Contents

1 Introduction **7**
- 1.1 Motivation . . . 7
- 1.2 Problem Description . . . 7
- 1.3 Objectives . . . 8
- 1.4 Document Outline . . . 8

2 State of Art **9**
- 2.1 Technologies . . . 9
  - 2.1.1 Stereoscopy . . . 9
  - 2.1.2 Structured Light . . . 9
  - 2.1.3 Time of Flight . . . 10
  - 2.1.4 LiDAR . . . 10
- 2.2 Related Academic Work . . . 14
- 2.3 Comercial Solutions . . . 16
  - 2.3.1 Matterport . . . 16
  - 2.3.2 Faro Focus . . . 17
- 2.4 3D Digital Models . . . 17

3 Experimental Infrastructure **21**
- 3.1 Hardware . . . 21
- 3.2 Software . . . 23
  - 3.2.1 Robot Operating System . . . 24
  - 3.2.2 Processing Application . . . 27

4 Methodology for Scene Capture **31**
- 4.1 Acquisition . . . 31
  - 4.1.1 Movement Programming . . . 32
  - 4.1.2 Parameterization Considerations . . . 32
  - 4.1.3 Acquisition node . . . 33
  - 4.1.4 Data Serialization . . . 34
- 4.2 Capture . . . 35

5 Methodology for Geometry Reconstruction **39**
- 5.1 Point Registration . . . 39
- 5.2 Laser Extrinsic Calibration . . . 40
  - 5.2.1 Radlocc Method . . . 40
  - 5.2.2 Proposed Method . . . 42

5.3 Normal Estimation 45
5.4 Acquisition Registration 47
5.4.1 Iterative Closest Point 47
5.4.2 Multiple Point Cloud ICP 48
5.5 Filters 49
5.5.1 NaN Removal 49
5.5.2 Statistic Outlier Removal 49
5.5.3 Voxel Grid Downsampling 50

6 Methodology for Image Registration **51**
6.1 Color Registration 51
6.1.1 Point to pixel coordinates transformation 51
6.1.2 Camera Distortion 52
6.1.3 Point filtering 53
6.1.4 Color Attribution 55
6.2 Color Fusion 55
6.3 Camera Intrinsic Calibration 57
6.4 Camera Extrinsic Calibration 58

List of Figures

2.1 Point cloud of Retz obtained by an airborne LiDAR. 12
2.2 Sick LMS511 2D laser scanner. 13
2.3 Velodyne VLS128 LiDAR laser scanner. 13
2.4 Google Autonomous Car. 14
2.5 3D laser scanner developed in [16]. 15
2.6 The KaRoLa 3D laser scanner. 15
2.7 Matterport Pro2 Camera. 16
2.8 Matterport "Pennsylvania Craftsman Home" model. 17
2.9 Faro Focus 3D laser scanner. 18
2.10 Faro 3D scan. 18

3.1 Lemonbot mobile 3D scanner 22
3.2 Laser scanners used. 23
3.3 PointGrey Flea3 FL3-GE-28S4 24
3.4 FLIR PTU-D46 25
3.5 ROS architecture overview 26
3.6 Cloud Compare screenshot. 29

4.1 Limitations of a single acquisition 32
4.2 Waypoints and movements in the pan/tilt joint space 33
4.3 Example of a recorded bag file info 34
4.4 Example of laser scan row 35
4.5 Example of the parameters YAML file 36

5.1 Transformation graph 39
5.2 Images captured for RADLOCC 41
5.3 Radlocc laser scans chessboard extraction 41
5.4 Example of a plane segmentation, where each color represents a cluster 43
5.5 Calibration Overview 45
5.6 Stanford rabbit [31] rendering with (left) and without (right) normals 46
5.7 Multiple Point Cloud ICP approaches 49
5.8 SOR filter in a point cloud 50
5.9 Stanford Lucy scan [28] after a voxel grid downsampling with different leaf sizes 50

6.1 Color registration for a single point 52
6.2 Barrel distortion in fish eye lens 53
6.3 Representation of the visual frustum of the camera 54
6.4 Result of the HPR operator in the Bunny point cloud 55

6.5 Bilinear interpolation in an image 56
6.6 Interface for the *cameracalibrator* node 58
6.7 Hand-in-eye transformation graph 59
6.8 ArUco maker detection and pose estimation. 60

List of Tables

3.1 Comparison of the three laser scanners used. 23
3.2 Characteristics of the PointGrey Flea3 FL3-GE-28S4 Camera 24
3.3 FLIR PTU-D46 characteristics. 25

Chapter 1

Introduction

Digital reconstruction of three dimensional scenes is a field that gained a high importance in areas like architecture, robotics, archeology and autonomous driving. As an example, virtual reality technologies allows high-detail and high-resolution models to be experiences in a immersive 3D experience, and the technology is becoming available for the general public, as prices for this 3D headsets and VR-ready phones decrease. This new technologies create a demand for reconstruction technology and new algorithms that are accurate and of high quality.

1.1 Motivation

Despite all the work done, there is still no perfect solution. 3D reconstruction is still a challenge, because real scenes are very complex and measurements are subjected to errors and noise. Past experience tells that a single sensor is not enough to model real environments, so currently the process lies into using multiple sensors and trying to merge the data from all the sensors, in order to capture a more realistic model.

However, this introduces a set of other problems that are inherent to this approach. For example, the data from the different sensors need to be merged accurately. For example, the positions of all the sensors need to be known accurately, which means that the calibration processes need to be more robust and precise.

Also, current reconstruction algorithms require a large amount of manual work, which means that a reconstruction require many man-hours to be processed, which is unfeasible for most applications. An automatized reconstruction method is still a challenge today, but new algorithms could allow it, which would make reconstruction work more accessible.

Nowadays, 3D reconstruction software is even available in smartphones, usually targeted for Augmented Reality applications. However, the 3D reconstructions are not accurate and the resulting models are far from perfect.

1.2 Problem Description

Recently, Lidar laser scanners became more available and, because of their properties, as their high precision and high range, became an unmatched technology for 3D reconstruction. The lidar lasers are available as 2D laser scanners or 3D laser scanners, like the Velodyne. Despite their immense potential, 3D laser scanner is still a very expensive solution and cheaper

solutions are comprised of a cheaper 2D laser scanner mounted on a moving frame. This solution, despite its low cost, can achieve good results, but requires a fine calibration.

Also, laser scanner do not register the color information, so a common practice is to pair the laser scanner with a camera to get both color and geometric data. This method also requires a fine calibration between both sensors to merge the data from both sensors.

1.3 Objectives

The objective is to develop a fully integrated solution for 3D acquisition using both laser and image data. This objective was divided into four main objectives.

The first objective was to develop a mobile 3D scanner, consisted of a laser scanner and a camera, and capable of recording data from both sensors in a fast and semi-autonomous way.

The second objective was to define a methodology to record the data from the scene. This methodology should take into account the limitations of both sensors and try to minimize their effect in the final result.

The third objective was to develop a set of methods to reconstruct the geometry of the scene reliably. The main challenge is the extrinsic calibration of the laser, because it is fundamental for a reliable reconstruction. So, a new calibration method was developed to achieve the wanted results.

The fourth and final objective was to develop a set of methods to merge the image data with the geometry of the scene, to reconstruct the color.

The final result is then, a point cloud with color and geometric information.

1.4 Document Outline

This dissertation is composed of eight chapters, which are arranged as follows:

Introduction The current chapter, in which the description of the problem is shown and the objectives of this work are defined.

State of Art Describes the technologies and solutions found in the field of 3D reconstruction, both commercial and academic, as well of a small technical background on this solutions.

Experimental Infraestructure Introduces all the software and hardware used to develop this work. In particular, the mobile robot for 3D scanner is described.

Methodology for Scene Capture Describes the methods and algorithms used to reconstruct the geometry of the scene, using the laser scan data recorded.

Methodology for Image Reconstruction Describes the methods and algorithms used to reconstruct the color information of the scene, using the camera images recorded.

Result and Discussion Presents and discusses the experimental results obtained in this work.

Conclusion and Future Work Summarizes the overall work developed and present possible future work.

Chapter 2

State of Art

2.1 Technologies

Many technologies were developed to capture tridimensional information of the environment. The following section describes such systems and describe the basic working principle along with the pros and cons inherent to each ones. These techniques can be categorized into triangulation and time-of-flight.

2.1.1 Stereoscopy

For many years, stereoscopy remained the most popular method for 3D sensing, also because its working principle resembles our stereoscopic vision. This system uses images taken from a pair of cameras and extract the depth information due to the perspective projection: the position of objects closer differ more than objects farther. To compute depth, features from both images are extracted and correspond together, which makes it a complex and computationally demanding, so it requires fast computers or dedicated software. This system has the advantage of having a good rate of acquisition and having high resolution. Also, color information is available. However, the reconstruction algorithm rely heavy on environment characteristics, like lightning conditions, texture and non-homogeneous regions[11]. This means that this method gives good results for edges and textured areas, but fails to get the depth information of continuous surfaces.

2.1.2 Structured Light

In 2010, the availability of consumer grade depth sensors based on structured light lead to the development of consumer-grade small factor RGB-D cameras, started by Microsoft, with the *Kinect* and followed by other devices, like *ASUS Xtion* and *Intel RealSense*. This cameras come in small form factors, are inexpensive and are capable of capturing both color and depth information at real-time rates[36].

This appealing characteristics lead to a huge research and development in 3D reconstruction using this camera, culminating in the KinectFusion algorithm [18], capable of a fast and precise 3D reconstruction using a *Kinect* RGB-D camera and commodity GPUs. This algorithm was capable of real-time reconstruction, using a Iterative Closest Point (ICP) for tracking the location of the device and for the registration of new RGB-D data. Nowadays,

it is possible to achieve the same result using a phone equipped with a depth camera, like the *Lenovo Phab 2* and with the *Google Tango* software.

Structured light sensors work by projecting an infrared pattern onto the scene and calculate the depth via the perspective deformation of the pattern due to the different object's depth. However, this technique yields sub-optimal results: the depth values from structured light have significant error or can be missing, specially from objects with darker colors, specular surfaces or small surfaces[25].

2.1.3 Time of Flight

Time of flight sensors, or ToF sensors, use the speed of light to measure the distance. A scene is illuminated by a light source and the reflected light is detected back by the sensor. The time that the light has taken to travel back and forth is then measured and the depth is calculated with this time. The measurement dependes on the type of ToF system used, that is ether *continuous* or *pulsed*. In *pulsed* systems, light is emitted in bursts with a fast shutter and the time between the emission and the reception is calculated. *Continuous* systems use a modulated light source and measure the phase-shift between the outgoing and incoming wave.

This technology has some advantages comparing to both previous approaches [36]. First, it is less computationally intensive, because the measurement is directly measured by a specialized sensor. Second, it is partially independent of the lightning conditions because the light detected is emitted by the device itself. Further, it is capable of a dense and accurate depth values, even for continuous or irregular surfaces, unlike the stereoscopic approach. Moreover, it is much faster that any other method, capable of acquisition rates of hundreds of Hz.

However, it has some disadvantages as well. Unlike stereoscopic, which is a passive method, ToF sensors interact with the scene, so are not possible to be implemented in certain environments. Also, the properties of the material, like the reflectivity, color and roughness can have significant effects on the accuracy of ToF sensors. Moreover, multi-path reflections are a common problem of ToF sensors, caused by multiple reflections of the light, causing errors in the measurements. Furthermore, interference can exist if multiple ToF sensors share the same environment. However, it is possible to mitigate this effect.

A popular ToF sensor nowadays is the Photonic-Mixer-Device camera, which looks similarly to a normal image sensor, but measures the phase shift of incoming light. This sensor is now being used in new generation RGB-D cameras, replacing the structured light approach, mainly because it is more resilient to background light, allowing the sensor to work in outdoor environments [36]. One of this example is the new *Kinetic 2*, that replaced the last depth sensor with this one.

2.1.4 LiDAR

Light Detection and Ranging, or LiDAR, is one of the most precise and reliable ways to measure distances. It began being used shortly after the Laser invention, in 1960, and it's valuable characteristics lead to the integration of it in the Apollo 15 mission, to serve as an altimeter to map the surface of the moon. Soon after, it was implemented in aircrafts to create high-precision and dense earth's surface models. Nowadays, its applications can

be found everywhere where an accurate distance measurement is required, as for example in geology, archeology, geography, oceanography and meteorology.

LiDAR success is related to the use of laser as its light source. Lasers are capable of emitting beams of light that are monochromatic, narrow and polarized. That is, lasers emit light in a narrow spectrum of light, so they can produce a single color of light, also known as monochromatic light. It improves the resilience against background light, making it possible to use even with sun light. Also, laser photons travel parallel, creating a narrow beam that stays narrow even at large distances, with minimum scattering, therefore measuring the distance in a very small area in the surface. This improves the measurements near sharp transitions, where a bigger area of measurement can cause errors in the measurement. Moreover, lasers can transition between an on-off state in a very short time. This is significant to reduce the error of the distance measurement, because it is directly influenced by the time between pulses, and a sharp transition reduces the error in the time measurement.

