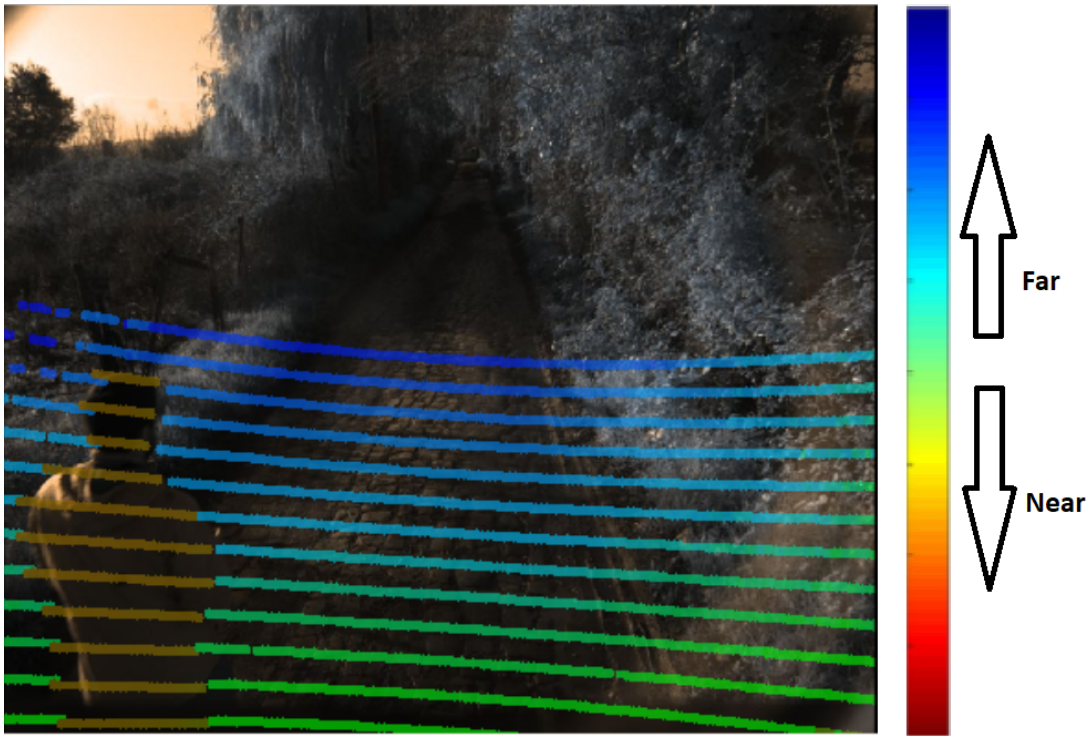


Figure 3.6: Colored sparse depth map overlaid on the camera image. The colors of the depth map are defined in RGB form, where more red means closer, blue more distant and combinations of these two colors with green mean average distances to the camera frame.

are positioned in the pixels of the camera image that also correspond to the person.

In order to create this image we only need the matrix that relates the two devices and the intrinsic parameters of the camera, the first one to transform the 3D points in the LIDAR frame to the camera frame, and the second to project 3D points to the image pixels. Let P be the projection and $[R|t]$ the Camera-LIDAR transformation matrices; IP the 2D Image points; $k_{1,\dots,6}$ and $p_{1,2}$ the radial and tangential coefficients. The structure of the code developed is described in the Algorithm 1.

RGB Point Clouds

RGB point cloud are a 3D representation of the perceived world where each point has the RGB color of an image. The combination of the 2D colored image with 3D points opens up a range of new possibilities, such as building 3D colored maps with real-world colors. Some stereo cameras, like the Intel Realsense included in the Ranger's sensing kit described in Chapter 2.10, provides utilities that build RGB point clouds using both embedded cameras and stereo vision. The same process is not possible for monocular cameras, so the combination of LIDAR and images is again the solution for building RGB point clouds with cameras

Algorithm 1 Merging LIDAR and camera image. Color definition example for a black/white depth map.

1: Read a camera image
2: **for** $iteration = 0, 1, \ldots, size(point\_cloud)$ **do**
3: Read a LIDAR point (XYZ)
4: point_in_camera_frame = [R|t] * point_in_LIDAR_frame
5: image_point = (u,v,w) = K * Rec * point_in_camera_frame
6: image_point normalization
7: Distortion correction (see equations 2.2 and 2.3)
8: **if** point in image **then**
9: depth = $\sqrt{X^2 + Y^2 + Z^2}$
10: color = $min(255, 255 \left\| \frac{depth - device_{max\_depth}}{device_{max\_depth}} \right\|)]$
11: Draw a circle on image pixel (u, v) and with the the computed color
12: **end if**
13: **end for**

that do not have depth information.

The creation of the RGB point clouds follows many of the same principles of merging LIDAR and Image data into a single image. It is also needed to project 3D points to the image 2D plane to find which pixel corresponds to which point on the LIDAR. The difference turns out to be the final output, instead of overlaying points on an image, we will extract the RGB data from the image and assign that color to the 3D point of the point cloud. This process can then be described by the following steps:

- Transform 3D LIDAR points to the camera frame.
- The point cloud points that are not within the camera's field of view (FOV) are excluded.
- 3D to 2D projection is performed and the corresponding pixel is found for each point.
- RGB data is extracted from the 2D image.
- RGB values are assigned to points in the point cloud.
- Back-projects to LIDAR space.

Taking into account the algorithm 1, the differences would be that in step 11, instead of defining a color according to the depth value, we extract the color of the current pixel and in step 12 we just assign those colors to all the corresponding 3D points of the LIDAR.

The software described in this section, as well as a detailed tutorial on how to deploy the software is compiled on the SEMFIRE project repository[19].

[19] `https://gitlab.ingeniarius.pt/semfire_ing_uc/perception/lidar_camera_pkg`

4 Experimental Validation

In order to evaluate and validate the performance of our system, experimental tests were done both in datasets and in real time using the robot sensing kit. As our system is interconnected, i.e. we execute the registration and use the results to feed the 3D mapping approach, i.e. RTAB-Map, the most correct way to validate our system is to validate the registration first and reinforce this validation by analyzing the final result of the mapping.

Throughout the experiments, different experimental scenarios were needed and different setups were used. In this chapter we will describe them separately as well as the reasons that promoted them.

4.1 Experimental Setup

The Ranger forestry UGV, which is the robot used throughout this work, has a set of sensors that are fixed so their poses, including their positions and orientations, with respect to the robotic system do not change. Multi-sensor registration is done with pairs of sensors, including at least one LIDAR and completed with a monocular camera or a stereo camera.

Recalling section 2.3.2, the setups for registration can be separated into:

- Intel RealSense D435 RGB-D camera with 3D LIDAR, for *LIDAR-stereo camera registration.*
- Dalsa multispectral camera with 3D LIDAR, for *LIDAR-monocular camera registration.*

Data-gathering and tests took place at Ingeniarius Ltd, the coordinator of the SEMFIRE project, on the outdoors (on the street and sidewalk) and on a nearby rural path. For data-gathering, the ROS functionality *rosbag* was used to record datasets[20]. Datasets in ROS are recorded in bag files, which are used for storing ROS message data and that can be reused

[20] `http://wiki.ros.org/rosbag/Commandline`

infinitely. The entire data flow is saved with the same synchronization as the one performed in real time and is not vulnerable to any changes. Therefore, the rosbag command-line tool provides a safe and reliable way to create, process, analyze and visualize datasets.

To run the sensor drivers and record the datasets in real time we used the Mini-ITX computer embedded in the Ranger (see specs in Section 2.3.2). In order to develop software and to work on datasets without using Ranger's hardware, we used an Asus GL552VW laptop equipped with an Intel Core i7-6700HQ CPU, 16 GB RAM and NVIDIA GeForce GTX 960M.

Intel cameras have well-documented support and software integration in ROS. In order to launch drivers, produce data with multiple cameras, build 3D Point Clouds through stereo vision and among other features and information on these devices, one should follow the installation and deployment instructions of the Intel RealSense team on github[21].

before we start devising the calibration scenario we need both boards and ArUco markers. ArUcos must be square and large enough for the two sensors to be able to detect them. The boards can have any size as long as they can fully support the markers and also be easily detected by the sensors. ArUco's IDs have to be different from each other, as long as they are from the same dictionary we should be able to detect both. To create, detect or learn the dictionaries using OpenCV please refer to the supporting webpage[22]. ArUcos markers must be printed on strong and consistent paper that does not tear or fold easily.

4.2 Experimental Scenarios

According to the method we describe in section 3.1, we need to be able to extract a set of corresponding 3D points in the LIDAR and camera frames. As the boards will be the main agent to achieve that, we require their poses to be unambiguous, while being as precise as possible in the 3D point extraction step.

A setup based on the Ankit Dhall *et al.* approach [19] was used for both LIDAR-camera calibrations.

Scenario for LIDAR-stereo camera registration

The apparatus for the LIDAR-stereo camera registration involves a room with enough light, no wind and with walls or objects on the sides where you can attach and stretch a rope, or

[21] `https://github.com/IntelRealSense/realsense-ros`

[22] `https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html`

Figure 4.1: Setup used to calibrate the Intel D435 camera with the 3D LIDAR. It was set up inside a room where there is no wind interference so the boards remain static throughout the calibration process.

any other similar tool, to support the boards. A suitable place that meets all requirements is, for example, a room or a garage, as seen in figure 4.1. The same setup perceived by the 3D LIDAR can be seen in Figure 4.2.

In the RGB-D camera case, only the board itself is needed. However, as we are going to use the same boards with ArUcos in another calibration and as they do not interfere in this calibration, we can reuse it here.

Scenario for LIDAR-monocular camera registration

For the LIDAR-monocular camera registration, which captures image data within specific light wavelength, including infra-red and ultra-violet. Unlike the registration mentioned above, the garage light was not enough to be able to perceive the environment with the camera's filter. Thus, the scene had to be set up in a place with natural light on a sunny day, a condition that was not relevant with a usual RGB camera. Also, as the setup is set up in an environment more vulnerable to the weather and as we depend on the setup to

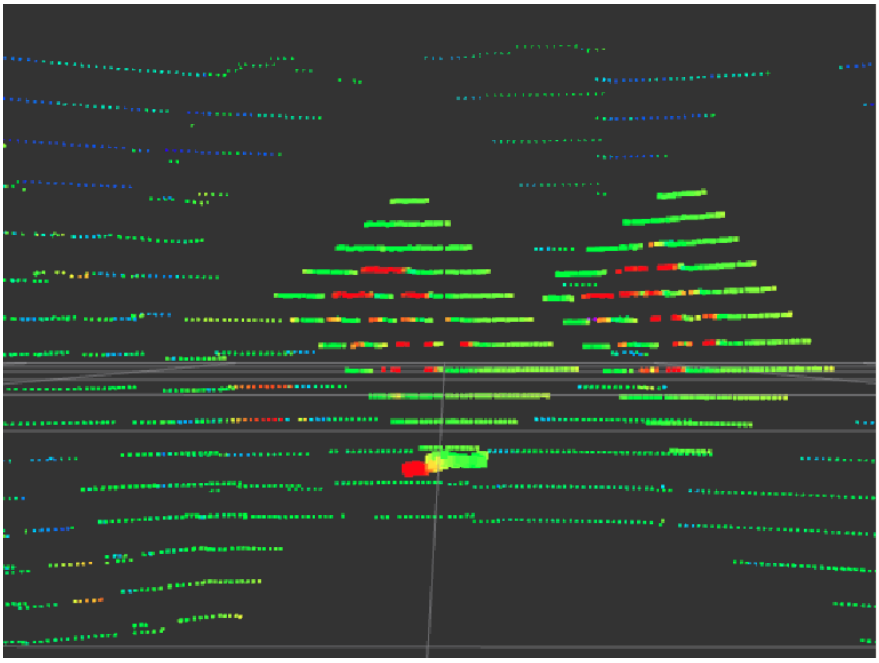


Figure 4.2: The setup from image 4.1 perceived by 3D LIDAR. The boards are fully distinguishable from the rest of the environment and it is possible to easily detect their corners and edges.

be stable and fixed, it must be on a day without much wind so that the boards' poses are stable. In Figure 4.3 we can observe the setup perceived by the monocular camera and, in the Figure 4.4, the result of the ArUcos detection in that same setup. Then, in Figures 4.5, 4.6 and 4.7 we can observe three different views of the setup perceived by the 3D LIDAR.