The first major application of LiDAR technology was to map and reconstruct the topology of the earth surface. To do it, high range laser scanners are mounted on a plane and are flown above the target area. This is possible due to their high power laser, which allows ranges in the km range, whiles maintaining precision in the cm range. This lasers also have a high sampling rate, in the order of the kHz, which is essencial to maintain a dense sampling, even at high travelling speeds. This high sampling also allows the terrain to be mapped even with high vegetation, because even that only a small percentage of points reach the terrain, there are still a significant number of points. This is just not possible with other methods, like aerial photography.

Moreover, airborne LiDAR is also used to detect and classify clouds, which is an important data in meteorology. This is possible by studying the rayleigh scattering effect, which occurs when a laser beam goes through a particles smaller that its wave length, as for example, the water molecules in clouds. This is essential for modern forecast prediction.

A airborne LiDAR scanner of the Austrian city Retz is shown in Figure 2.1, which is available in [20], taken with a Riegl laser scanner placed in a Unmanned Aerial Vehicle. Figure 2.1a shows the entire scan with an area of $1300\,\mathrm{m} \times 1300\,\mathrm{m}$, Figure 2.1b shows a portion correspondent to the city, and Figure 2.1c shows the town hall, which has an area of about $150\,\mathrm{m} \times 70\,\mathrm{m}$. As can be seen, the result is a massive point cloud that covers a large area while still maintaining a high point density so small details are not smeared out.

In recent years, LiDAR scanner became a fundamental technology for industrial and robotics applications. Their small form factor and high precision are essencial for numerous applications. In general, two types of laser scanners exist: the 2D laser scanners and the 3D laser scanners.

2D laser scanners emit a single laser beam, which is reflected by a rotating mirror to scan across a planar area, as seen on figure X. They are also the most accessible type, as their price ranges range from 800€up to 20 000€, depending on their characteristics. One example of this laser scanners is the SICK LMS511, shown in Figure 2.2.

This laser scanners have a large number of applications. For example, 2D laser scanners are used in autonomous robots, to provide precise 2D mapping information of the environment, which can be user afterwards for location and navigation [26]. Compared to other technologies, like stereo vision, this one requires small processing power and yields accurate results, so their application is easy to implement and requires low processing power.

Another widely used application is intrusion detection. The 2D laser scanner can be spaced in a room or door to detect if any object enters to the space. For example, it is used

(a) The entire scan.

(b) Pormenor of the City.

(c) Pormenor of the Town Hall.

Figure 2.1: Point cloud of Retz obtained by an airborne LiDAR.

Figure 2.2: Sick LMS511 2D laser scanner.

to ensure the safety of workers in industrial environment, ensuring that workers do not get close to working machines. Other example is in theft prevention in museums and banks to secure specific areas against robbery or vandalism [1].

Recently, 3D laser scanners become available, but at a high price range. One example of this new sensors is the Velodyne, which is a 3D laser scanners priced at 80 000€. This laser scanners are capable of producing a 3D point cloud directly, unlike the 2D laser scanner. This is possible because they emit multiple laser beams, instead of just one. The laser beams are reflected using a rotating mirror, to get a continuous 3D scan. Moreover, 3D laser scanners are capable of producing a 3D scan at a higher rate than any solution employing a 2D laser scanner. In the case of Velodyne *VLS-128*, shown in Figure 2.3, the number of simultaneous laser beams are 128, with a vertical Field of View of $-25°$ to $15°$ and up to 300 m range[32]. This LiDAR laser is capable of a sampling rate of about 9.6 million points per second.

Figure 2.3: Velodyne VLS128 LiDAR laser scanner.

3D laser scanners are a very new technology, and are mostly used for high research projects, mainly in autonomous vehicles, like the Google Autonomous car [24] or the Stanford Junior

Vehicle[9]. This application benefits mostly from this scanners high sampling ranges and 360° horizontal field of view.

Figure 2.4: Google Autonomous Car.

2.2 Related Academic Work

Many scientific studies can already be found concerning the research and development of 3D sensors using laser scanners. In most studies, a cheaper alternative using 2D laser scanners is build instead of a more expensive 3D solution. In order to create a full 3D scan, the 2D laser is mounted on top of a moving platform and each individual laser scan is registered on a static frame of reference. The motion of the laser scanner can be classified as continuous or discontinuous. Usually a continuous motion is used for real-time systems, like autonomous vehicles, while a discontinuous motion is used when real-time is not important, like accurate 3D reconstructions of scenes. In the following paragraphs such systems that were developed so far are described.

In [30], a mobile robot was capable of autonomous navigation, thanks to a tilting *LMS200* laser scanner, that provided a depth map of the front of the robot, with a maximum resolution of 721×256 points. However, a scan of 181×256 points took about 3.4 s, and scans with more points (361 or 721) meant double or quadruple this time, making it not suitable for real-time operation. This previous system had a limited field of view, so in [2] a *LMS291* was mounted on a pan-tilt unit for generating a 3D point cloud with a parameterized field of view.

More recently, 3D laser scanner began being developed for continuous operation for real-time for Simultaneous Mapping and Navigation, or SLAM, for autonomous robots. This was specially due to the *DARPA Grand Challenge*, that offered a $1 million cash prize to the fastest autonomous unmanned vehicle that completed a 300 miles track. In [16], a 3D laser scanner (Figure 2.5) was developed by placing two *LMS200* planar laser scanners on a rotating vertical axis, capable of generating a high-quality 3D point cloud with a 360° field of view. Lots of other lasers were developed by rotating the laser in a continuous motion using a turntable [17], a swinging platform [34]. This sensor also became lighter, compact and modular, making it possible to integrate in multiple systems easily. One of this systems is *KaRoLa* (Figure 2.6), described in [19]. This laser scanner was then applied to several system, specially in search and rescue robots.

Figure 2.5: 3D laser scanner developed in [16].

Figure 2.6: The KaRoLa 3D laser scanner.

The 3D laser sensors are only able to reconstruct the geometry of the scene. Some sensors are also able to measure the intensity of the reflected light, create a grayscale value for each point. This intensity is measured, of course, in the frequency spectrum of the emitted light of the sensor, which is usually infrared (950 nm). To reconstruct the color, one or more cameras are coupled to the sensor, and both depth and color data is merged, in a process called fusion, to create colorized model. This is specially important for areas like architecture or archeology, where color information is very important. Such work can be seen on [3], where a 3D sensor like the one described in [30] was paired with a camera, to generate a 3D reconstruction with color. Another technique was created in [27], where the range and color images are captured separately and then a method is used to make the registration of the color images in respect to the extracted geometry. The registration was optimized for scenes with high geometry content.

This techniques are applied, for example, in cultural heritage, to model important art pieces. One of the most famous examples is the Michelangelo project [12], which developed a technique to register data from a triangulation sensor and color image data to reconstruct the 3D geometry of the statue of Michelangelo's David. One of the challenges in this project was to capture the chisel marks in the surface of the status, requiring a resolution of $1/4$ mm, in a statue 5 m tall [12].

2.3 Comercial Solutions

In this section two commercial solutions are shown and discussed. Both solution aim for a precise 3D reconstruction with color information, but the first solution does it with a structured light sensor and the second solution relies on a laser scanner.

2.3.1 Matterport

Matterport is advertized as an all-in-one solution, capable of both 3D reconstruction and capture 4K resolution images from the scene. Their target are mostly the reconstruction of indoor scenes, more specifically, the interior of houses. Then, the 3D model can be used to showcase the interior of the house, using both virtual reality or panoramic photography, or to make 3D measurements and automatically generate floor plans[15].

Matterport offers two products: a 3D camera and a cloud service to process the raw data taken with the camera. The camera, as seen in Figure 2.7, consists of two sensors: a structured light sensor and a photographic camera[15]. The structured light sensor has an advertized accuracy of 99% within the 4.5 m maximum range. The Photography sensor is a 4K HDR camera.

Figure 2.7: Matterport Pro2 Camera.

The overall process to capture a scene is fast and easy: the 3D camera is placed on a tripod and is controlled remotely. Each acquisition takes about 20 s and the result is a 3D colorized mesh with 4 million vertices and a 360° panoramic photography with 134.2 MP. To scan an entire environment, an operator moves the camera to each space and make multiple new acquisitions from that space.

A set of models reconstructed from the with the Matterport solution can be found in [13]. As an example, the model named "Pennsylvania Craftsman Home" [14] can be seen in Figure 2.8. This model represents the complete interior of an house and looks very realistic.

(a) Side view.

(b) Top view.

Figure 2.8: Matterport "Pennsylvania Craftsman Home" model.

2.3.2 Faro Focus

Faro Focus [5] are a series of 3D laser scanners targeted for the sectors of architecture, engineering, construction and product design. As such, this solution is capable of fast 3D reconstructions both on outside and inside environments with great accuracy. The 3D scanner, as seen in Figure 2.9, is design for portability and is equipped with a laser scanner with a precision of ± 1 mm and a range of 0.6 m to 350 m, and a 8 MP HDR camera.

An acquisition, like in the *Matterport Pro2 Camera*, is quick and easy, but does not require any remote computer, as the scanner incorporates a touch LCD screen. All the subsequent processing is done afterwards in a computer, using their proprietary software.

Faro Focus scans are very precise, as they are used for precise measurements of the reconstructed scene. As an example, a scan obtained by the Faro Focus can be seen in Figure 2.10 (from [29]).

2.4 3D Digital Models

Recreating a real scene with computer graphics is an important topic in today's world and many advancements are being made to create the illusion, for the user, that what he is seeing

Figure 2.9: Faro Focus 3D laser scanner.

Figure 2.10: Faro 3D scan.

is real. Many technologies and advancements appeared, like Virtual-Reality, high definition models, photo-realistic rendering and dynamic models. One of the biggest objectives of 3D reconstruction is the possibility of using this technologies to recreate real 3D scenes, making it possible for anyone with a computer or a VR-set to experience this scenes. The number of applications are huge, but the problem remains: how can we save a 3D environment?

In computer graphics, the most common representation is a polygonal mesh. It is composed by a collection of vertices, edges and faces composing a polygon that represents a simplification of the original geometry. Because of its flexibility and wide use, graphic cards are specially optimized to render this meshes and the results are very good. Properties of the object, like color and material, can be added per-vertex or per-face, making it a flexible model to save all the details of the scene. There are also many drawbacks to this representation, for example, the inadequate representation of non-linear surfaces, requiring large sampling to represent its reliably.

However, it is impossible to get a mesh directly from a 3D scanner, so a simpler model is used instead: the point cloud. A point cloud is a set of points sampled from the surface of

the objects, so its detail depend largely on the density of the points. Point clouds can also store extra data pixel-wise, for example, color, normals, intensity or segmentation index.

Because its such a simple representation, it is very used in robotics and 3D vision, for example, in search and rescue robots, for mapping and navigation, in unmanned vehicles, for road segmentation and in bin-picking robots, for object segmentation. However, it is not adequate for scene reconstruction, because a realistic model requires a huge density of points, which is unfeasible for numerous reasons. To start with, rendering point clouds is slow in modern graphic cards, because the rendering pipeline is not optimized for it, resulting in slow framerate, which then causes a poor experience for the user and unsuitable for VR. Despite new advancements in point-rendering algorithms, capable of rendering point clouds with billion points in commodity hardware [33], it will always be a limiting factor for this model. Also, some of the processing algorithms for point clouds have non-linear time complexity, so point clouds with many points can have long processing times. One of the most wide-spread solutions is to down-sample the point cloud to speed up the algorithms. Finally, large amounts of data are repeated in the point cloud, resulting in redundancy problems and large file sizes. For example, planar surfaces require the same density of points as any complex surface, while in meshes, the density of faces can be adjusted to the gradient of the surface.

These problems are usually solved by performing a triangulation of the point cloud, a process where a mesh is produced. However, this process requires a fine point cloud, that usually require many man-hours of thorough processing, to yield acceptable results. That is why LiDAR Laser scanner is so valuable for 3D reconstruction, as it yields raw point clouds with a better definition than any other technology, requiring less post-processing work in the triangulation process.

Chapter 3

Experimental Infrastructure

This chapter describes in detail both the hardware and software used in this project. The hardware - a mobile robot - is described in Section 3.1 and all the software implemented in this robot is described in Section 3.2.

3.1 Hardware

The hardware used in this work was a mobile 3D scanner called "lemonbot", shown in Figure 3.1. This scanner was developed to perform the acquisitions, and the goal was to build a platform that had minimum interference with the environment (for example, no cables are required), mobile (lightweight and easy to transport) and did not require the presence of the operator. Therefore, the robot was all packed into a tripod with a also has a battery capable of powering all the systems. The control is made via a remote connection through the router, which is specially useful to make the repetitive task of acquisitions faster and more agile.

This robot has, in total, 7 components. Three of which are the essencial components for the acquisitions: the 2D laser scanner, the camera and the pan-tilt unit, or PTU. The other 4 components form the infraestructure: the minicomputer, the wireless router, the battery pack and finally the tripod. Each of this components are described in detail next.

2D Laser Scanner

One of the objectives of this work is to use different 2D laser scanners to study the performance of the reconstruction and calibration algorithms and to evaluate if not every laser scanner is fit for this application. So, three laser scanners were chosen: the SICK LMS200, the Hokuyo UTM30LX and the Hokuyo URG04. Each of the laser scanners differ in their characteristics, like the size, price, range and error. In Table 3.1, all three laser scanners can be compared and in Figure 3.2 are shown.