A very important aspect is that the paper with the printed ArUco needs to be well stretched without air bubbles and without any relief so there will be no errors or inaccuracies in detecting the markers.

Figure 4.3: Setup used to calibrate the multispectral (monocular) camera with 3D LIDAR. It is similar to the setup used for stereo cameras but ArUcos are now a key agent for calibration. Human intervention was required to keep the boards static because the test was done in a moderately windy environment.

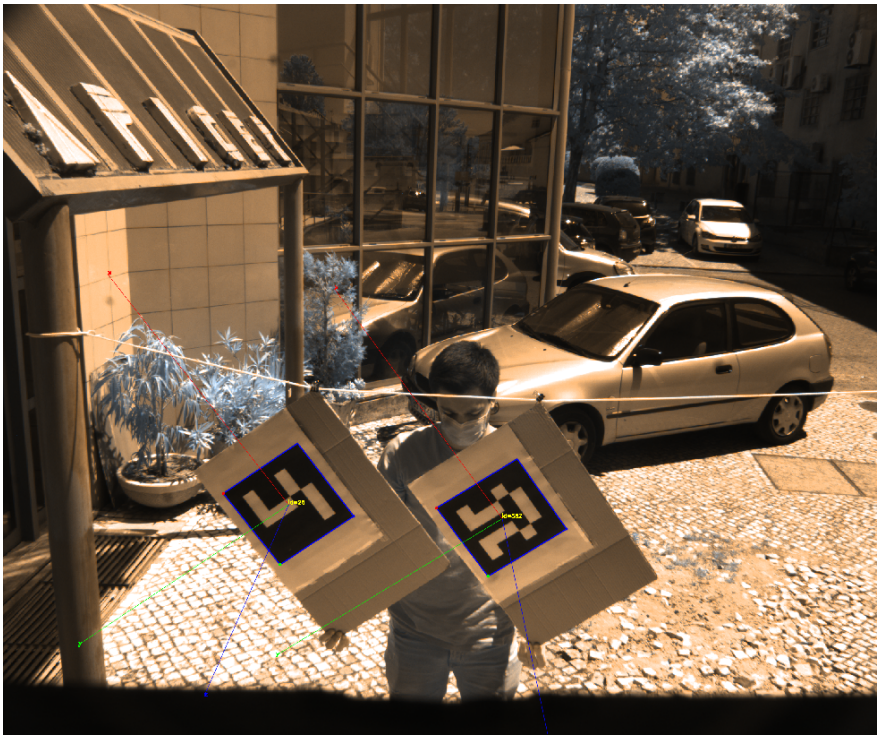


Figure 4.4: Detection of ArUco markers. It is possible to know the center of the marker, an axis that represents the plane that includes each pair of markers with their board, the marker ID and even an outline of the marker format.

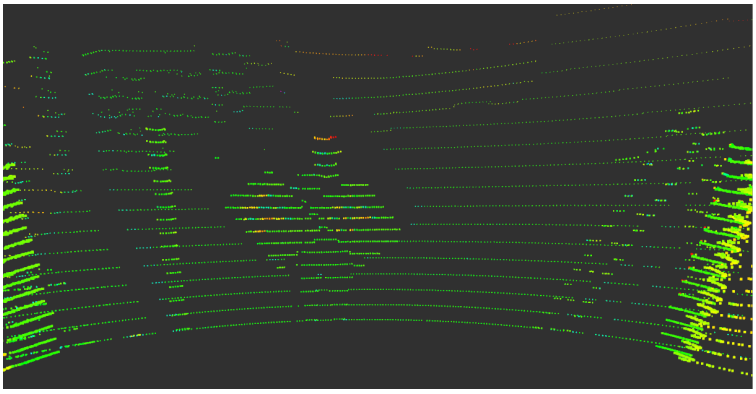


Figure 4.5: Front view of the outdoor setup perceived by 3D LIDAR. The corners and edges of the boards are well detectable. In this view, the body can make reading the points more complicated.

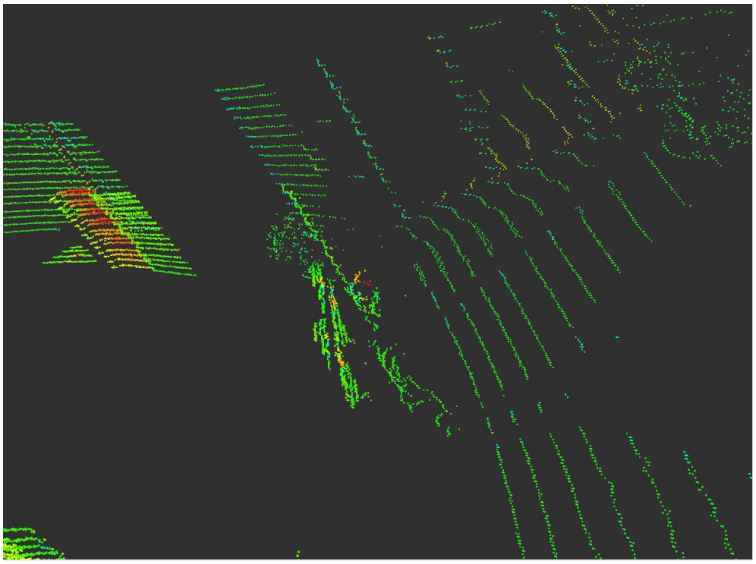


Figure 4.6: Side view of the outdoor setup perceived by 3D LIDAR.

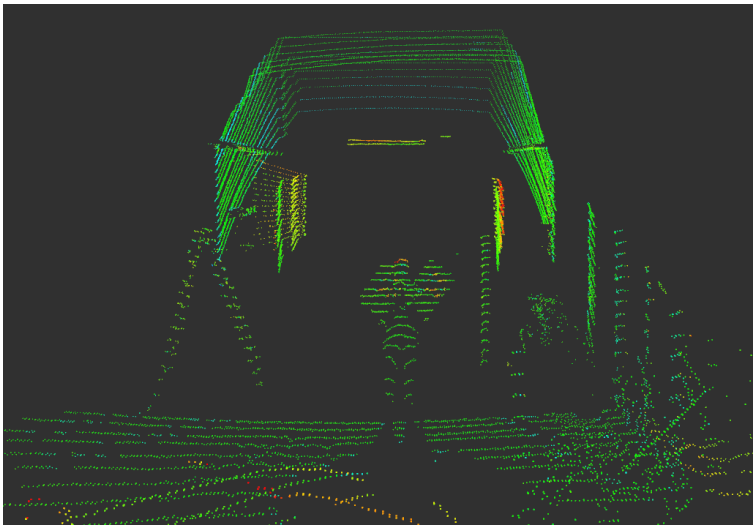


Figure 4.7: Back view of the outdoor setup perceived by 3D LIDAR.

4.3 The RTAB-MAP 3D Mapping Approach

In section 2.2, we discussed about several different mapping methods. Despite the variety, it is not possible to define a single method that presents the best results for all robotic systems and their components, for all scenarios and possible adversities, for all types of devices and data, and for all types of processing time and computational power requirements.

RTAB-Map is a graph-based SLAM approach that has been integrated in ROS as a package[23]. It is an open source library implementing loop closure detection with a memory management approach so that, by limiting the size of the map, loop closure detections are always processed under a fixed time limit, thus satisfying real time processing in large-scale environment mapping [74]. The sensor's data is received and synchronized by the RTAB-Map node. Then, in addition to odometry and other useful data, they are stored in a short term memory (STM) to be later reused by the next nodes for Loop Closure and Proximity Detection. When new data is obtained and new nodes are created, graph optimization carries the computed error to the whole graph, decreasing odometry drift.

In our work, the odometry supplied to the RTAB-Map is generated by the robot by using a visual odometry fusion method (intel RealSense), with laser scan matching (LeiShen C16) and with data from an IMU. However, localization is not a part of the scope of this dissertation work.

RTAB-Map is the most suitable mapping method for our work for the following reasons:

- It is an open source and proven approach with full integrate in ROS.
- It is a RGB-D, Stereo and Lidar Graph-Based SLAM approach which matches perfectly with the hardware used in our robotic system.
- Since our goal is to have a real-time system, this approach meets this requirement by having memory management.
- Supports the inputs provided by our system for mapping, namely 3D depth data from the LIDAR together with colored images from a forward-facing multispectral camera.

Benchmarking works of graph-based SLAM methods have shown that RTAB-Map is one of the methods that simultaneously has more precision and that maps with higher quality [76]

[23] `http://wiki.ros.org/rtabmap_ros`

[24] `https://www.youtube.com/watch?v=L8Jz4Q3KQNM&ab_channel=ActaScholaAutomataPolonica`

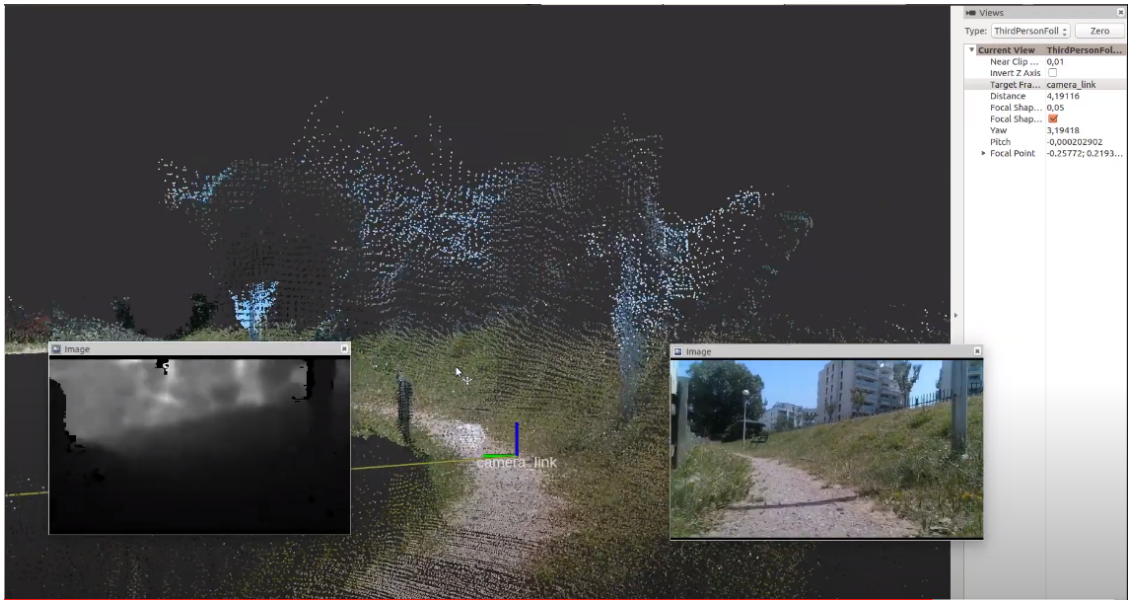


Figure 4.8: Example of mapping results with RTAB-Map using an Intel D435i camera. The map is visualized in rviz along with the camera depth image (on the left) and the RGB image (on the right). Example extracted from an online video[1].