Camera

The camera used in this work was a PointGrey Flea3 FL3-GE-28S4 Camera (Figure 3.3), which is extensively used in industrial and traffic applications. The high quality of the images, the programming interface and it's compact size and weight makes it perfect for computer vision applications in industrial environment. The most relevant characteristics are represented on the Table 3.2.

Figure 3.1: Lemonbot mobile 3D scanner

Table 3.1: Comparison of the three laser scanners used.

	SICK LMS 100	Hokuyo UTM 30LX	Hokuyo URG04
Aperture angle	270°	270°	240°
Angular resolution	0.25°	0.25°	0.36°
Scanning frequency	10 Hz	40 Hz	10 Hz
Maximum range	20 m	30 m	5.6 m
Systematic error	±40 mm	not available	not available
Statistical error	20 mm	30 mm	30 mm
Dimensions (mm^3)	152 × 102 × 106	60 × 60 × 87	60 × 60 × 87
Weight	1100 g	370 g	160 g
Power consumption	<12 W	8.4 W	2.5 W

(a) SICK LMS100 (b) Hokuyo UTM30LX (c) Hokuyo URG04

Figure 3.2: Laser scanners used.

Pan Tilt Unit

Both the laser and camera are placed on top of a pan and tilt unit for their movement. The ptu chosen was the FLIR PTU-D46 (Figure 3.4), which is a compact and light module with the characteristics in Table 3.3.

3.2 Software

This section describes the software used in this work. In general, two different applications were developed. One application was the software that constitutes the mobile 3D scanner, which was developed in a framework called Robot Operating System, described in Section 3.2.1. The other one is the software used to process the data recorded by the 3D scanner, which is described in Section 3.2.2.

Figure 3.3: PointGrey Flea3 FL3-GE-28S4

Table 3.2: Characteristics of the PointGrey Flea3 FL3-GE-28S4 Camera

Resolution	1920×1448
Framerate	15 fps
Pixels	2.8 MP
Color	Yes
Interface	GigE Vision
Power	12 V to 24 V
Dimensions	$29\,\text{mm} \times 29\,\text{mm} \times 30\,\text{mm}$

3.2.1 Robot Operating System

Robot Operating System, or ROS, is a software architecture for robot development, providing a collection of tools, libraries and conventions to simplify the development of complex robotic systems. It was originally created at stanford University in the mid-2000s and new is widely adopted as the standard framework for robotics by most research communities.

Its design principles follow the one of a distributed system. In ROS, a system is composed by multiple nodes that have just one task and communicate between them by message passing. To achieve this, ROS, in its core, has the infrastructure responsible for the:

Orchestration . ROS runs, stops and monitors all nodes, so in the case of failure, for example, ROS is capable of restarting the node.

Communication . ROS provides both the pipelining to distribute messages as well as the standard serialization specification.

Configuration . ROS provides a key-value store for parameters that are accessible for nodes. These parameters can be specified when the node is created and also changed dynamically at runtime.

Discovery . Each node in the environment can inspect it, such as finding other nodes and finding topics.

This architecture has many advantages, such as:

Table 3.3: FLIR PTU-D46 characteristics.

Pan range	$\pm 159°$
Tilt range	$-47°$ to $31°$
Maximum payload weight	4 kg
Angular resolution	$0.0032°$
Communication	serial interface
Size	small

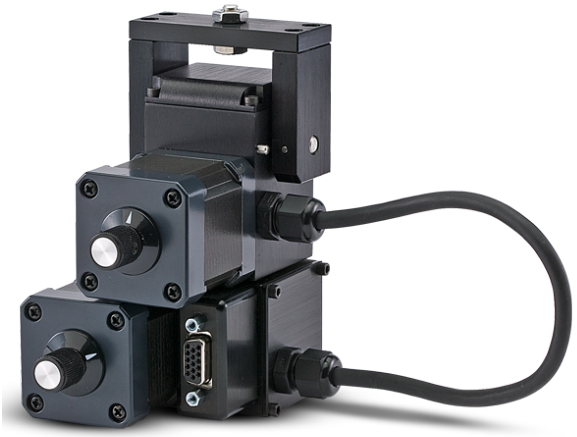


Figure 3.4: FLIR PTU-D46

Fault-tolerant . A failure in one node does not affect other nodes, so there is not a overall system crash, unlike monolithic systems. Usually, because errors are transitive and not very frequent, a restart-on-failure policy is used to keep the downtime low.

More generic . Because each node has a single responsibility, they tend to be more generic and detached to a single project, so integration in a new project can be easy. This is fundamental to reduce the need to "reinvent the wheel", therefore reducing the development cost and time of this complex systems.

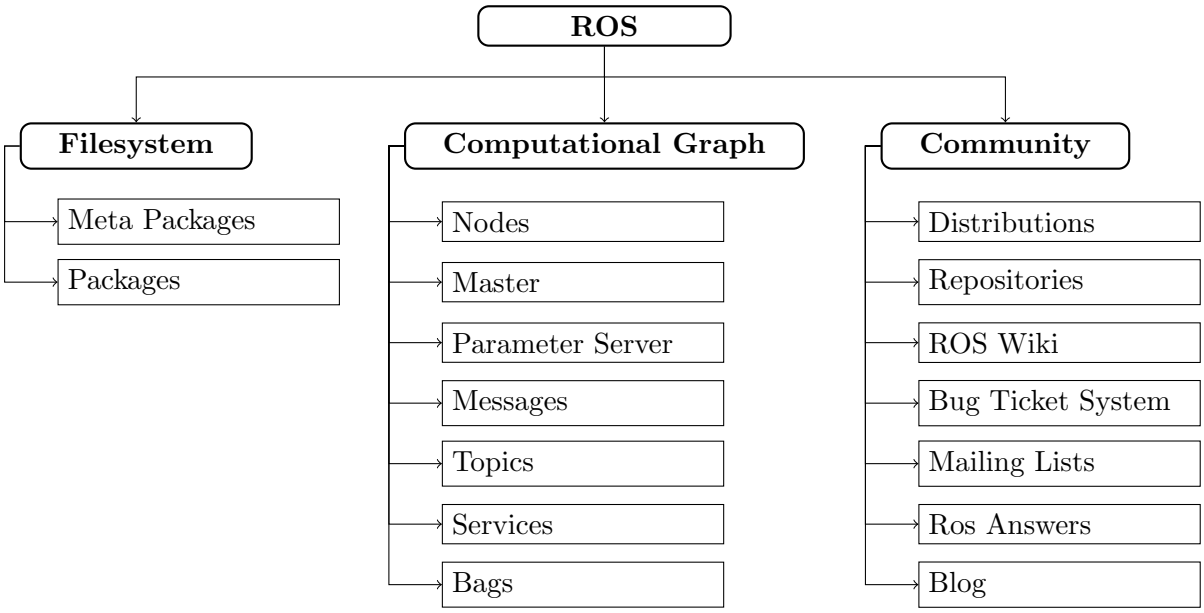
Easier to develop . Each component can be developed be separate developers, with different languages and independent release cycles, because they do not have any dependency between each over and only a message specification needs to be agreed on between developer teams, making collaborative development possible. Debugging is also easier, because each node can be unit-tested separated from the "real environment".

Large Community . ROS is open source, which incentivises different research groups to share packages. Also, ROS is widely adopted so multiple packages for numerous tasks are already programmed, and can be integrated easily into new systems. One of this

examples are drivers, which almost never require to be developed, because most robotic hardware already has a developed driver.

ROS is split into 3 levels, according to [6]: the *filesystem* level, the *the computational graph* level and the *community* level. Each level is composed of several core components, that make the whole system work, as seen in Section 3.2.1. Some components that were required for this work are explained next.

Figure 3.5: ROS architecture overview



Messages

Messages are the communication element and are just structures of data composed by primitive types such as integers, floating point numbers, strings and other messages. All messages follows a schema, which is required to encode and decode the message. Messages are serialized in a binary format before and after the exchange, so messages are small and efficient. Moreover, all messages have a header, which contains a timestamp and the source of the message.

Topics

Topics are named buses over which nodes exchange messages. Topics follow the publisher/-subscriber paradigm, so nodes can both subscribe to receive messages or publish messages to the topics. The exchange of data is done anonymously, so nodes are not aware which nodes are publishing or subscribing to a topic. This way of exchanging data is well suited for streaming data, such as sensor data.

Launch Files

Launch files are xml files that describe the steps to launch multiple node, as well as setting parameters. Launch files also support composition, so a launch file can invoque other launch files. Launch files were used in this project extensively, to launch the drivers of the mobile robot and to launch the acquisitions.

Bags

A bag is a file format for storing ROS messages, and have a myriad of tools to store, process, visualize and analyze them. During runtime, bags can be used to store the messages published in multiple topics, so data can be analysed later. Also, messages in bag files can be republished back into the system for testing or visualization purposes.

In this work, bag files played a very important role, as they were responsible to store the sensor data and also the transformation graph of the 3D scanner.

RViz

RViz is a 3D visualizer for ROS for displaying sensor data, like laser scans and point clouds, and the representation of the robot state, like the position of the coordinate frames and the joints. RViz can be a indispensable debug tool, for example, by comparing the real environment with the displayed environment shown in RViz.

TF

TF is a package that keeps track of multiple coordinate frames and maintains the relationship between coordinate frames in a tree structure, called the transformation graph. This transformation graph can be queried to obtain the transformation between two frames at any point in time. Also, tf can work in distributed systems, just like ROS, so any node can publish transformations and the transformations can be obtained in any node. TF is also responsible to interpolate between the discrete transformations and handle transformations with different sampling rates.

URDF

Unified Robot Description Format, or URDF, is a format to represent a robot model, like the joints and links configuration and the geometry of the joints. This file is loaded at runtime and the transformations are published according to the joint state.

3.2.2 Processing Application

This application required a wide spectrum of libraries, frameworks, file formats and graphical programs, which are described next.

Libraries and Frameworks

The software developed in this work that implements the processing algorithms was done using the Python programming language and some libraries to provide both data structures and common algorithms. Both the language and libraries are described next.

Python is a general purpose programming language that become popular for its syntax and small learning curve. It is also the defacto language for science, along with MATLAB. However, inlike MATLAB, it is s open source, has large community and has plenty of libraries that provide many algorithms and efficient data structures. Also, it is a dynamic language, which facilitates the process of testing and debugging the code developed.

Numpy is a library that contains an implementation of *nd*-arrays, as well as algorithms to manipulate them. This library was fundamental for this work to store and process the point data. The main advantage of this library is that it is implemented in compiled languages like C and Fortran to implement high performance and optimized data structures and algorithms, available through a clean interface in Python.

Pandas is a library that provides a fundamental data structure which was extensively used in this work: the DataFrame. DataFrames store data in columns, which is perfect to store tabular data. This is a common way to store point information, because point clouds are, fundamentally, tables, where each property are stored as a column, like x, y, z for position and r, g, b for color. Also, because it relies on numpy arrays to store the data, it is still very high performance.

PIL , or Python Image Library, is a library for image loading and manipulation, and was used to read and write the images recorded by the camera.

Jupyter provides an interactive interfaces, called Jupyter notebooks, which provides interactive documents with embedded code. This notebooks are extremely useful and were used to document and explore the code that was used in this work.

File Formats

AVRO is a binary data serialization format that is user to store collections of structured data. This format was chosen to store the laserscans and the image metadata. AVRO relies on schemas, which describe the data in the file is stored with the data. Therefore, an AVRO file is self-describing and data can be read and write without much overhead. Moreover, this format is implemented in Python and has numerous tools for inspection and conversion of the data.

PLY , or Polygon File Format is one of the most used and supported file formats to store three dimensional data, like point clouds and meshes. It was originally developed and used in the Stanford University to store data from 3D scanners. It supports a wide number of properties, like color, transparency, surface normals and texture coordinates. It also supports the storage of custom properties, which were required for this work, for example in the segmentation for the calibration. Moreover, it supports binary encoding, so files are small and fast to read and write.

JPEG is a commonly used format for images and was used to store the recorded images.

YAML is an human readable format that was used to store the parameters of the acquisitions, suck as the the extrinsic calibration of the sensors. The advantage of this format is that files are very easy to read and modify by the user.

Graphical Software

CloudCompare is a software to render, process and manipulate 3d point clouds. It includes many algorithms, like point cloud registration, re-sampling, scalar fields handling, and automatic or interactive segmentation. It can also render point clouds using different shaders and support point cloud decimation, which is a technique that allows manipulation of large point clouds without a decrease in performance. A screenshot of this software can be seen in Figure 3.6.