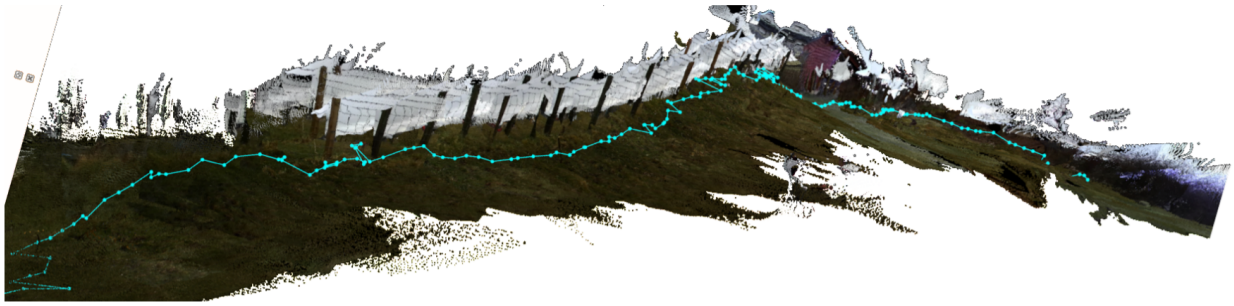


Figure 4.9: High-quality 3-D reconstruction from RTAB-Map of a farm field. Image from [78].

[77]. It is also a widely used mapping method so there are several works that demonstrate acceptable results, as can be seen in the figure 4.8 and figure 4.9.

We will use RTAB-Map to verify that our calibration and registration method can be used for building maps.

4.4 Results and Discussion

Registration

At a starting point in our experiments a manual measurement was made, using measuring tape to estimate the translation between the sensors. We estimated rotation using rviz and

Figure 4.10: Setup for LIDAR-Stero camera registration validation.

test the rotation values until we found acceptable results in reghards other words, knowing that the two sensors perceive some plans in common, such as floor, walls and ceiling, we tried to align the plans as best we could.

The realsense stereo camera used in this work (more details in section 2.3.2) provides us with a depth cloud. This depth cloud contains information about the distances of objects to the camera, just as the LIDAR point cloud contains the same information to the LIDAR. Thus, we should observe the points of both point clouds superimpose on the same objects in the space that they perceive. Therefore, to validate the result of our LIDAR-Stereo camera registration method, we compared the intersection of the depth cloud with the LIDAR by using a setup with some easily distinguishable elements from the environment (as seen in Figure 4.10), such as people, chairs, boards, etc. The Figures 4.11 and 4.12 illustrates both superimposing clouds in which, we can verify that there is a reasonable of intersection, namely in the most distinctive objects such as boards and people. LIDAR's points (in white) follow the shapes of the corresponding objects represented by the depth cloud, for example the boards and the person (in red and yellow in the depth cloud, respectively), indicating an acceptable result for our calibration.

By overlaying the LIDAR data on the camera image based on the transformation between sensors that was manually acquired,we can create a depth map where the depth of objects in the real world, given by the point cloud, determines the image pixel color, we can verify whether the depth map points are accurately overlapping on the corresponding objects. Therefore, if an element of the environment is at a very different distance to the camera from

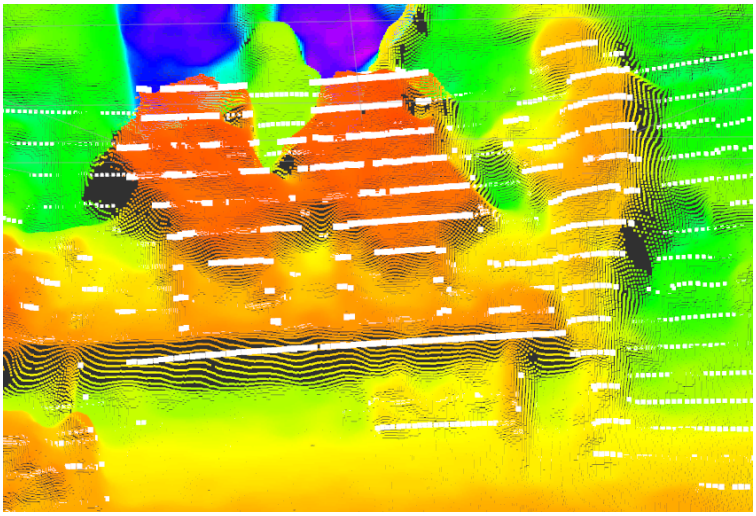


Figure 4.11: Front view of the stereo camera's depth cloud with the LIDAR's point cloud. The LIDAR's points are shown in white and the depth cloud has a RGB color scale which varies with the distance that the objects are from the camera.

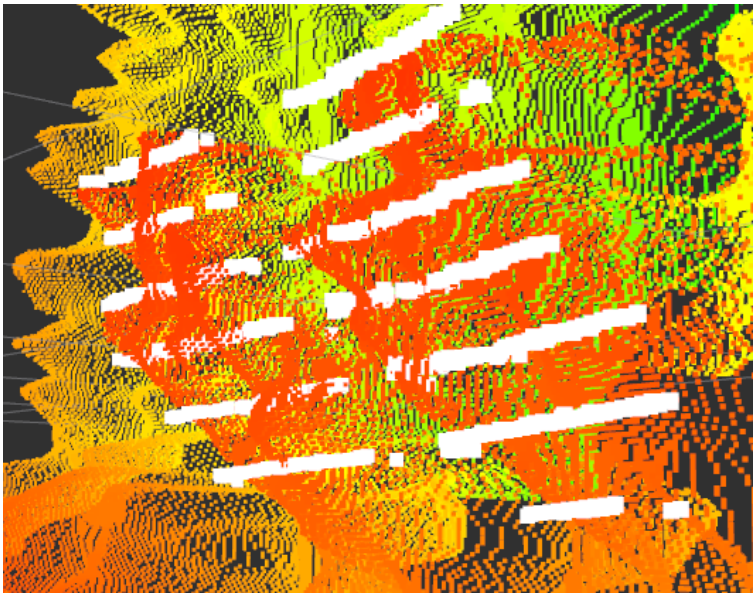
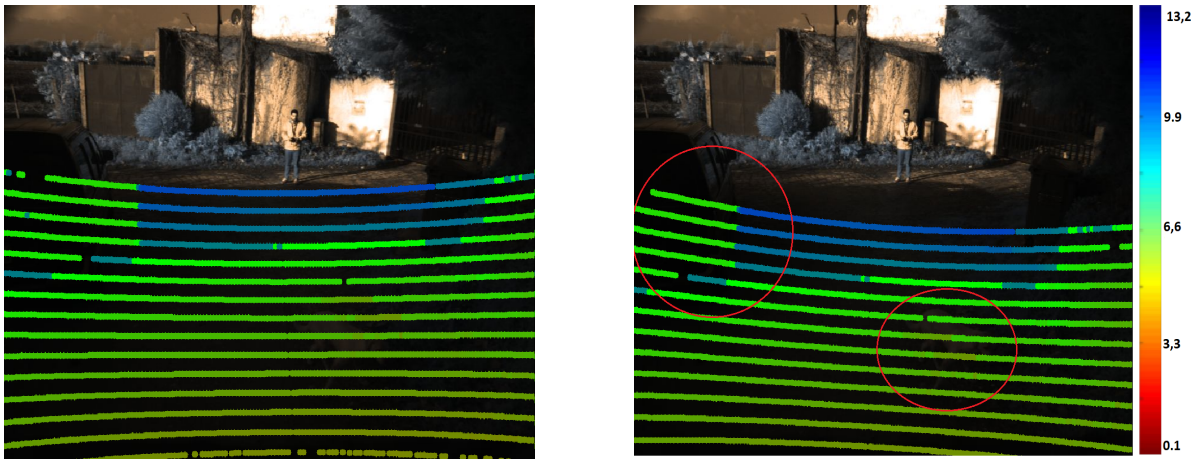


Figure 4.12: Closer view of the board points in the stereo camera's depth cloud and in he LIDAR's point cloud. The intersection of the points of the two sensors on the boards is visible. In red, the boards perceived by the camera and, in white, the same boards perceived by the LIDAR.

the rest of the elements that surround it, we should observe not only a wide variation in color but its shape correctly defined. This provides hints that calibration has been conducted appropriately. The Figures 4.13, 4.15, 4.17, 4.19 and 4.21 are the original camera image. The Figures 4.14, 4.16, 4.18, 4.20 and 4.22 show and compare the results of this process with manual calibration and calibration with the proposed method. In general, it is clear that

Figure 4.13: Raw image with car and dog.



(a) With manual calibration.

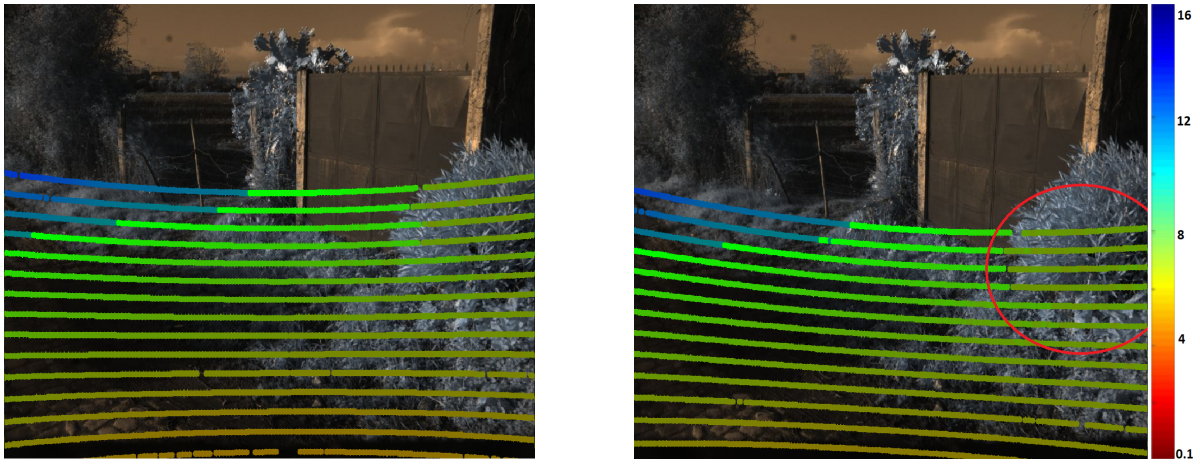
(b) With our calibration method.