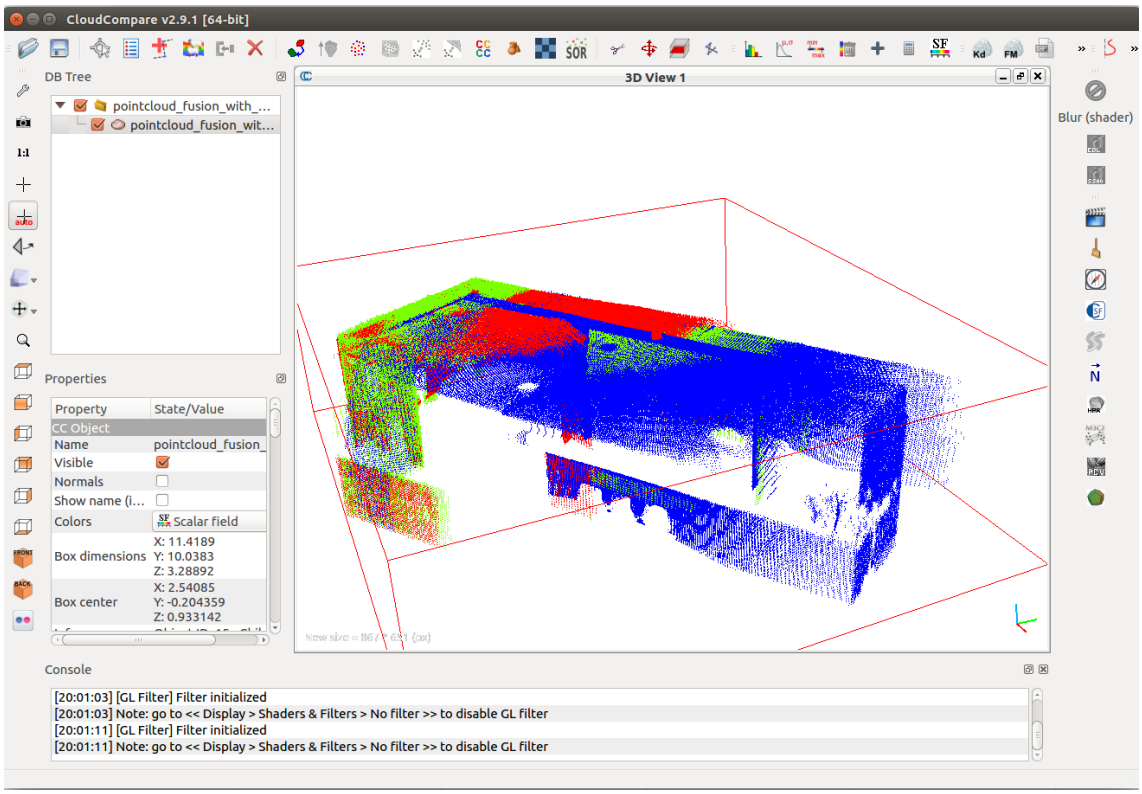


Figure 3.6: Cloud Compare screenshot.

Chapter 4

Methodology for Scene Capture

This chapter describes the concepts and methodology around a capture of a scene. To further explain the methodology, some core concepts need to be defined first:

Acquisition is a collection of sensor data collected in a specific pose in the scene. This sensor data are ether images, taken from a camera, or laser scans, taken from a 2D laser scanner. Each one of this sensor data is always tagged with temporal and spatial information, to identify where and when this data was recorded.

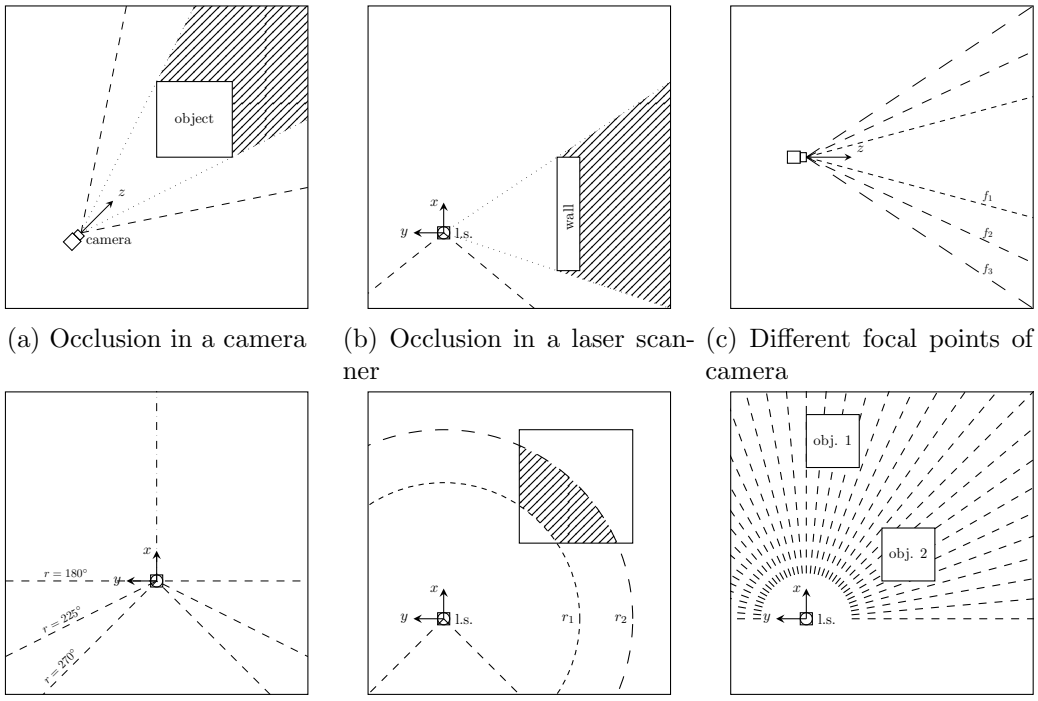
Capture is a collection of acquisitions taken from the same scene, from different poses and times.

A single acquisition has only limited information about the scene and is insufficient to create a complete reconstruction. This is often caused by occlusions, hardware limitations (like a small aperture, low range limits or low resolution) or environment factors (like reflective surfaces or lightning conditions). To circumvent this limitations, multiple acquisitions are taken from the same scene, to get enough data from it. However, this also comes with some challenges, for example, how to merge all the acquisitions and how to handle with all the redundant data.

So, acquisitions and captures are different levels and each one has a different method and objectives. In an acquisition level, the focus is on how to operate the scanner and define how the data is recorded. In a capture level, the focus is on how to plan multiple acquisitions so a good reconstruction is possible. In the following sections, both acquisitions and captures are further explained.

4.1 Acquisition

An acquisition is a collection of sensor data (laser scans and images) collected by the sensors in the scanner. Both sensors sample only a small subset of the whole environment: the laser scans only have points from a planar region of the space and cameras are limited by their focal length. To overcome this limitation, both sensors are moved to different poses in space to cover a wider space. In this case, the cause of movement of the sensors is the movement of the joints of the PTU.



(a) Occlusion in a camera (b) Occlusion in a laser scanner (c) Different focal points of a camera

(d) Radial aperture of a laser (e) Range limitations (f) Radial resolution

Figure 4.1: Limitations of a single acquisition

4.1.1 Movement Programming

To program the motion of the PTU joints, a list of waypoints in pan and tilt are defined and the joints move from waypoint to waypoint. The waypoints are defined in a grid in the joint-space and the movement between waypoints is the one that defines the shortest path possible and most of the movement is done in pan. So, each acquisition is parameterized with the following parameters: the range (minimum and maximum angle) of pan/tilt, the speed of each join and the number of waypoints in pan/tilt. An example of this parameterization can be seen in Figure 4.2.

Once the movement of the scanner was defined, the next step is to define when to record the laser scans and the images, according to it. Because of the nature of both sensors, it was established that laser scans are captured continuously during the pan movement between the waypoints and images are captured at every waypoint.

4.1.2 Parameterization Considerations

This methodology has the numerous implications.

First, the pan and tilt range is only limited by the PTU capabilities, but it beneficial to use the maximum range possible, in order to get as much data as it is possible, even if some data is redundant. In this work, the mobile scanner have a laser configuration such that the interpolation if different tilt angles is almost exclusively redundant data. However, this improves the final reconstruction by increasing the density of the point cloud.

Second, the number of laser scans recorded is going to depend on the pan speed and the frequency of scanning of the 2D laser scanner. So, it is expected that a laser scanner with a

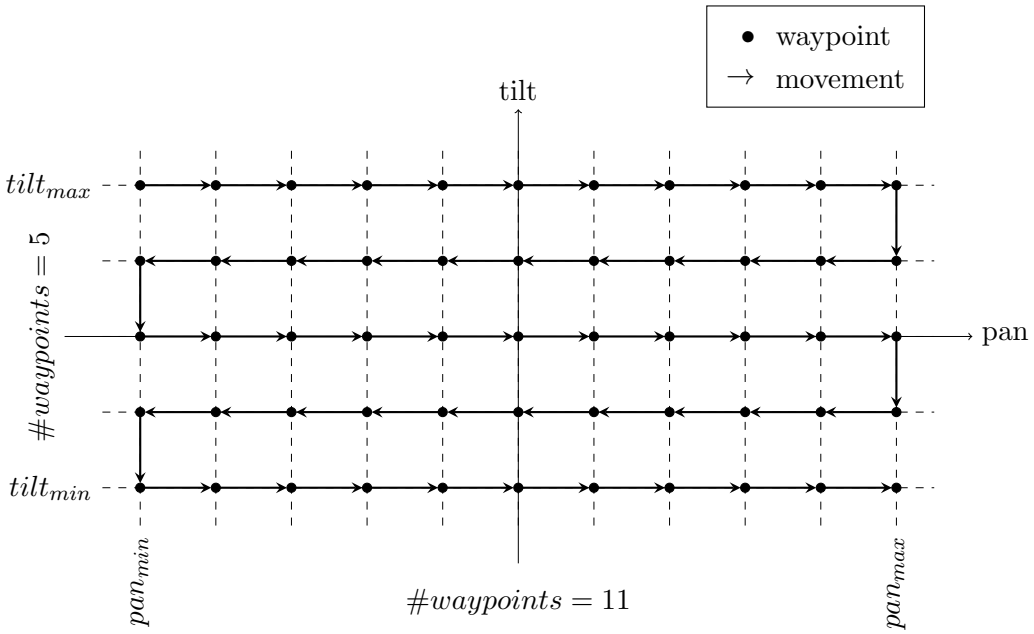


Figure 4.2: Waypoints and movements in the pan/tilt joint space

lower scanning frequency to require a slower speed compared to a faster one, to get the same results.

Third, the camera used in this work did not have stabilization so, to get sharp images, a complete immobilization was required in each waypoint. This was achieved by setting a time between the stop of all joints and the capture of the image by the camera. In this work, a time of 1.5 s was enough.

Last, the waypoints' angle increment has to be enough so that part of the previous image appear in the next image, so that every observable part of the scene is seen at least once. This depends heavily on the focal point of the camera: the bigger the focal point, the least area it captures and more waypoints are required.

4.1.3 Acquisition node

To implement this functionality, a ROS node was developed according to the previously defined specifications. This node, called *single_acquisition_node* is present in the *lemonbot_acquisition* package and the way is implemented is the following: the PTU movement is controlled by it and the selected messages are republished into a new topic. For convenience, all the acquisition topics are republished into the */acquisition* namespace. So, during an acquisition, two topics can be found, each one corresponding to each sensor, inside this namespace: the laser scans are in */acquisition/laserscans* and the images are in */acquisition/images*. This idea of republishing all the important messages greatly improved the acquisition organization, so all the topics that were required were also republished into this namespace. This topics were the */acquisition/camera_info*, containing the intrinsic parameters of the camera, and */acquisition/tf* and */acquisition/tf_static*, containing all the transformations of the robot.

Now, data from this topics need to be saved permanently, so this was done using a ROS tool called *rosbag*, that saves all the data from a predefined set of topics into a binary file

```
path:        acquisition_2018-09-07-16-01-46.bag
version:     2.0
duration:    4:53s (293s)
start:       Sep 07 2018 16:01:47.11 (1536332507.11)
end:         Sep 07 2018 16:06:40.96 (1536332800.96)
size:        87.0 MB
messages:    6690
compression: none [16/16 chunks]
types:       sensor_msgs/CameraInfo [c9a58c1b0b154e0e6da7578cb991d214]
               sensor_msgs/Image      [060021388200f6f0f447d0fcd9c64743]
               sensor_msgs/LaserScan  [90c7ef2dc6895d81024acba2ac42f369]
               tf2_msgs/TFMessage     [94810edda583a504dfda3829e70d7eec]
topics:      camera_info    953 msgs    : sensor_msgs/CameraInfo
               images          10 msgs    : sensor_msgs/Image
               laserscan     2788 msgs    : sensor_msgs/LaserScan
               tf            2938 msgs    : tf2_msgs/TFMessage
               tf_static        1 msg     : tf2_msgs/TFMessage
```

Figure 4.3: Example of a recorded bag file info

called a *bag* file. This was a easy and powerful solution, because it allows the acquisition to be reproduced again, by republishing all the messages back into the system. To save a set of topics, a node called *record* from the *rosbag* package is run with the list of topics that required to be recorded into disk. In this case, the required topics are all the topics inside the */acquisition* namespace.

To streamline the acquisition process, all this components (the acquisition node, the topic republisher nodes and the rosbag record node) can be all launched through a *launch file*. A set of all the parameters required for each acquisition can be override over the default parameters. Therefore, running an acquisition just requires a single command:

```
roslaunch lemonbot_acquisition single_acquisition.launch \
    pan_min:=-90 pan_max:=90 pan_vel:=10 pan_nsteps:=25 \
    tilt_min:=-15 tilt_max:=15 tilt_nsteps:=5
```

In conclusion, running the previous command will run an acquisition and in the end, a bag file will be present, with the topics *images*, *laserscan*, *camera_info*, *tf* and *tf_static*, therefore all the information relevant for the reconstruction.

To have a better insight in the bag file, a tool called *rosbag info* can be used. All the details about when the calibration took place, how long it took as well as how many messages it contains are printed. An example of this information is:

4.1.4 Data Serialization

Despite it's potentiality, bag files are not the best way to store the acquisition data for the reconstruction pipeline. There are some limitations of bag files in this application. The most noticeable is that the full transformation graph is stored, while in fact only the transformations between the start and end frame of the PTU are needed, as well as the transformations between the PTU mount link and each one of the sensors, which are static. Also, this

```
{
    "ranges" : [ ... ],
    "limits" : {
        "min" : 0.100000001490116,
        "max" : 29
    },
    "timestamp" : 1536174204611117487,
    "angles" : {
        "min" : -2.35619449615479,
        "max" : 2.35619449615479
    },
    "transform" : {
        "rotation" : [ ... ],
        "translation" : [ ... ]
    }
}
```


Figure 4.4: Example of laser scan row

transformation messages are not synchronized with the laser scans and image messages, which means an interpolation has to be performed each time the data is read. Another drawback is that bag files stores messages in it's own format, which hinder reading and inspecting the data, which can be helpful to check if an acquisition was successful. For example, the images are serialized into a ROS message, instead of being in a file with a known format, like *JPG*, which would allow for easier access and inspection.

To solve this issues, a preprocessing of the bag files was performed, to convert and extract all the important information into well known and useful formats. Each laser scan was stored in a *AVRO* file row that contains the timestamp (when it was taken), the minimum and maximum angle (aperture of the laser scan), the minimum and maximum ranges that the laser can capture, the transformation of the ptu and the list of all the measured ranges. An example of such row is show in Figure 4.4, obtained using the *avro cat* command. Each image was stored in a separate *JPEG* file and it's timestamp and transformation was stored in a row, again in a *AVRO* file. The parameters inherent to the acquisition, such as the name of the bag, the extrinsic and intrinsic calibration of the camera used and the extrinsic calibration of the laser was stored in a *YAML* file. The transformations in both the images and laser scans are stored as vector for translation and quaternion for rotation.

4.2 Capture

As seen before, acquisitions only capture a subset of the scene geometry and color, so multiple acquisitions are required. This problem can be partially solved by recording multiple acquisitions instead of once. Therefore, a capture is a collection of acquisitions of the same scene and it's goal is to contain as much data as possible from the scene, in order to create a proper reconstruction. However, this raises some challenges, on how to plan and execute the multitude of acquisitions and how to merge the data from all of the acquisitions (discussed in Section 5.4).

Planning determines where should the 3D scanner be places in each acquisition and the

```
bag: acquisition_2018-09-05-20-02-46.bag
camera:
    extrinsic:
    translation: [ ... ]
    rotation: [ ... ]
    intrinsic:
    principal_point: [ ... ]
    height: 1448
    focal_lenghts: [ ... ]
    width: 1928
    distortion_coef: [ ... ]
    distortion_model: plumb_bob
laser:
    extrinsic:
    translation: [ ... ]
    rotation: [ ... ]
    limits:
        max: 29
        min: 0.1
    angles:
        max: 2.356194
        min: -2.356194
```

Figure 4.5: Example of the parameters YAML file

sequence of the acquisitions. In this work, this was done prior to any acquisitions, according to a set of rules, to minimize the occlusion, maintain a minimum point density on all surfaces, capture color information of as much surfaces as possible and minimize the processing errors (specially the acquisition registration (see chapter III)). Each one of this problems and its solutions are explained in more detail hereupon.

To begin with, occlusion and range limitations restrict the covered area of an acquisition to a subset of the scene, which is dependent of the position and orientation of the 3D scanner in the scene. For example, in the example room, a 3D scanner placed in two different positions and orientations captures just a limited area of the scene.

Secondly, the point density decreases with the distance of the object to the sensor, which can influence the reconstruction, specially if the objects have smaller details. For example, a wall does not need a big point density, but a smaller object such as a chair or table should have a higher one. Therefore, the position and orientation of the acquisitions should regard this, such that the point density is adequate to the dimensions of the objects.

At last, the acquisition registration requires that between each acquisition there is enough overlap between the point clouds, so a transformation between acquisitions can be determined. So, between each acquisition there should be a maximum distance, such that this registration is possible. Also, this registration requires a good initial estimate for the transformation, otherwise it is not able to find a correct transformation. The solution proposed is to define a sequence of acquisitions such that each subsequent acquisition is near to the previous one and the relative rotation is small.

In conclusion, a good capture planning requires that key acquisitions are made to minimize occlusion and maintain a adequate point density and multiple acquisition have to be made,

connecting the key points, and each acquisitions should be close enough to the previous one, such that the registration between acquisitions are possible. In this work, we determined this sequence of acquisitions by determining a path inside the scene. This process, however, can be very subjective and dependent of the user, and the evaluation of the capture is all done afterward, because no feedback exists during the capture.

Chapter 5

Methodology for Geometry Reconstruction

This chapter present the methodology to reconstruct the geometry of the scene. In Section 5.1, the laser scans of each acquisition are transformed into point clouds. In Section 5.2, two calibration methods, one of which was developed in this work, are described to obtain the extrinsic calibration of the laser scanner. Section 5.3 describes a method to estimate the normals, based on the structure of the point cloud. In Section 5.4, a method to find the transformations between acquisitions is described, to merge the acquisitions into one point cloud. Finally, in Section 5.5, three point cloud filters used in this work are described.

5.1 Point Registration

Each laser scan is a collection of points in polar coordinates, so each range point (r_i, θ_i) is transformed to a point in the laser frame of reference according to Equation (5.1). The angles are uniform distributed between a minimum and maximum angle, θ_{min} and θ_{max}, respectively, so $\theta_i = \{\theta_{min},\ \ldots,\ \theta_{max}\}, i = 1 \ldots N$. The index i is defined as the range index of each laser scan.

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} r_i \cos(\theta_i) \\ r_i \sin(\theta_i) \\ 0 \end{pmatrix} \tag{5.1}$$

Further on, each point p_{ij} is registered in the referencial of the acquisition. According to the transformation graph (see Figure 5.1), there are two transformation from the acquisition frame and the laser scanner frame: the transform from the acquisition frame to the PTU frame ${}^{ptu}_{acq}\mathcal{T}$, which is dynamic and changes for each laser scan, and the transformation from the PTU frame and the laser scanner frame ${}^{laser}_{ptu}\mathcal{T}$, which is static. This two transformations can be chained together to obtain the point in the acquisition frame, according to Equation (5.2).

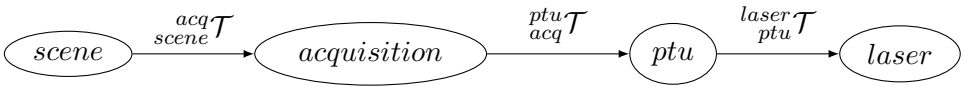


Figure 5.1: Transformation graph

$$p_{i,j} = \begin{pmatrix} x_{i,j} \\ y_{i,j} \\ z_{i,j} \\ 1 \end{pmatrix} = {}^{ptu}_{acq}\mathcal{T} \cdot {}^{laser}_{ptu}\mathcal{T} \cdot \begin{pmatrix} r_i \cos(\theta_i) \\ r_i \sin(\theta_i) \\ 0 \\ 1 \end{pmatrix} \tag{5.2}$$

At this phase, each point has 2 indexes, one for the laser scan index $j = 1 \dots L$ and another for the range index $i = 1 \dots N$, relative to the each laser scan. Despite that point clouds commonly only use one index, at this stage the points are structured in a bidimensional structure of $L \times N$ points. This structure is useful in the normal estimation phase, explained in Section 5.3.

This reconstruction phase depends heavily on the transformation from the PTU to the laser scanner. This transformation is obtained by a calibration process and is commonly referred as the extrinsic calibration of the laser scanner. The calibration method used to obtain this extrinsic calibration is explained in detain in Section 5.2.

In conclusion, for each acquisition results a point cloud with $L \times N$ points, where L are the number of laser scans and N the number of range values in each laser scan. Each point can be indexes in a bidimensional index, which can be useful for subsequent algorithms.

5.2 Laser Extrinsic Calibration

The key for a good geometric reconstruction is the laser scanner extrinsic calibration, which has to be accurate, so that every point is correctly registered. Therefore, two calibration methods are here presented: the RADLOCC camera-laser calibration (Section 5.2.1) and a new method developed in this work (Section 5.2.2), that aims to achieve better results than the latter.

5.2.1 Radlocc Method

[35] presents a method for auto calibration of a camera with a laser scanner. This method, known as Radlocc, uses information from both sensors and tries to find point correspondences to optimize the calibration. In this work, this method was used together with the extrinsic calibration method (explained in section ??), to obtain the full extrinsic laser calibration.

To use this method, the user has to obtain a calibration dataset, which is a set of synchronized images and laser scans containing a chessboard in multiple poses. The chessboard serves as the calibration object, which is the link between the two sensors. In this work, a ROS package was developed to handle this capture and to convert between the ROS messages and the RADLOCC format. Its source code and all documentation can be found in `https://github.com/bernardomig/radlocc_calibration`. In the 3D scanner, the laser scanner was positioned such that the laser scans are horizontal and about 20 to 30 images were taken per dataset. Such images are shown in Figure 5.2.

First, a chessboard extraction algorithm finds both finds the intrinsic calibration of the camera, as well as the poses of each chessboard in the camera coordinate frame. Then, laser scans are segmented into board/background, and all the board points are extracted, as seen on Figure 5.3.

Then, the reprojection error of the laser scans points to the chessboard plane are computed, and the transformation from

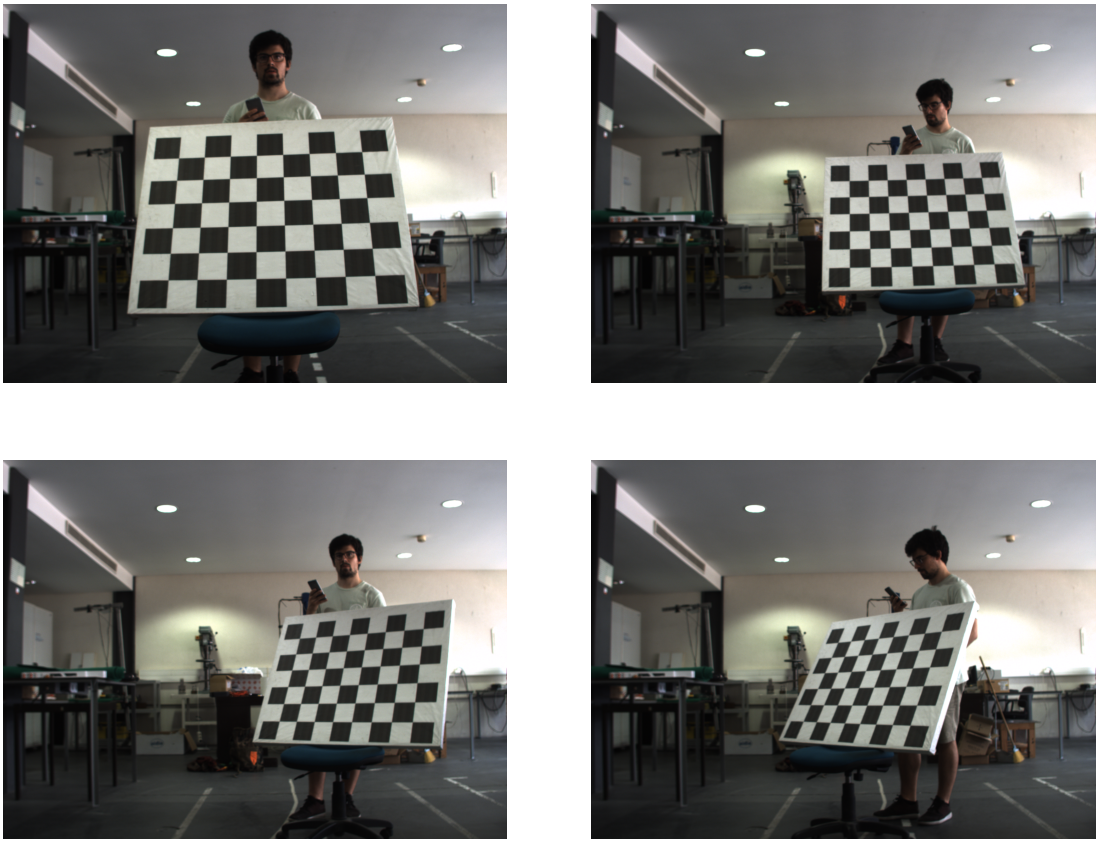


Figure 5.2: Images captured for RADLOCC

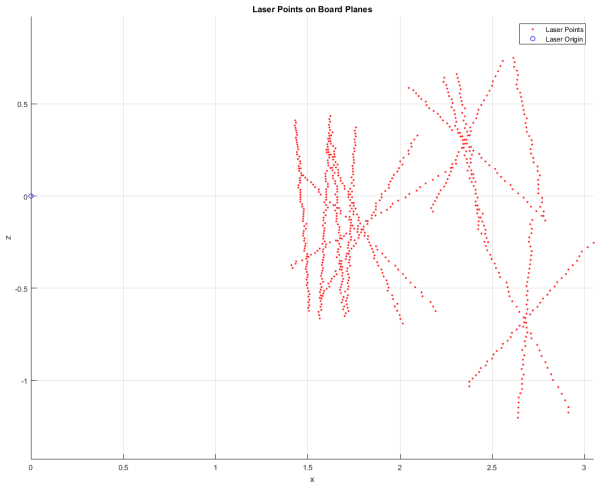


Figure 5.3: Radlocc laser scans chessboard extraction

Then, the reprojection error of the laser scans points to the chessboard plane are calculated, depending of the transformation from the camera to the laser scanner. The reprojection error is minimized during the calibration optimization, and the transformation from the camera to the laser if found.