Figure 4.14: With our method the points of the car (top left) and the dog (middle below) are correctly overlapping in the corresponding pixels. We can also see that the blue points (top half) correspond correctly to the width of the path while intersected by the car. The color scale is in meters.

with our method the points correspond with much more precision in the different elements, including a dog, a car, a plant and a person. The regions marked by red highlight the improved results compared to manual calibration.

Our method is influenced by possible camera calibration process errors and so the camera's intrinsic parameters may not be perfectly determined. Thus, some points can be projected in such a way that they are outside or slightly out of step with the supposed pixel.

Figure 4.15: Raw image with a plant.



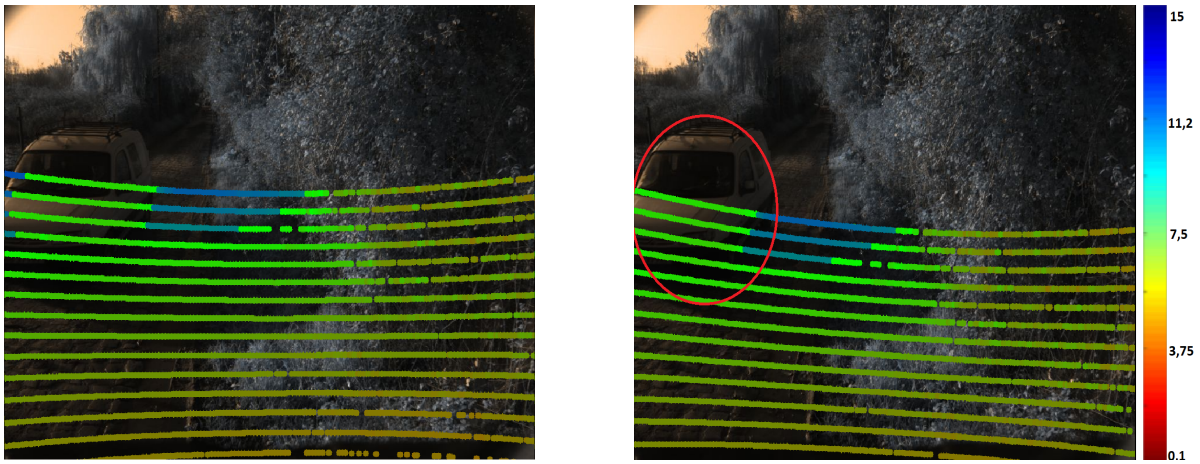
(a) With manual calibration.

(b) With our calibration method.

Figure 4.16: Although at first glance it appears that the manual registration puts the points of the plant (middle right) in the right place, they are actually not. Some points in the manual calibration are not correctly on the plant boundaries, with our method this does not happen. The color scale is in meters.

In order to verify that the strategy (see Figure 4.23) of extracting the points in the calibration step manually using rviz is precise, despite being made by human action in rviz, the same points were extracted several times with the same dataset for the same timestamp. The results can be checked in the table 4.1. In all the iterations of this process, the coordinates of the extracted points were very close to each other as confirmed by the very low value of the root mean square error (RMSE). Therefore, we can validate that this

Figure 4.17: Raw image with a car.



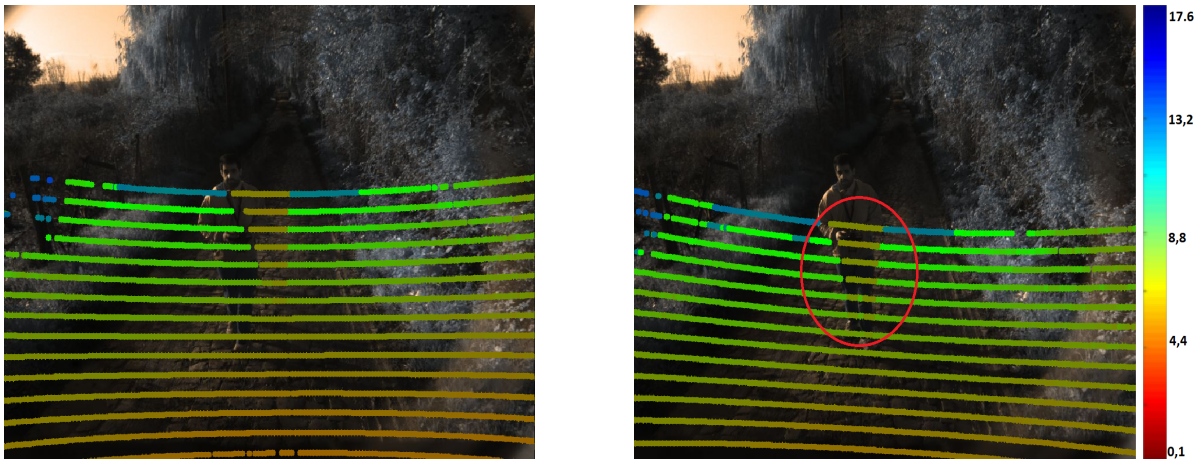
(a) With manual calibration.

(b) With our calibration method.

Figure 4.18: This is another case where manual calibration seems to put the points as intended. The points superimposed on the van seem to correspond to the back side of the van, however with our calibration we verify that they actually correspond to the front side. Also the blue points on the left side in 4.18a correspond to very distant dots but they are on top of the van. With our calibration we verify that all points of the car have no color that ambiguously represents its distance from the camera. The color scale is in meters.

process is very accurate and is not affected by the number of times it is done.

Figure 4.19: First raw image with a person.



(a) With manual calibration.

(b) With our calibration method.

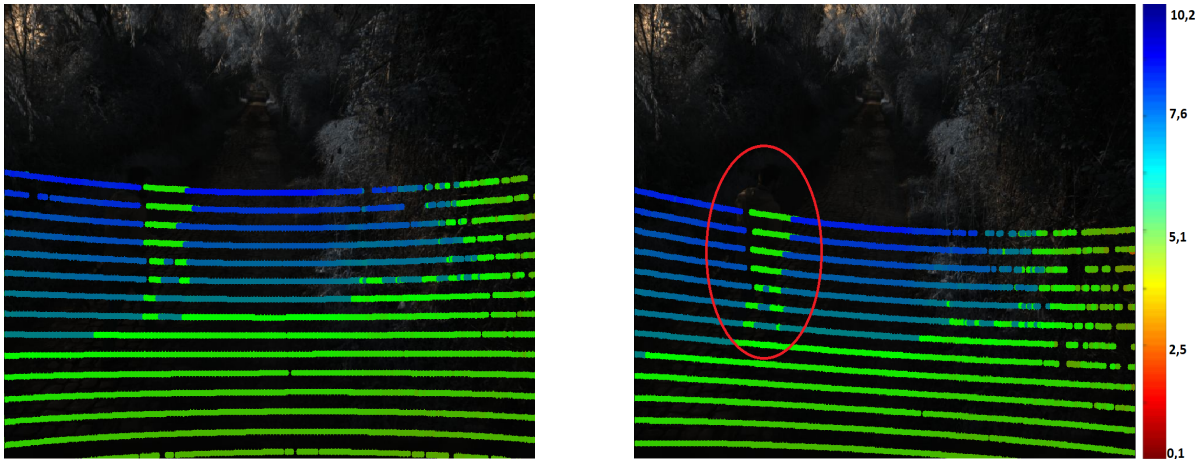
Figure 4.20: In these two images the improvement of the results with our calibration is very noticeable. The points are correctly superimposed on the person, those same points correctly define the shape of the body and the surrounding points corresponding to the path start as soon as the body contours finish. The color scale is in meters.

Mapping

In order to map the environment using the useful information from the multispectral camera's image and fulfilling the input requirements of RTAB-Map, the package `pixel_cloud_fusion`[25] was used. This package was developed for the Autoware.AI project, which provides an open source repository, targeting autonomous vehicles and perception software. With the soft-

[25] `https://github.com/Autoware-AI/core_perception/tree/master/pixel_cloud_fusion`

Figure 4.21: Second raw image with a person.



(a) With manual calibration.

(b) With our calibration method.

Figure 4.22: Another clear example that shows the errors of the manual calibration and improvements that our calibration provided. The color scale is in meters.

ware mentioned, we can form a colored point cloud with the colors of an image by using the camera's intrinsic parameters and the depth estimation using the 3D LIDAR via the transform that related both sensors. Thus, the point cloud formed has both depth and multispectral data. Providing this colored point cloud to the mapping approach, together with the camera's intrinsic parameters, allows us to build a colored map of the environment based on information from the multispectral camera and the 3D LIDAR.

The Figure 4.24 shows the main input data (left), parameters (top) and output data (right).

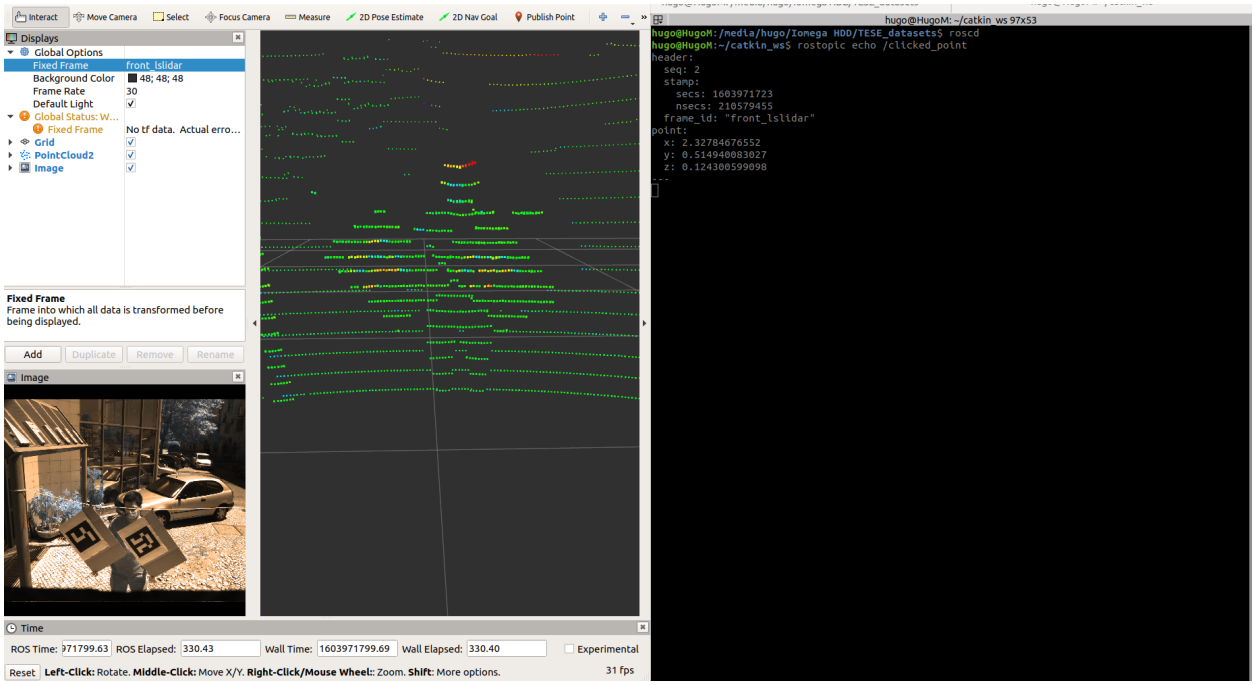


Figure 4.23: Example of the point extraction process in rviz. With the `publish_point` tool from rviz we can select a point in the point cloud and know its coordinates in the topic `clicked_point`.

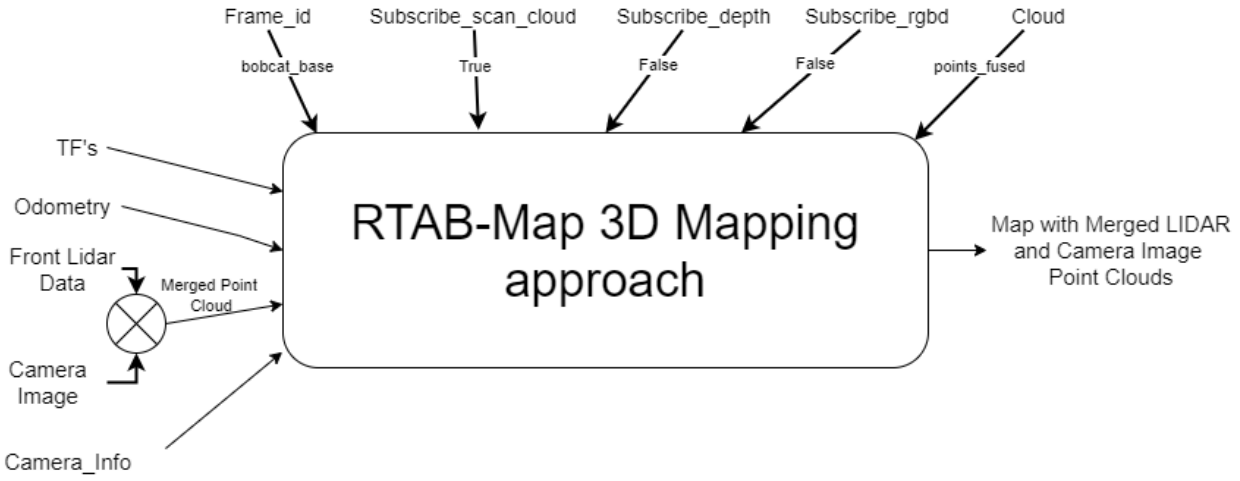


Figure 4.24: Input data, parameters and output data from RTAB-Map.

The inputs represent the data with which we need to feed the RTAB-Map. Odometry allows us to estimate the position of the robot relative to a starting location. Camera_info, with the camera's intrinsic parameters, is used in nodes related to camera images, in this case to be able to relate an image to a point cloud forming the Merged Point Cloud. Finally, the geometric transforms are needed to be able to have the data related to a common reference and therefore build a coherent map. The geometric transforms used are:

- base_sensing_kit → dalsa (multispectral camera).

Table 4.1: Precision of extracting the board's corner coordinates using the rviz tool. In order to check the accuracy and ease of selecting the desired points, three iterations were made where the coordinates of the four corners of the board (Top, Right, Bottom and Left) were extracted.

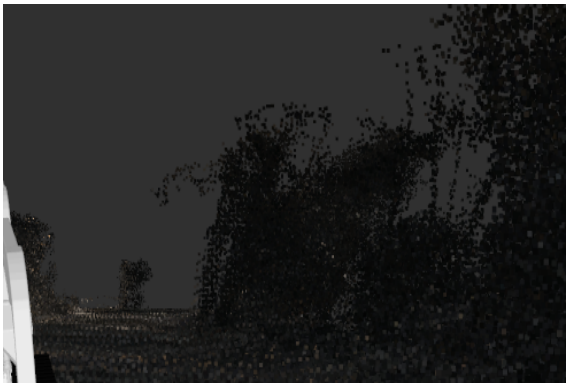
Point Position	Point Coordinates	Iteration 1	Iteration 2	Iteration 3	RMSE
Top	x	2.232	2.230	2.2347	0.0019
	y	0.2356	0.2347	0.2339	0.0006
	z	0.3530	0.3514	0.3561	0.0019
Right	x	2.4301	2.4334	2.4321	0.0013
	y	-0.10504	-0.1056	-0.10517	0.0002
	z	-0.1317	-0.1303	-0.1292	0.0010
Bottom	x	2.4713	2.4708	2.4717	0.0003
	y	0.1999	0.1967	0.1978	0.0013
	z	-0.3070	-0.3078	-0.3057	0.0008
Left	x	2.2915	2.2910	2.2831	0.0038
	y	0.5560	0.5532	0.5542	0.0011
	z	0.12297	0.12297	0.12334	0.0001

- base_sensing_kit → front_lslidar (3D LIDAR).
- bobcat_base → base_sensing_kit, so that the two sensors related to base_sensing_kit are related to bobcat_base.
- map → cartographer_odom.
- cartographer_odom → bobcat_base.
- bobcat_base → imu.

Briefly, we have 3D point cloud, camera images, odometry and imu all related to the same reference.

Some parameters had to be changed from the default values in order to integrate the colored point cloud. At first, `frame_id` has been changed from the default `base_link` to `bobcat_base` which is the frame attached to the base of our mobile robot system. Then, as we are going to feed the RTAB-Map with a point cloud, we are not going to provide either depth images or rgb images and so the parameters `subscribe_depth` and

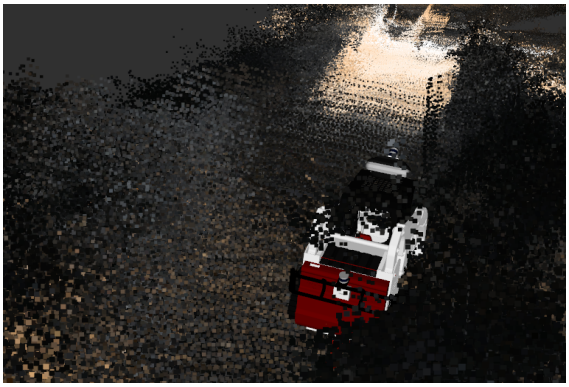
(a) Raw image from the dataset that highlights a set of trees (on the left).



(b) Set of trees in the map.

Figure 4.25: A set of trees that stand out from the rest of the elements around. The image 4.25b was taken in the opposite direction to get a better view. It is possible to observe the set of points that represent the tree highlighted in the image 4.25a.

(a) Raw image from the dataset that highlights a post and a set of plants.

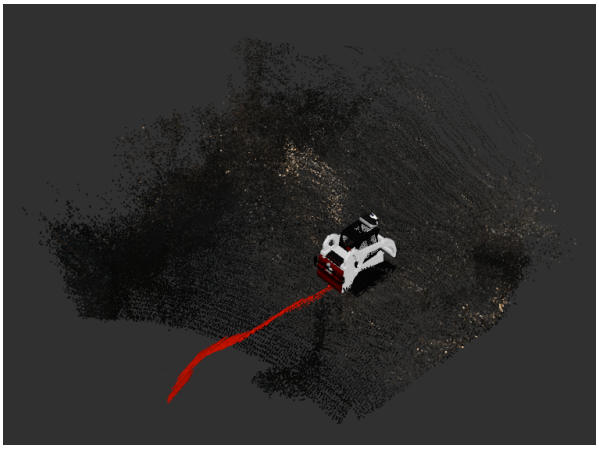


(b) Set of plants and a post in the map.

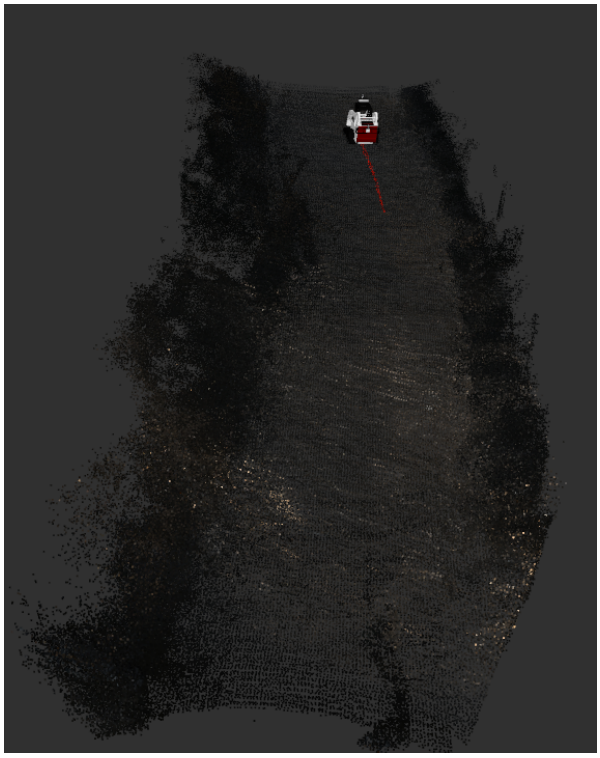
Figure 4.26: A set of plants (on the left) and a post (on the right) that stand out from the environment.

`subscribe_rgbd` are assigned as false. On the other hand, the `subscribe_scan_cloud` parameter is assigned as true as we are going to provide the point cloud mentioned earlier. The `rtabmap_ros/point_cloud_xyzrgb` node builds a point cloud with RGB through depth images or stereo images. However, as we use a monocular camera, we are no able to take advantage of this node and it is necessary to assign the point cloud, built with the Autoware.AI's

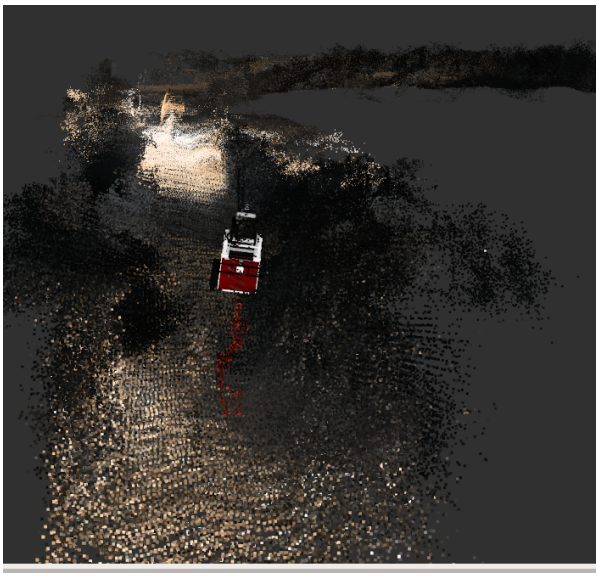
(a) Initial mapping phase.



(b) Top view of the initial mapping.



(c) Half-time map. With precise odometry and position estimation, the map starts to form coherently with the different point clouds.



(d) Close look to the final map. We can observe the path followed by the robot, and elements such as the car and people represented by the multispectral colored information.

Figure 4.27: The different mapping phases.

software, in other way. For this, we use the node `rtabmap_ros/point_cloud_assembler` which allows us to assign a point cloud to the output topic for `rtabmap_ros/point_cloud_xyzrgb` called `cloud` . Thus, the `cloud` topic is remapped to our colored point cloud topic, `points_fused`, completing the integration of our colored point cloud into RTAB-Map, making possible to map with it.