Finally, the extrinsic calibration can be calculated as in Equation (5.3).

$$\mathcal{T}_{calibration} = {}^{laser}_{PTU}\mathcal{T} = {}^{camera}_{PTU}\mathcal{T} \cdot {}^{laser}_{camera}\mathcal{T} \tag{5.3}$$

5.2.2 Proposed Method

A new method was developed in this work to calibrate the laser scanner in this system. One of the key differences to the previous method (in Section 5.2.1) is that the calibration is done only using the laser range data, and the calibration from the PTU to the laser scanner is directly find. The hypothesis that this method relies, is that in a good calibration the deviation of a point set is minimal. In other words, in a point set representing a planar surface, the deviation from the points to the planar surface is the lowest, if the extrinsic calibration is the correct one.

This method is, therefore, an optimization problem. For each extrinsic calibration transformation $\mathcal{T}$, corresponds a point cloud $\mathcal{P}$, following the method shown on Section 5.1. Then this point cloud is evaluated by a loss function, which determines quantitatively how good each generated point cloud is. Finally, an optimizer will find the transformation $\mathcal{T}$ that minimizes the loss function. Each one of these steps is described in detail next.

Segmentation

This calibration method uses an acquisition as its dataset, which is a big advantage. Also, a point cloud has to be generated using a estimation of the calibration transformation. This point cloud does not have to be geometrically accurate but needs to be sufficient for the plane segmentation, which is done manually prior to the calibration. In this work, the software CloudCompare was used to segment the point cloud into multiple planes, and the data was saved as a scalar index in each point. An example of a segmentation can be seen in Figure 5.4, where each cluster is represented with a different color.

The segmentation was not done automatically because most segmentation algorithms, for example the RANSAC algorithm, were not capable of achieving a reliable segmentation for the initial estimate, because the point cloud had significant deformation. In addition, manual segmentation is easy to do and accurate, considering that it is a one-time process.

During the optimization, this segmentation serves as a blueprint for all the segmentations. Each point cloud is generated in the same way, so the sequence of points is always the same. Therefore, it is always possible to match any point on the generated point cloud to the point in the segmented point cloud, and get the corresponding cluster index for all the points.

Loss Function

A loss function, or cost function, is a measure in optimization that compares the result of a model with its expected result, and returns a value that signifies the difference between the two. More concretely, in this calibration the loss function has two components: the loss function per cluster and the loss function per point cloud, which combines the loss of all the clusters.

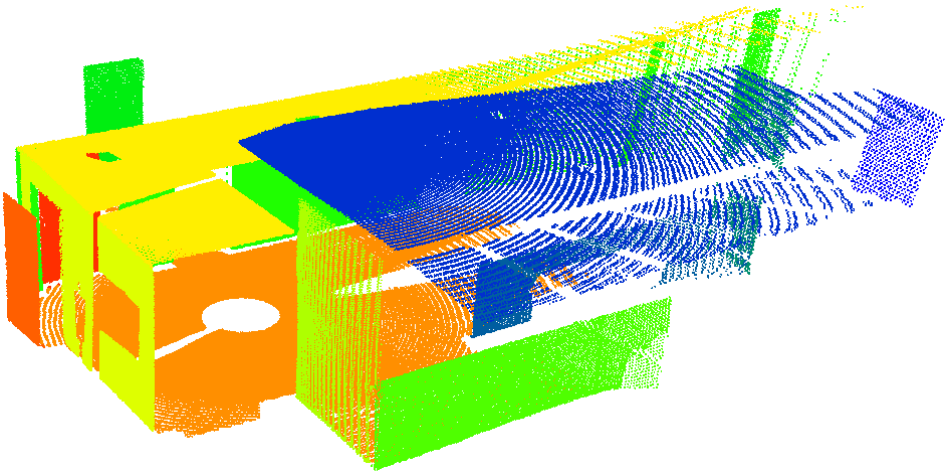


Figure 5.4: Example of a plane segmentation, where each color represents a cluster

To start with, the plane equation for each cluster is found, using the same method as in Section 5.3, the Principal Component Analysis, or PCA. First the centroid of each plane is found, which is the same as the mean value of all the points $\bar{p}$ (Equation (5.4)). Then, the covariance matrix $\mathcal{C}$ is calculated (Equation (5.5)).

$$\bar{p} = \sum_i p_i \tag{5.4}$$

$$\mathcal{C} = \sum_i (p_i - \bar{p}) \otimes (p_i - \bar{p}) \tag{5.5}$$

Then, the principal axes of the plane is find by an eigen decomposition of the covariance matrix. The smallest eigenvalue λ_3 will be the variance σ^2 of the cluster. In other words, σ^2 is the mean square of the orthogonal distance of all points in the cluster to the plane. So, σ^2 can be a quantitative factor to measure the loss for each plane. Formally, let us admit that the σ^2 has two components: the statistical error of the laser sensor σ^2_{sensor}, which is not affected by the calibration and a second component σ^2_{calib}, which results from the calibration error. Thus, it is possible that by minimizing σ^2, a exact calibration can be obtained. For this calibration, however, the value σ was used instead of σ^2, which is known as the Root Mean Square Deviation, or RMS. Therefore, the loss of each cluster will be the σ value.

Next, the losses of the clusters are combined into a scalar value, which is the loss of the point cloud. The method found was to, again, calculate the RMS of the values of the partial losses $loss_i$, according to the Equation (5.6). This value is expected to be minimal when all the partial losses are minimal which, according to this hypothesis, corresponds to a correct calibration.

$$RMS = \sqrt{\sum_i^N x_i^2} \tag{5.6}$$

Paramerization

The parameters in this calibration should define a geometric transformation in space, which is, in the end, a transformation matrix (Equation (5.7)). This transformation can be decomposed into two components, a translation and a rotation. The translation can be represented as the vector $t = (t_x, t_y, t_z)$, and the rotation can be represented as a 3×3 rotation matrix R. Since a rotation matrix has only $3 \times 3 = 9$ elements but only 3 degrees of freedom, another parameterization has to be used to represent a rotation. Popular parameterization for rotations are euler angles, quaternions and axis/angle representation.

$$\mathcal{T} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.7}$$

However, not all representations are suitable for an optimization. In fact, in [8] the term fair parameterization was introduced: a parameterization is called fair, if it does not introduce more numerical sensitivity than the one inherent to the problem itself. Therefore, fair parameterization are a requirement for optimizations, as it increases the changes of convergence. For example, euler angles, which are probably the most used angle parameterization, are not suitable for optimizations [22], because they do not yield smooth movements, each rotation is non-unique and, most notably, there are singularities, so-called *gimbal-lock* singularities, where one degree of freedom is lost [22]. Also, quaternions are not suitable for optimizations, because quaternions have 4 components which are norm-1 constrained. Despite being a fair parameterization, quaternions introduce some complexity in the algorithm to handle the norm-1 constrain, so they are not usually used for optimizations [22].

The axis/angle parameterization is the most widely used to represent a rotation in an optimization, as it is a fair parameterization and has only three components. Any rotation can be represented as a rotation around an axis a, by an angle θ. Since a only represent the direction of the rotation (hence only has 2 degrees of freedom), it can be combined with the angle θ into a single vector ω, as in Equation (5.8).

$$\begin{aligned} \theta &= |\omega| \\ a &= \frac{\omega}{|\omega|} \end{aligned} \tag{5.8}$$

Computing the rotation matrix from ω is done using the Rodrigues' formula (Equations (5.9) and (5.10)) [22].

$$R = I + \frac{\sin\theta}{\theta}[\omega] + \frac{1 - \cos\theta}{\theta^2}[w] \tag{5.9}$$

$$[w] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{5.10}$$

In conclusion, the parameter vector will have 6 values: 3 representing the translation (t_1, t_2, t_3) and 3 representing the rotation in the axis/angle representation (r_1, r_2, r_3). So, the parameter vector is shown in Equation (5.11).

$$P = \{t_1, t_2, t_3, r_1, r_2, r_3\} \tag{5.11}$$

Optimizer

The optimization is performed by the Powell's method, described in [21]. This method finds a local minimum of a multi-dimensional unconstrained function, and does not require the gradient of this function, which fits this particular optimization. This method is implemented in the python scientific library SciPy [23].

Overview

To sumarize, the overview of the entire steps of the calibration is shown in Figure 5.5.

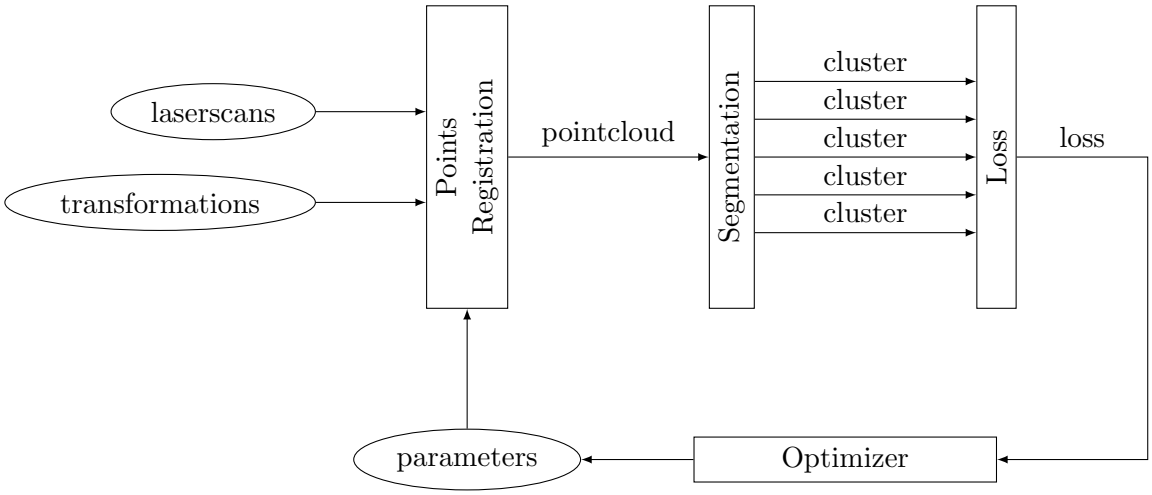


Figure 5.5: Calibration Overview

5.3 Normal Estimation

Surface normals are an important property of geometric surfaces, and are fundamental for meshing algorithms and computer graphics, as for example, to calculate the shading and other visual effects [4]. As an example, the Stanford Bunny model [31] rendered with and without normals are show in Figure 5.6. Normal estimation is quite trivial for surfaces, but for point clouds the process is quite not as easy. Usually there are two ways to estimate the normals: either by meshing the surface first, and then calculate the normals for the mesh, or using the point cloud itself to infer the normals. However, most meshing algorithms already require the normals to achieve a good result, so the latter option is more effective.

The most common solution is, for each point, to find the k closest point, defined as the k-neighborhood of a point, and calculate the normal of the best-fitting plane formed by this points. However, finding the k-neighborhood of all the points in a point cloud has a time complexity $O(N \log N)$, so it can become quite slow for point clouds with a large number of points. In this work, a new solution was used to find the closest points, exploiting the bidimensional structure of the point cloud. This solution has a linear time complexity $O(N)$, which makes it a valuable solution for large point clouds.

The solution were presented acknowledges that each point in the point cloud resulting from Section 5.1 has two indexes, one for the range index i and one for the laser scan j. For

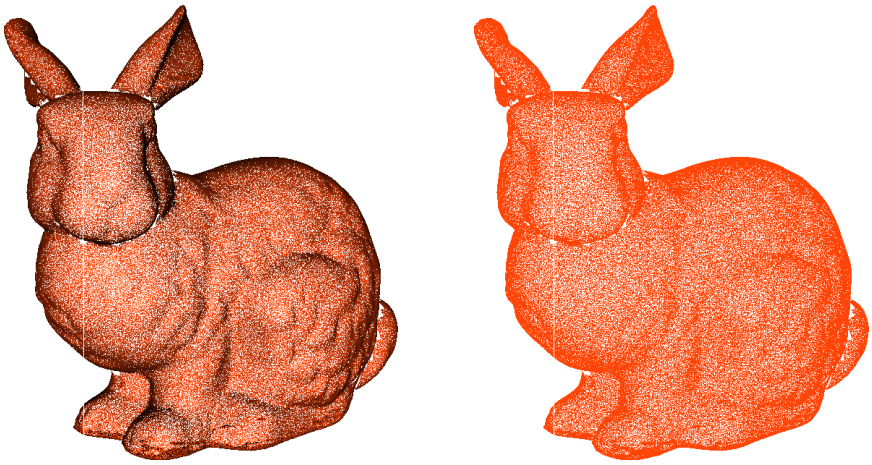


Figure 5.6: Stanford rabbit [31] rendering with (left) and without (right) normals

each laser scan, each point p_i has a neighborhood $p_{i-k}, \ \ldots, \ p_{i+k}$, because each subsequent point has an increasing angle to the previous point. Between successive laser scans each point has an increasing angle (the pan angle) to the previous one. Therefore, for this algorithm, the neighborhood of each point is show in Equation (5.12). The value of k_1 and k_2 have to be adjusted for a better result, because if the values are large, fine details are going to disappear and edges are going to be smeared, and on the other hand if the values are small, the surface will appear as too noisy.

$$neighborhood(p_{i,j}, k_1, k_2) = \{p_{i-k_1,j-k_2}, \ \ldots, \ p_{i+k_1,j+k_2}\} \tag{5.12}$$

Then, for each point, the tangent plane that fits the neighborhood is calculated, which in turn is a least-square plane fitting problem. This is usually solved by an analysis of the eigenvalues and eigenvalues of the covariance matrix of the neighborhood points. This method is also known as the Principal Component Analysis.