Before building the map, we expected the map to be dominated in areas with more light by yellowish and bluish / gray colors. In shady areas we expect a cluster of bluish / gray colors in places with vegetation but somewhat dominated by the black color due to the lack of light. The map at different times of the experiment can be seen in Figure 4.27. The results are acceptable because at first the compatibility of point clouds with image colors was fulfilled into the mapping process, the multispectral images provides useful information into the 3D map, allowing proper perception of the environment. The map is built in real time, being dense, consistent and detailed, enabling the distinction of elements and entities, such as cars, people, etc. The Figure 4.25 and Figure 4.26 illustrate some distinctive elements of the mapped environment while comparing with the captured image. The demonstration of the mapping process and the results obtained can be seen in the video[26].

[26] `https://www.youtube.com/watch?v=3LiQAMEM-8A&t=208s`

5 Conclusion

In this dissertation we propose to calibrate and register a set of sensors from a robotic system consisting of a stereo camera, a monocular camera and a 3D LIDAR (goal 1 in Chapter 1). After that, we proposed to build, in real time, a 3D map formed by different point clouds with usable information, which are created by using the registration outputs and the sensors mentioned, (goal 2). A manual calibration was performed between the sensors by using measuring tapes, in order to estimate with little precision and rigor the calibrations that we would calculate with a more accurate method.

We improved a method based on matching the same 3D points but in the different sensor frames. For that, we implemented the Kabsch algorithm for points obtained through a well-defined and strategic setups. During all the point extraction processes we used rviz, a ROS tool for data visualization and manipulation.

For stereo cameras, as we can obtain 3D information through stereo vision, we used software that forms 3D point clouds of the camera image. In this way, we are able to obtain and give our algorithm the 3D points.

On the other hand, by using monocular cameras, we are not able to obtain depth information. ArUco, a minimal library for Augmented Reality using specific markers, fulfills this role by allowing us to easily detect markers in a 2D image and determine its pose in relation to the camera. Knowing the camera-ArUco relationship, it is possible to know the coordinates of 3D points that are on the same plane as the marker.

Our calibration method got acceptable results, as we can see from 4.14 to 4.22. It is possible to see that the LIDAR data is correctly projected on the corresponding elements in the camera image, a fact that does not occur while using manual calibration. This results are only possible if the calibration and registration between the both sensors are correct.

Finally, in order to map a 3D environment the RTAB-Map method was used, taking advantage of its compatibility with point clouds. After merging data from a camera and a LIDAR into a point cloud, we successfully mapped a forest environment from which it is

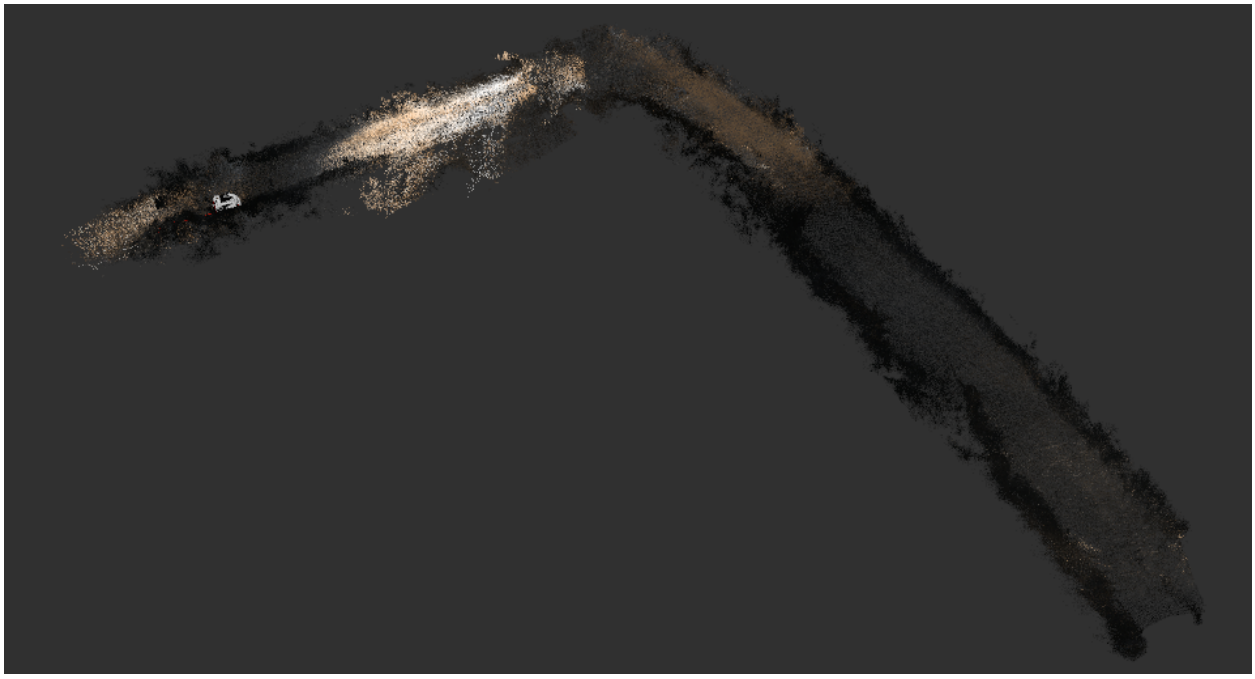


Figure 5.1: Top view of the final map.

possible to perceive useful multispectral information with our robotic system, as we can see in figure 4.27 and figure 5.1.

5.1 Future Work

LIDAR-LIDAR Registration

In this work we registered almost all sensors, lacking the LIDAR-LIDAR registration. We thought in a 3D LIDAR-LIDAR calibration method with which we do not need data intersection or to know corresponding points in both sensor frames. 3D LIDAR-LIDAR registration methods to date are developed through ICP processes with two point clouds with identical number of points and with a lot of intersection of common features. However, it is quite common for two LIDAR to be placed in robotic systems with opposite directions and orientations to be able to make a all around perception, so not sharing much of the same data.

We propose a calibration method based on the geometry of the space where the two LIDAR are inserted. Walls, ceilings and floors represent plans that can be well defined through some of the LIDAR points. The main idea is based on the equality of a same plane perpendicular vectors. For this idea to be true, all points must be defined in relation to the same origin in the world. In the end, if we know the equation of a plane seen by one LIDAR and the equation of the same plane seen by other LIDAR, we know two normal vectors of

the plane which must be equal to less than the point of application and vector dimension. Even though the points do not correspond for both sensors, the perpendicular vector formed are still equal and so it does not depend in the ability do match data. Therefore, as long as the sensor setup is in a place surrounded by well-defined plans, such as a garage or rooms, it is possible to relate the two sensors.

The mathematical development is described in appendix A.

MoDSeM Integration

In works like this one where cooperative perception is key, each of the members of the robot team contribute to the global knowledge of the system by sharing and cooperatively processing data and percepts from one another, combining the sensory abilities, perspectives and processing power of various agents to achieve better results. The Modular Framework for Distributed Semantic Mapping (MoDSeM) aggregates all this information to create a semantic map that can be shared among all system agents. It also aims to be reusable and adaptable to different sensing systems. MoDSeM consists of sensors that generate signals, Perception Modules (PM) that process these signals and, finally, the semantic map that represents a unique view of the world according to the information processed. Another important and very useful feature is the fact that PMs are only dependent on both sensors and the map, but not on other PMs. This allows the perception system to be adaptable to the computing power and the available resources. Any modules that are removed from the system do not affect it, so there is no need to redesign when changes are needed. PMs can use semantic maps, from others or from previous versions of the same technique, as input and thus be generally adaptable. However, while removing modules does not affect the overall system, errors can occur due to inter-dependencies information and so it is a process that needs some caution.

Integrating our entire method for an adaptable architecture such as MoDSeM is an interesting work to be carried out in the future, as it addresses the flexibility of the robotic system and promotes a better use and connection between the data of the different modules present in our dataflow.

Automating the process of finding matching points

Throughout our process of calibrating and registering cameras with LIDAR, we used techniques where corresponding points were chosen by hand, i.e. a user needs to select the desired

points. Although we verify that the existing tools for this type of processes are very precise, intuitive and concise, which does not demand extraordinary capacity from the user, there are already many edge and corner detection techniques which fits with the boards and markers used. A possible improvement is developing techniques that can automatically detect and obtain information about corresponding points in point clouds.

6 Bibliography

[1] J. Daley, "Study shows 84% of wildfires caused by humans." https://www.smithsonianmag.com/smart-news/study-shows-84-wildfires-caused-humans-180962315/.

[2] "Forest area (% of land area)." https://data.worldbank.org/indicator/AG.LND.FRST.ZS.

[3] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and vision computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[4] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[5] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6d slam—3d mapping outdoor environments," *Journal of Field Robotics*, vol. 24, no. 8-9, pp. 699–722, 2007.

[6] M. Pierzchala, "Greatest challenges for forest robots." https://www.forest-monitor.com/en/greatest-challenges-forest-robots/.

[7] G. S. Martins, J. F. Ferreira, D. Portugal, and M. S. Couceiro, "Modsem: modular framework for distributed semantic mapping," in *The 2nd UK-RAS Conference for PhD Students and Early-Career Researchers on Embedded Intelligence*, 2019.

[8] L. G. Brown, "A survey of image registration techniques," *ACM computing surveys (CSUR)*, vol. 24, no. 4, pp. 325–376, 1992.

[9] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611, pp. 586–606, International Society for Optics and Photonics, 1992.

[10] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

[11] E. W. Weisstein, "Affine transformation," *https://mathworld. wolfram. com/*, 2004.

[12] M. Pollefeys, R. Koch, and L. Van Gool, "Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters," *International Journal of Computer Vision*, vol. 32, no. 1, pp. 7–25, 1999.

[13] J. Wang, F. Shi, J. Zhang, and Y. Liu, "A new calibration model of camera lens distortion," *Pattern recognition*, vol. 41, no. 2, pp. 607–615, 2008.

[14] R. Hamad, S. A. Sattar, and R. Al-Azawi, "Calculating the inverse radial distortion model based on zhang method," *Advances in Natural and Applied Sciences*, vol. 11, no. 3, pp. 86–91, 2017.

[15] "What is camera calibration?." `https://www.mathworks.com/help/vision/ug/camera-calibration.html`.

[16] "Camera calibration." `https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html`.

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

[18] J. Feldmar, N. Ayache, and F. Betting, "3d–2d projective registration of free-form curves and surfaces," *Computer vision and image understanding*, vol. 65, no. 3, pp. 403–424, 1997.

[19] A. Dhall, K. Chelani, V. Radhakrishnan, and K. M. Krishna, "LiDAR-Camera Calibration using 3D-3D Point correspondences," *ArXiv e-prints*, May 2017.

[20] A. Sarkar, R. J. Santiago, R. Smith, and A. Kassaee, "Comparison of manual vs. automated multimodality (ct-mri) image registration for brain tumors," *Medical Dosimetry*, vol. 30, no. 1, pp. 20–24, 2005.

[21] G. Tagliabue, A. Maghini, S. Fabiano, A. Tittarelli, E. Frassoldi, E. Costa, S. Nobile, T. Codazzi, P. Crosignani, R. Tessandori, *et al.*, "Consistency and accuracy of

diagnostic cancer codes generated by automated registration: comparison with manual registration," *Population Health Metrics*, vol. 4, no. 1, p. 10, 2006.

[22] F. Pomerleau, F. Colas, and R. Siegwart, "A review of point cloud registration algorithms for mobile robotics," 2015.

[23] B. Bellekens, V. Spruyt, R. Berkvens, R. Penne, and M. Weyn, "A benchmark survey of rigid 3d point cloud registration algorithms," *Int. J. Adv. Intell. Syst*, vol. 8, pp. 118–127, 2015.

[24] R. S. Phogat, H. Dhamecha, M. Pandya, B. Chaudhary, and M. Potdar, "Different image registration methods—an overview," *Int J Sci Eng Res*, vol. 5, pp. 44–9, 2014.

[25] L. M. Fonseca and B. Manjunath, "Registration techniques for multisensor remotely sensed imagery," *PE & RS- Photogrammetric Engineering & Remote Sensing*, vol. 62, no. 9, pp. 1049–1056, 1996.

[26] X. Dai and S. Khorram, "A feature-based image registration algorithm using improved chain-code representation combined with invariant moments," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 37, no. 5, pp. 2351–2362, 1999.

[27] E. Mair, G. D. Hager, D. Burschka, M. Suppa, and G. Hirzinger, "Adaptive and generic corner detection based on the accelerated segment test," in *European conference on Computer vision*, pp. 183–196, Springer, 2010.

[28] T. Lindeberg, "Scale invariant feature transform," 2012.

[29] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, pp. 404–417, Springer, 2006.

[30] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[31] S. Mohammad and T. Morris, "Binary robust independent elementary feature features for texture segmentation," *Advanced Science Letters*, vol. 23, no. 6, pp. 5178–5182, 2017.

[32] E. Karami, S. Prasad, and M. Shehata, "Image matching using sift, surf, brief and orb: performance comparison for distorted images," *arXiv preprint arXiv:1710.02726*, 2017.

[33] D. Tyagi, "Introduction to feature detection and matching." `https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d`.

[34] "Feature detection (computer vision)." https://en.wikipedia.org/wiki/Feature_detection_(computer_vision).

[35] S. Bouaziz, A. Tagliasacchi, and M. Pauly, "Sparse iterative closest point," in *Computer graphics forum*, vol. 32, pp. 113–123, Wiley Online Library, 2013.

[36] Y. He, B. Liang, J. Yang, S. Li, and J. He, "An iterative closest points algorithm for registration of 3d laser scanner point clouds with geometric features," *Sensors*, vol. 17, no. 8, p. 1862, 2017.

[37] M. Ovsjanikov, "Rigid shape registration." http://www.lix.polytechnique.fr/~maks/Verona_MPAM/TD/TD1/.

[38] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[39] D. Kortenkamp and T. Weymouth, "Topological mapping for mobile robots using a combination of sonar and vision sensing," in *AAAI*, vol. 94, pp. 979–984, 1994.

[40] G. d. S. Martins, "A cooperative slam framework with efficient information sharing over mobile ad hoc networks," Master's thesis, 2014.

[41] R. Chellali, K. Baizid, and Z. Li, "2d and 3d virtual environment for human robot interaction: from virtual perception to real localization," in *2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 694–699, IEEE, 2009.

[42] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[43] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part ii," *IEEE robotics & automation magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[44] M. J. Milford, "Robotic mapping methods," in *Robot Navigation from Nature*, pp. 15–28, Springer, 2008.

[45] V. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello, "Bayesian filtering for location estimation," *IEEE pervasive computing*, vol. 2, no. 3, pp. 24–33, 2003.

[46] R. Sim, P. Elinas, M. Griffin, J. J. Little, *et al.*, "Vision-based slam using the rao-blackwellised particle filter," in *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, vol. 14, pp. 9–16, 2005.

[47] J. Nieto, T. Bailey, and E. Nebot, "Scan-slam: Combining ekf-slam and scan correlation," in *Field and service robotics*, pp. 167–178, Springer, 2006.

[48] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the ekf-slam algorithm," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3562–3568, IEEE, 2006.

[49] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings of the 2005 IEEE international conference on robotics and automation*, pp. 2432–2437, IEEE, 2005.

[50] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[51] R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard, "Large scale graph-based slam using aerial images as prior information," *Autonomous Robots*, vol. 30, no. 1, pp. 25–39, 2011.

[52] M. Labbé and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, 2013.

[53] M. Labbé and F. Michaud, "Memory management for real-time appearance-based loop closure detection," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1271–1276, 2011.

[54] M. Labbé and F. Michaud, "Online global loop closure detection for large-scale multi-session graph-based slam," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2661–2666, 2014.

[55] M. Labbé and F. Michaud, "Long-term online multi-session graph-based splam with memory management," *Autonomous Robots*, vol. 42, no. 6, pp. 1133–1150, 2018.

[56] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the rgb-d slam system," in *2012 IEEE International Conference on Robotics and Automation*, pp. 1691–1696, IEEE, 2012.

[57] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time.," in *Robotics: Science and Systems*, vol. 2, 2014.

[58] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4758–4765, IEEE, 2018.

[59] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: Low-drift, robust, and fast," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2174–2181, IEEE, 2015.

[60] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4758–4765, IEEE, 2018.

[61] M. Quigley, K. Conley, B. 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*, vol. 3, p. 5, Kobe, Japan, 2009.

[62] C. B. Duane, "Close-range camera calibration," *Photogramm. Eng*, vol. 37, no. 8, pp. 855–866, 1971.

[63] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[64] J. Weng, P. Cohen, M. Herniou, *et al.*, "Camera calibration with distortion models and accuracy evaluation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 14, no. 10, pp. 965–980, 1992.

[65] F. Remondino and C. Fraser, "Digital camera calibration methods: considerations and comparisons," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 36, no. 5, pp. 266–272, 2006.

[66] C. S. Fraser, "Automatic camera calibration in close range photogrammetry," *Photogrammetric Engineering & Remote Sensing*, vol. 79, no. 4, pp. 381–388, 2013.

[67] O. D. Faugeras, Q.-T. Luong, and S. J. Maybank, "Camera self-calibration: Theory and experiments," in *European conference on computer vision*, pp. 321–334, Springer, 1992.

[68] R. I. Hartley, "Self-calibration of stationary cameras," *International journal of computer vision*, vol. 22, no. 1, pp. 5–23, 1997.

[69] O. Sorkine, "Least-squares rigid motion using svd," *Technical notes*, vol. 120, no. 3, p. 52, 2009.

[70] "Aruco: a minimal library for augmented reality applications based on opencv." `https://www.uco.es/investiga/grupos/ava/node/26`.

[71] "Detection of aruco markers." `https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html`.

[72] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in neural information processing systems*, pp. 2366–2374, 2014.

[73] D. Anton Mikhailov, Senior Software Engineer, "Turbo, an improved rainbow colormap for visualization." `https://ai.googleblog.com/2019/08/turbo-improved-rainbow-colormap-for.html`.

[74] M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[75] A. González, D. Vázquez, A. M. López, and J. Amores, "On-board object detection: Multicue, multimodal, and multiview random forest of local experts," *IEEE transactions on cybernetics*, vol. 47, no. 11, pp. 3980–3990, 2016.

[76] I. Z. Ibragimov and I. M. Afanasyev, "Comparison of ros-based visual slam methods in homogeneous indoor environment," in *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, pp. 1–6, IEEE, 2017.

[77] N. Altuntaş, E. Uslu, F. Çakmak, M. F. Amasyalı, and S. Yavuz, "Comparison of 3-dimensional slam systems: Rtab-map vs. kintinuous," in *2017 International Conference on Computer Science and Engineering (UBMK)*, pp. 99–103, IEEE, 2017.

[78] M. A. Post, A. Bianco, and X. T. Yan, "Autonomous navigation with ros for a mobile robot in agricultural fields," in *14th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2017.

Appendix A

3D LIDAR-LIDAR Calibration

1 3D LIDAR-LIDAR calibration method

With three points we were able to define the equation of a plane. We managed to know the coordinates of the points in relation to each LIDAR. Therefore, by selecting three points that we know are part of the plane of the wall, floor or ceiling, we can define the equation of these plans.

The main idea of our method for calibration is based on the equality of the normal vectors of the equations that define the same plane. For this idea to be true, all points must be defined in relation to the same origin in the world. In the end, if we know the equation of a plane seen by one LIDAR and the equation of the same plane seen other LIDAR, we know the two normal vectors of that plane, which must be equal to less than the point of application and of vector dimension.

$$V_W = V_L = F({}^{L}T_W P_L) \tag{1.1}$$

Where,

- V_W is the vector perpendicular to the plane which is defined with three points in relation to the world origin.
- V_L is the perpendicular vector of the same plane but formed by the points obtained in the LIDAR frame and later transformed into the world frame.
- ${}^{T}T_W$ the transformation matrix that links the LIDAR to the world frame, i.e. it transforms the points obtained in the LIDAR frame to the world frame.
- F is the function that calculates the perpendicular vector knowing three points.
- X_L are three points of the plan in relation to the origin of LIDAR.

Through this equality we can define a transformation matrix which we do not know any value. We know how to define V_W, we know the points P_L and we know the function F to form the normal vector with these points. So to generalize our problem we will use X_n, Y_n

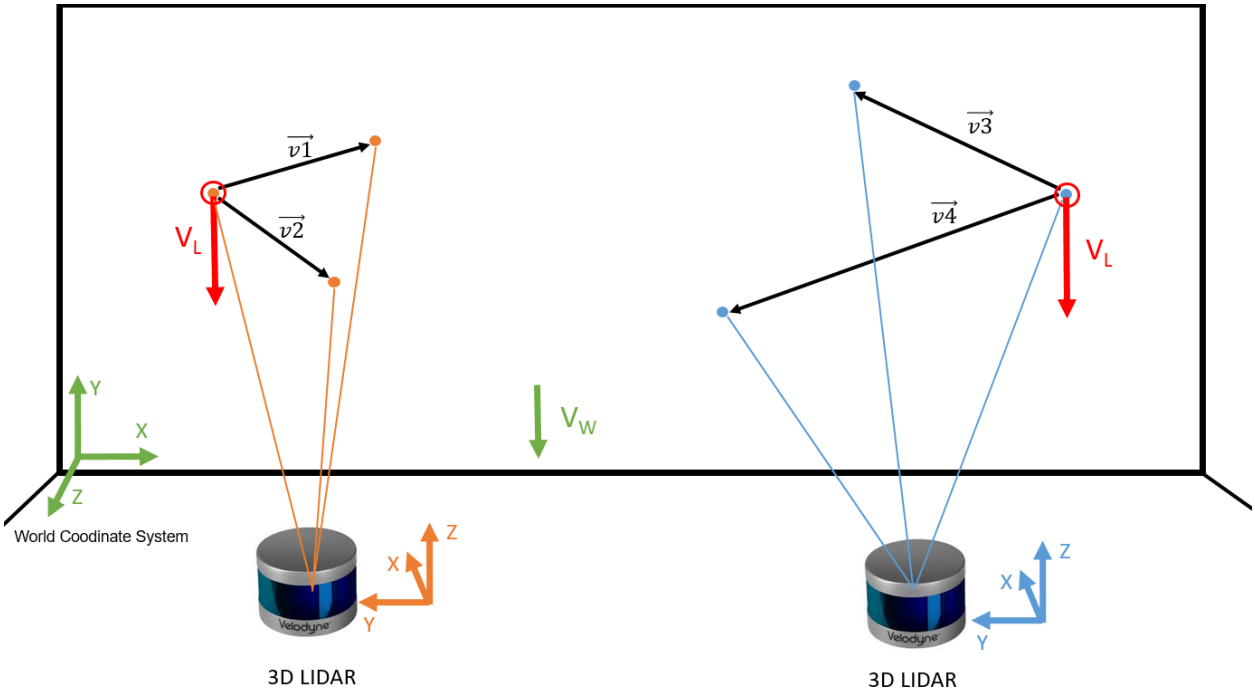


Figure 1.1: Representation of a plane seen by two LIDAR, the coordinate systems and all the vectors mentioned.