To start with, for any neighborhood, the covariance matrix $\mathcal{C}$ is assembled as in Equation (5.13), where n is the number of points in the neighborhood and $\bar{p}$ is the centroid of the points ($\otimes$ is the tensor product). Then, an eigen-decomposition is performed in the covariance matrix to calculate the eigenvalues λ_i and the corresponding eigenvectors $\mathtt{v}_i$. Finally, the direction of the normal vector of the point is the eigen vector corresponding to the smallest eigenvalue, so $\hat{n} = \pm\mathtt{v}_3$.

$$\mathcal{C} = \frac{1}{n}\sum_{i}^{n}(p_i - \bar{p}) \otimes (p_i - \bar{p}) \tag{5.13}$$

Then, the orientation of the point has to be defined, because the result of the PCA is ambiguous, which may lead to inconsistent normals in the point cloud. In this case, the solution found was to orientate the normals towards the frame of the 3D scanner, which for each acquisition if the zero position. Therefore, each normal has to satisfy the Equation (5.14).

$$\hat{n} \cdot p < 0 \tag{5.14}$$

Further, there should be some filtering, specially for points at an edge, because the neighborhoods method described here is going to fail in this case. So, a proposed solution is to

reject any point that exceeds a certain threshold distance to the center point. This has the advantage that the normal estimation can still work in the edges.

5.4 Acquisition Registration

During an acquisition, multiple acquisitions are done and to each one corresponds a transformation (position and orientation) to the scene referencial. In this section, a method is described to find each one of this transformations, so all the acquisitions are merged into a single point cloud. The method chosen is the Iterative Closest Point, or ICP. This method is capable of aligning two point clouds, the reference and the target point cloud, by finding the transformation between the second to the first one. This is also known as point cloud registration.

5.4.1 Iterative Closest Point

This method can formally be described as follows: let $\mathcal{P}$ be the target point cloud and $\mathcal{Q}$ the reference point cloud. Then, the aim of the registration is to estimate the transformation $\mathcal{T}$ from the referencial of $\mathcal{P}$ to $\mathcal{Q}$ by minimizing the error function $\text{error}(\mathcal{P}, \mathcal{Q})$ in Equation (5.15), where $\mathcal{T}(\mathcal{P})$ is the result of the application of the transformation $\mathcal{T}$ to the point cloud $\mathcal{P}$.

$$\mathcal{T} = \underset{\mathcal{T}}{\text{argmin}}(\text{error}(\mathcal{T}(\mathcal{P}), \mathcal{Q})) \tag{5.15}$$

The error function $error(P, Q)$ is computed on pair of points that are associated between the two point clouds. This association is, ideally, between points that are closest in position in both point clouds. Then, the distance between the matching points (p_i, q_i) are used in the error function in Equation (5.16). The matching algorithm can be based on features or geometric properties, so a better matching can be found. In this work, a simple point-to-point matching was used.

$$\text{error}(\mathcal{P}, \mathcal{Q}) = \sum_{(p_i, q_i)} |p_i - q_i| \tag{5.16}$$

In order to make the error function more robust, outlier points can be removed first from the match list. In addition, weights w_i can be associated to each matching points (p_i, q_i) to increase or decrease the influence of each matching points in the error function. As an example, normals can be used as a weight, so points with similar normals ($w_i = n_{p_i} \cdot n_{q_i}$) have a greater influence in the error function.

However, the result of this minimization is always dependant on the association between the points, which, unless the descriptors are good enough (like visual correspondences), the matching is not perfect, and is worst the farther apart both point clouds are. The idea behind the ICP algorithm is that, even with a bag association, the resulting estimate can be used to find a better one. So, the ICP algorithm creates a series of transformation $\mathcal{T}_i$ at each iteration, yielding a new transformed point cloud $\mathcal{P}_i$. Then, the next transformation is found:

$$\mathcal{T}_{i+1} = \underset{T}{\text{argmin}}(\text{error}(\mathcal{T}_i(\mathcal{P}_i), \mathcal{Q})) \tag{5.17}$$

Finally, the final transformation estimate is the composition of all the intermediary transformations:

$$\mathcal{T} = \mathcal{T}_1 \circ \mathcal{T}_2 \circ \cdots \circ \mathcal{T}_N \tag{5.18}$$

5.4.2 Multiple Point Cloud ICP

ICP can only register pairs of point clouds, whereas this work requires a registration of N point clouds, corresponding to n acquisitions. So, a technique has to be found so that the ICP algorithm can be used with n point clouds. Three of this such techniques are now describes, ordered by their complexity.

The first approach and the easier one to implement is to register each point clouds sequencially. In other words, this method registers the point cloud $\mathcal{P}_i$ to the point cloud $\mathcal{P}_{i-1}$ and the transformation $\mathcal{T}_{i-1}^{i}$ is found. The final accumulated point cloud is assembled using the Equation (5.19). This method is the one that requires less overall registration, but has the disadvantage that the transformation errors increases for each successive point cloud. This approach is shown in Figure 5.7a.

$$\mathcal{P} = \bigcup \left(\mathcal{T}_1^2 \circ \mathcal{T}_2^3 \circ \cdots \circ \mathcal{T}_{i-1}^{i}\right)(\mathcal{P}_i) \tag{5.19}$$

The next approach is widely used in robotics for Simultaneous Location and Mapping , or SLAM. This method holds an accumulated point cloud A in memory, and each new incoming point cloud $\mathcal{P}$ registers to the accumulated point cloud. Afterwards it is merged into A, which is then used for the next iteration, as shown in Figure 5.7b. It has the advantage that each new registration is done against a wider point cloud and has a smaller influence than in the previous method. Also, at each iteration the current pose of the 3D scanner is obtained, which is used as an initial estimate for the next iteration. However, the accumulated point cloud grows at each iteration, so a down-sampling is done at each iteration to keep the number of points bounded. In conclusion, each iteration can be calculated by Equations (5.20) and (5.21).

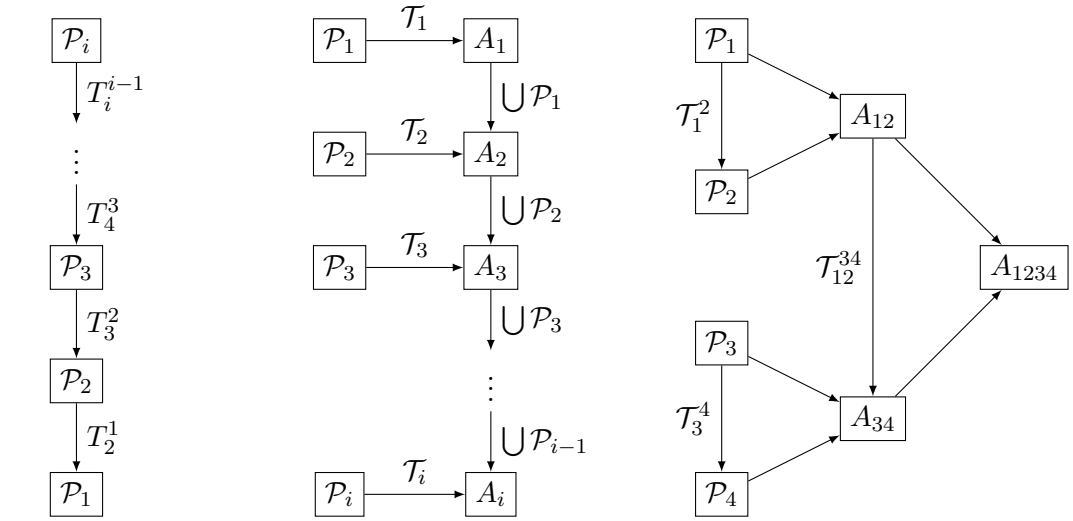
$$\mathcal{T}_i = \text{ICP}(A, \mathcal{P}_i, \mathcal{T}_0 = \mathcal{T}_{i-1}) \tag{5.20}$$

$$A_{i+1} = Ai \bigcup \mathcal{T}_i(\mathcal{P}_i) \tag{5.21}$$

The last approach is the most complex. The idea of this approach is to minimize the number of transformation combinations, to minimize the propagation of the error. In particular, the registrations for the N points clouds are done pairwise and are merged together to create $N/2$ point clouds. Then, this process is done recursively until an unique point cloud is obtained. This way, the maximum number of transformation combinations are equal to the number of levels of the tree, which is $\log_2(N)$, instead of N combinations in the first approach. This algorithm is formalized in Equations (5.22) and (5.23), for a list of point clouds $S = \{P_1, P_2, \ \ldots, \ P_n\}$. At each level l a new list of point clouds ${}^{l}P$ and transformations ${}^{l}T$ are computed, as shown in Figure 5.7c.

$${}^{l}\mathcal{T} = \left\{\text{ICP}({}^{l-1}\mathcal{P}_1, {}^{l-1}\mathcal{P}_2), \ \ldots, \ \text{ICP}({}^{l-1}\mathcal{P}_{n-1}, {}^{l-1}\mathcal{P}_n)\right\} \tag{5.22}$$

$${}^{l}\mathcal{P} = \left\{{}^{l-1}\mathcal{P}_1 \bigcup {}^{l}\mathcal{T}_1({}^{l-1}\mathcal{P}_2), \ \ldots, \ {}^{l-1}\mathcal{P}_{n-1} \bigcup {}^{l}\mathcal{T}_{n/2}({}^{l-1}\mathcal{P}_n)\right\} \tag{5.23}$$



(a) First method (b) Second method (c) Third method

Figure 5.7: Multiple Point Cloud ICP approaches

In conclusion, three methods are possible to extend the ICP algorithm to multiple point clouds, and the three methods were used in this work and compared. After all the registrations are performed, point cloud can be assembled from all the point clouds, to form a big point cloud of the final scene. There is, however, a limitation of all this methods, because all of them have the principle that every point cloud is close to the previous one, which can be false. In this work, this was ensured in the capture methodology.

5.5 Filters

The final point cloud, after the assembly from every acquisition's point cloud, can have unnecessary or redundant information, which can make the point cloud size too large for any use. A common solution is to use filters to remote unnecessary points and downsample the point cloud.

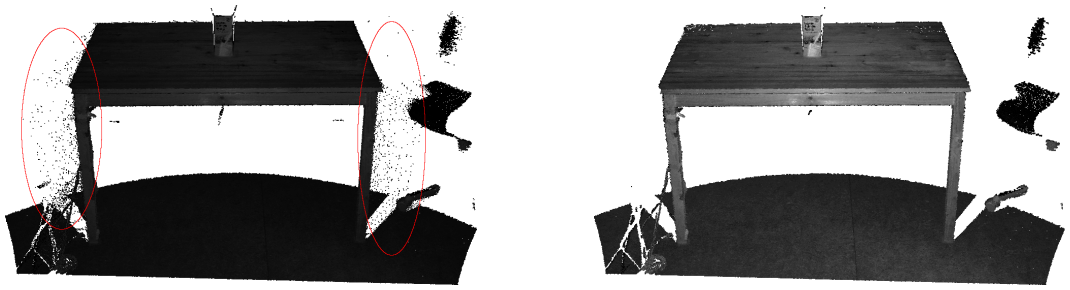
5.5.1 NaN Removal

The first filter is the Not a Number removal, or NaN removal. In the acquisition, any range that is not measured is stored as a NaN, to signal that they are missing. During the point registration phase, all this missing ranges remain as NaN, and should be removed, because their information is irrelevant and take as much space as a real value. So, each point that contains a NaN value is removed from the final point cloud.

5.5.2 Statistic Outlier Removal

Usually point clouds contains different point densities, dependent on the distance of the object to the sensor. Also, measurement errors also occur next to edges or corners. As a result, point clouds tend to have sparse outliers that can affect subsequent algorithms, like segmentation or registration algorithms. A usual solution is to perform an statistically analysis on each

point, removing the ones that do not reach a certain criteria. In particular, the mean distance of each point to its neighbors is computed, and if this distance is outside an interval centered in the mean of all the distances, then it is removed. An example can be seen in Figure 5.8.



(a) Before SOR (b) After SOR

Figure 5.8: SOR filter in a point cloud

5.5.3 Voxel Grid Downsampling

This method downsamples, that is, reduce the number of points of a point cloud, using a voxel grid. A voxel is a cubical space and is the element in a tridimensional grid. So, each point in the point cloud belongs to some voxel. Then, in each voxel, all the points are represented by their centroid. This is an effective and fast method to downsample a point cloud. The level of detail can be parameterized with the voxel leaf-size (the size of each voxel in the x, y, z direction). A smaller leaf-size maintains more details but generates a bigger point cloud. A bigger leaf-size does not keep as much detain but generates a smaller point cloud. As an example, Figure 5.9 shows the Stanford Lucy model [28] after a voxel grid downsampling with different leaf size values: Figure 5.9a with 2 mm (288.000 pts), Figure 5.9b with 5 mm (55.000 pts) and Figure 5.9c with 2 mm (18.000 pts).