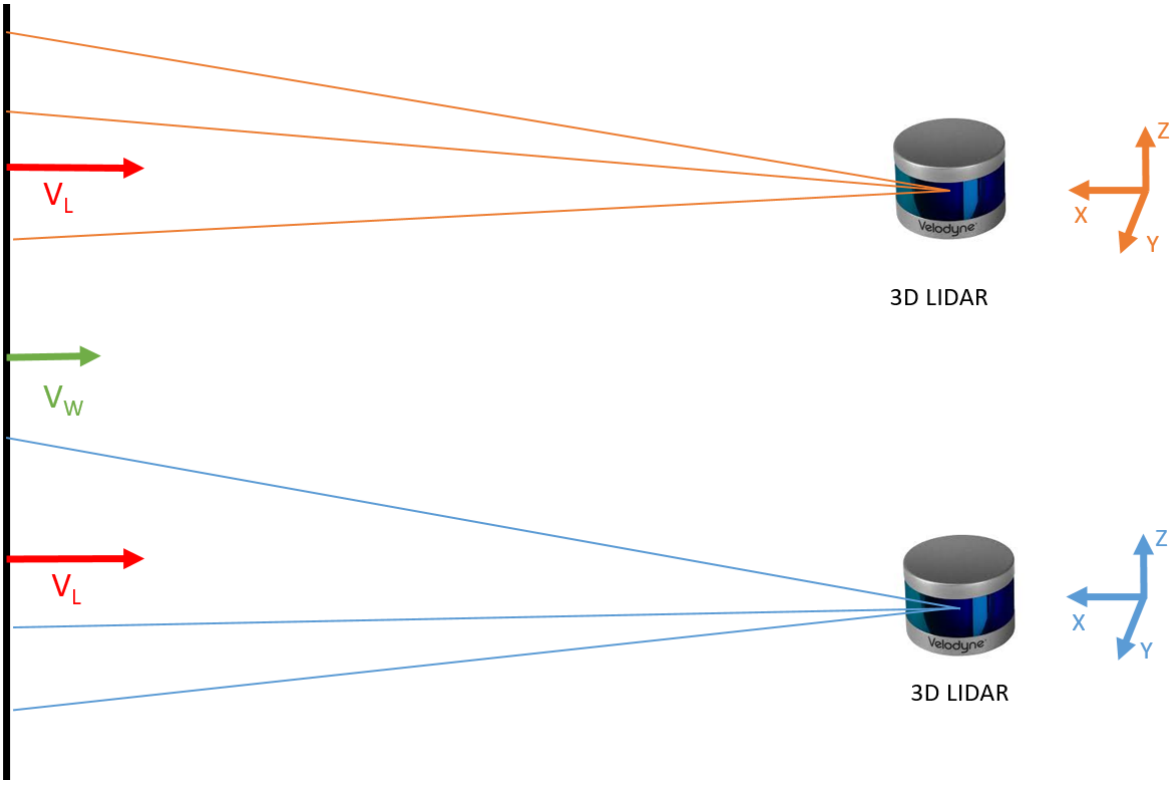


Figure 1.2: Side view of the setup to view the perpendicular vectors formed by the points of the plane.

and Z_n to represent the coordinates of the points and T_{lc} the indices of the rows and columns of the transformation matrix.

Starting with the matrices in parentheses:

$$P_{W_n} = \begin{bmatrix} T_{11} & T_{12} & T_{13} & T_{14} \\ T_{21} & T_{22} & T_{23} & T_{24} \\ T_{31} & T_{32} & T_{33} & T_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \\ Z_n \\ 1 \end{bmatrix} \tag{1.2}$$

P_W are the LIDAR points in the world framework. We will represent the X, Y and Z coordinates of the point n (n=3) by P_{x_n}, P_{y_n} and P_{z_n} respectively.

$$\begin{aligned} T_{11}X_n + T_{12}Y_n + T_{13}Z_n + T_{14} &= P_{x_n} \\ T_{21}X_n + T_{22}Y_n + T_{23}Z_n + T_{24} &= P_{y_n} \\ T_{31}X_n + T_{32}Y_n + T_{33}Z_n + T_{34} &= P_{z_n} \end{aligned} \tag{1.3}$$

At the end of this process ,we have the points in the world reference. We will define it as P_{W_1}, P_{W_2} and P_{W_3}. We now apply the F function to create the normal vector. This function encompasses two processes: creation of two vectors and vector cross product.

To create the two vectors ($\vec{v}_1$ and $\vec{v}_2$) we make the difference between two points: $P_{W_2} - P_{W_1}$ and $P_{W_3} - P_{W_1}$:

$$\begin{aligned} \vec{v}_1\hat{x} &= (T_{11}X_2 + T_{12}Y_2 + T_{13}Z_2 + T_{14}) - (T_{11}X_1 + T_{12}Y_1 + T_{13}Z_1 + T_{14}) \\ &= T_{11}(X_2 - X_1) + T_{12}(Y_2 - Y_1) + T_{13}(Z_2 - Z_1) \\ \vec{v}_1\hat{y} &= T_{21}(X_2 - X_1) + T_{22}(Y_2 - Y_1) + T_{23}(Z_2 - Z_1) \\ \vec{v}_1\hat{z} &= T_{31}(X_2 - X_1) + T_{32}(Y_2 - Y_1) + T_{33}(Z_2 - Z_1) \end{aligned} \tag{1.4}$$

And for the second vector:

$$\begin{aligned} \vec{v}_2\hat{x} &= (T_{11}X_3 + T_{12}Y_3 + T_{13}Z_3 + T_{14}) - (T_{11}X_1 + T_{12}Y_1 + T_{13}Z_1 + T_{14}) \\ &= T_{11}(X_3 - X_1) + T_{12}(Y_3 - Y_1) + T_{13}(Z_3 - Z_1) \\ \vec{v}_2\hat{y} &= T_{21}(X_3 - X_1) + T_{22}(Y_3 - Y_1) + T_{23}(Z_3 - Z_1) \\ \vec{v}_2\hat{z} &= T_{31}(X_3 - X_1) + T_{32}(Y_3 - Y_1) + T_{33}(Z_3 - Z_1) \end{aligned} \tag{1.5}$$

Finally, when making the vector product of $\vec{v}_1$ and $\vec{v}_2$ we have the normal vector V_L:

$$\begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ T_{11}(X_2 - X_1) + T_{12}(Y_2 - Y_1) & T_{21}(X_2 - X_1) + T_{22}(Y_2 - Y_1) & T_{31}(X_2 - X_1) + T_{32}(Y_2 - Y_1) \\ +T_{13}(Z_2 - Z_1) & +T_{23}(Z_2 - Z_1) & +T_{33}(Z_2 - Z_1) \\ T_{11}(X_3 - X_1) + T_{12}(Y_3 - Y_1) & T_{21}(X_3 - X_1) + T_{22}(Y_3 - Y_1) & T_{31}(X_3 - X_1) + T_{32}(Y_3 - Y_1) \\ +T_{13}(Z_3 - Z_1) & +T_{23}(Z_3 - Z_1) & +T_{33}(Z_3 - Z_1) \end{bmatrix} \tag{1.6}$$

To facilitate the visualization of the problem we do:

$$\begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ A & B & C \\ D & E & F \end{bmatrix} \tag{1.7}$$

Getting the cross product in the form:

$$V_L = (BF - CE)\hat{i} - (AF - CD)\hat{j} + (AE - BD)\hat{k} \tag{1.8}$$

We know all the differences between X_n, Y_n and Z_n, we just don't know T_{lc}:

$$\begin{aligned} (BF - CE) = &\Big[\big[(X_2 - X_1)T_{21} + (Y_2 - Y_1)T_{22} + (Z_2 - Z_1)T_{23})\big] \cdot \big[(X_3 - X_1)T_{31} + (Y_3 - Y_1)T_{32} \\ &+ (Z_3 - Z_1)T_{33})\big]\Big] - \Big[\big[(X_2 - X_1)T_{31} + (Y_2 - Y_1)T_{32} + (Z_2 - Z_1)T_{33})\big] \cdot \big[(X_3 - X_1)T_{21} \\ &+ (Y_3 - Y_1)T_{22} + (Z_3 - Z_1)T_{23})\big]\Big] \end{aligned} \tag{1.9}$$

$$\begin{aligned} (AF - CD) = &\Big[\big[(X_2 - X_1)T_{11} + (Y_2 - Y_1)T_{12} + (Z_2 - Z_1)T_{13})\big] \cdot \big[(X_3 - X_1)T_{31} + (Y_3 - Y_1)T_{32} \\ &+ (Z_3 - Z_1)T_{33})\big]\Big] - \Big[\big[(X_2 - X_1)T_{31} + (Y_2 - Y_1)T_{32} + (Z_2 - Z_1)T_{33})\big] \cdot \big[(X_3 - X_1)T_{11} \\ &+ (Y_3 - Y_1)T_{12} + (Z_3 - Z_1)T_{13})\big]\Big] \end{aligned} \tag{1.10}$$

$$(AE - BD) = \Big[\big[(X_2 - X_1)T_{11} + (Y_2 - Y_1)T_{12} + (Z_2 - Z_1)T_{13})\big] \cdot \big[(X_3 - X_1)T_{21} + (Y_3 - Y_1)T_{22} + (Z_3 - Z_1)T_{23})\big]\Big] - \Big[\big[(X_2 - X_1)T_{21} + (Y_2 - Y_1)T_{22} + (Z_2 - Z_1)T_{23})\big] \cdot \big[(X_3 - X_1)T_{11} + (Y_3 - Y_1)T_{12} + (Z_3 - Z_1)T_{13})\big]\Big] \tag{1.11}$$

As we can see from the development of the equations, the translation parameters were cut, being able to determine the 3x3 rotation matrix only. Thus, as we have nine values to find, we need nine equations.

The remaining equations are obtained by adding two more points, for a total of five, to all the development shown above in order to obtain three perpendicular vectors and nine equations. With everything prepared, we need to solve the system of non-linear equations, obtaining the rotation matrix between each LIDAR to the world. Knowing the relationship of the two LIDAR to the same point in the world, we consequently know the relationship between the two sensors.