(a) Leaf size of 2 mm (b) Leaf size of 5 mm (c) Leaf size of 8 mm

Figure 5.9: Stanford Lucy scan [28] after a voxel grid downsampling with different leaf sizes

Chapter 6

Methodology for Image Registration

This chapter describes the methodology for image registration, that is, the process that colorizes (defines the color) the point cloud based on the images taken in the acquisitions. This method can be split into two parts: the Color Registration (Section 6.1), where the process is described per-image and each image colorizes a portion of the point cloud, and the Color Fusion (Section 6.2), where all the colorized point cloud partials are merged into the final colorized point cloud. Also, the pixel registration relies on a camera calibration, both the intrinsic calibration and also the extrinsic, so two methods are shown to obtain this calibration (Sections 6.3 and 6.4).

6.1 Color Registration

This method describes how to colorize a point cloud based on a single image, using a working principle similar to the ray tracing used in computer graphics. As an overview, each point in the point cloud can be transformed as a ray in the camera perspective, which is basically the path from the eye point to the point. This ray can be used to retrieve the original color of the point from the image. However, this process is not so straightforward, because the position and orientation of the camera has to be very precise and occluded points need to be rejected.

6.1.1 Point to pixel coordinates transformation

To start, each point has to be transformed, because the original point is registered in the scene coordinate frame (p_{scene}) and has to be registered into the camera coordinate frame (p_{camera}). So, the transformations ${}^{acquisition}_{scene}\mathcal{T}$, ${}^{ptu}_{acquisition}\mathcal{T}$, ${}^{camera}_{ptu}\mathcal{T}$ can be used according to Equation (6.1). The ${}^{acquisition}_{scene}\mathcal{T}$ transformation is obtained in Section 5.4, ${}^{ptu}_{acquisition}\mathcal{T}$ is the transformation of the PTU and ${}^{camera}_{ptu}\mathcal{T}$ is the extrinsic calibration of the camera and the method to obtain it is in Section 6.4. The transformation graph can be seen in Figure 6.1.

$$p_{scene} = {}^{acquisition}_{scene}\mathcal{T} \cdot {}^{ptu}_{acquisition}\mathcal{T} \cdot {}^{camera}_{ptu}\mathcal{T} \cdot p_{camera} \tag{6.1}$$

Next, each point was transformed into pixel coordinates (u, v), using the pinhole camera model. This model defined how a light ray in projected in the image sensor of a camera and has two parameters: the focal length $f = (f_x, f_y)$ and optical center $(c = (c_x, c_y))$. This

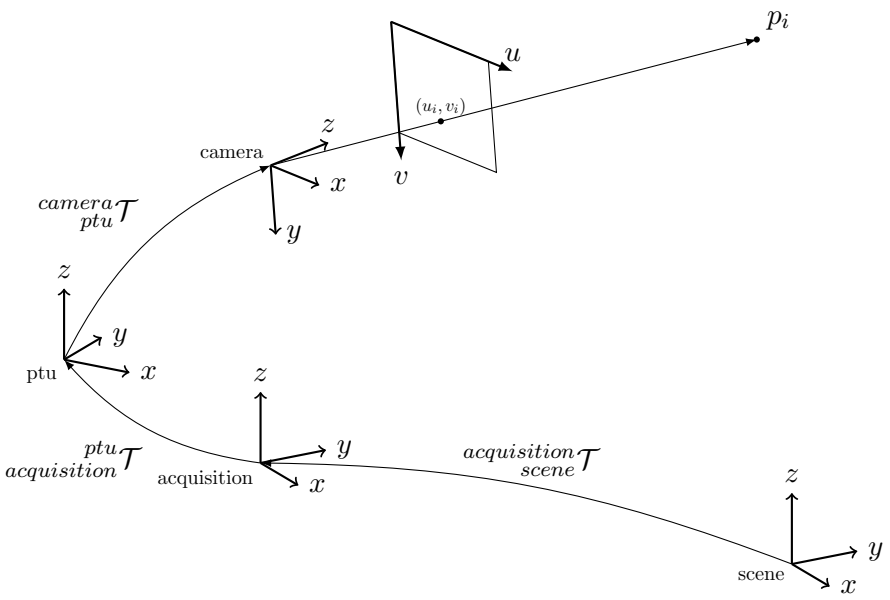


Figure 6.1: Color registration for a single point

parameters are obtained in the intrinsic calibration of the camera (Section 6.3). According to this model, each point is projected as pixel coordinates (u, v) to a plane located a unit distance from the camera eye point, using the perspective projection matrix in Equation (6.2), according to Equation (6.3).

$$\mathcal{P} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.2}$$

$$\begin{pmatrix} uz \\ vz \\ z \end{pmatrix} = \mathcal{P} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{6.3}$$

6.1.2 Camera Distortion

The pinhole camera model does not regard the distortion caused by the lens, which is not negligible for most cameras. The two sources of distortion are radial and tangential distortion. Radial distortion makes straight lines appear curved, known as the barrel distortion and pincushion distortion. This distortion is highly noticed in images taken with fish-eye lenses, as seen in Figure 6.2. This distortion can be solved by transforming the (u, v) with Equation (6.4). Similarly, tangential distortion is caused by a misalignment of the lens to the imaging plane, which causes areas in the image to appear closer than expected. This deformation can be solved with the Equation (6.5). In brief, to undistort the image five parameters need to be determined, also known as the distortion coefficients: $\{k_1, k_2, p_1, p_2, k_3\}$, which are obtained in the camera intrinsic calibration method, described in Section 6.3.

$$\begin{pmatrix} u \\ v \end{pmatrix}_{calibrated} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{pmatrix} u \\ v \end{pmatrix} \tag{6.4}$$

$$\begin{pmatrix} u \\ v \end{pmatrix}_{calibrated} = \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} 2p_1 uv + p_2(r^2 + 2u^2) \\ p_2(r^2 + 2v^2) + 2p_2 uv \end{pmatrix} \tag{6.5}$$

Figure 6.2: Barrel distortion in fish eye lens

6.1.3 Point filtering

Not all points are eligible for the color registration, based on it's location and camera properties, so two filtering steps were used: the first filter removes the points outside the view frustum and the second removes the hidden points.

View Frustrum Removal Filter

The view frustum is defined as the region of space that is captured by the camera sensor, which for pinhole cameras is a pyramid truncated by two parallel planes (the near and far clipping planes), as seen in Figure 6.3. The sides of the frustum are limited by the size of the sensor, so the points that lie outside the bounding box defined by the points $(0, 0)$ and $(width, height)$ is excluded.

The near and far clipping planes should reject the points based on the depth of field of the camera. The depth of field, or DOF, is the distance from the camera to the objects range, so that this objects are in focus. If the object falls out of this range, it starts to loose focus incrementally the farther out it is. There is a point where the object is sharper, which is called point of optimal focus. Two factor influences the DOP: the aperture size and the focal length. The aperture influences the amplitude of the DOF, so a bigger aperture results in a narrower DOF, and the focal length defines the point of optimal focus, so a bigger focal

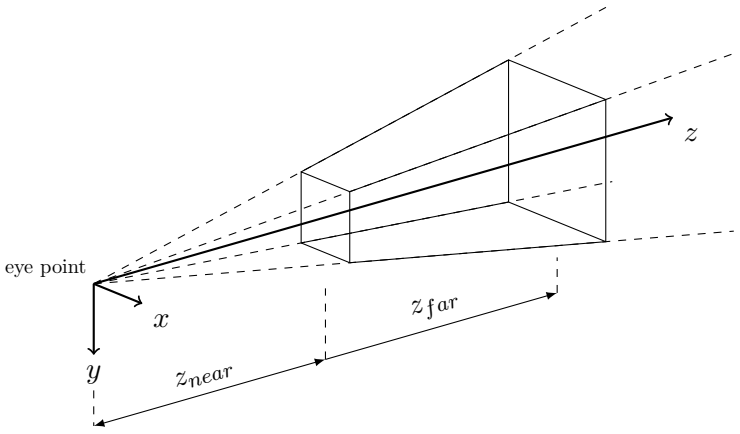


Figure 6.3: Representation of the visual frustum of the camera

length moves the DOF farther away from the camera. The near and far clipping plane should defined to be the boundaries of the DOF, so that all points are in focus.

Based on the frustum of the camera defined above, the points that do not respect Equation (6.6) should be rejected.

$$\begin{aligned} 0 \quad &< u < width \\ \wedge\ 0 \quad &< v < height \\ \wedge\ z_{near} \quad &< z < z_{far} \end{aligned} \tag{6.6}$$

Hidden Point Removal Filter

Not all points that lie on the frustum of the camera are seen by the camera, because some of this points are occluded by nearer objects, so they need to be removed. A fast and straightforward solution is to use the point cloud resulting from the same acquisition as the image, because the sensor are considered close together. However, this is not the best solution, as it would be better if the whole point cloud was used.

In [10], an simple and fast operator, the Hidden Point Removal, or HPR, determines the visibility of point sets, viewed from a given viewport. This method is easily implemented and has a asymptotic complexity of $O(n \log n)$, where n is the number of points in the point cloud. Moreover, this method work well for both sparse and dense point clouds.

The HPR operator operates on a set of points $\mathcal{P} = \{p_i | i = 1 \dots n\}$, and the goal is to determine whether p_i is visible from a viewpoint $\mathcal{C}$. In this application, C is the origin of the point cloud. The algorithm consists of two steps: the inversion and the convex hull construction.

The inversion step maps each point p_i along the ray from $\mathcal{C}$ to p_i, such that $|p_i|$ is monotonically decreasing. There are multiple ways to perform the inversion, but in [10] the *spherical flipping* was used. Spherical flipping reflects a point p_i with respect to a sphere of radius R to the new point $\hat{p_i}$ by applying the Equation (6.7).

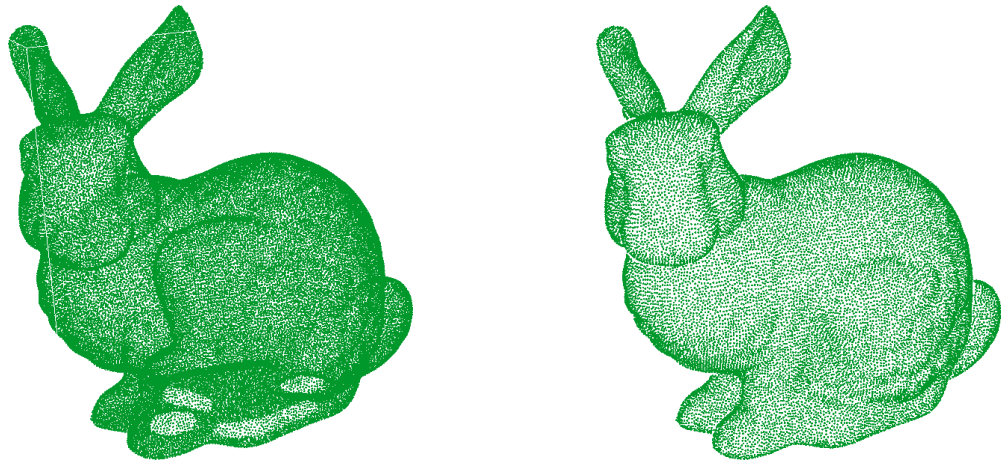
$$\hat{p_i} = p_i + 2(R - |p_i|)\frac{p_i}{|p_i|} \tag{6.7}$$

Afterwards, the convex hull of $\hat{\mathcal{P}} \bigcup \{\mathcal{C}\}$, where $\hat{\mathcal{P}}$ is the transformed point set and $\mathcal{C}$ is the

center of the sphere, is computed. Finally, the points that lie in the complex hull are the visible points of the point set.

This algorithm only has a parameter, which is the radius R of the sphere used for the spherical flipping, which influences the amount of false positives of the algorithm. In general, R is determined based on the maximum point length $max(|p_i|)$ and a exponential factor α, such that $R = max(|p_i|) \times 10^{\alpha}$. In this application, a factor of $\alpha = 3$ was adequate.

As an example, the HPR operator was used in the Stanford Bunny point cloud, as seen in Figure 6.4 and, as seen, Figure 6.4b only presents the points that are visible, as opposed to Figure 6.4a.



(a) Before the HPR (b) After the HPR

Figure 6.4: Result of the HPR operator in the Bunny point cloud

6.1.4 Color Attribution

Finally, the color is extracted from the image at the pixel coordinates (u, v) and saved for the correspondent pixel. Because images are discrete, the color is interpolated using a bilinear interpolation, which uses the neighbor pixels to interpolate the color C at (u, v) in an image I according to Equation (6.8) (the ceil and floor operators are, respectively, $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$). The interpolation can be visualized in Figure 6.5.

$$\begin{aligned} C(u,v) = &(u - \lceil u \rceil)\,(v - \lceil v \rceil)\; I_{\lfloor u \rfloor, \lfloor v \rfloor} \\ &+ (u - \lceil u \rceil)\,(v - \lfloor v \rfloor)\; I_{\lfloor u \rfloor, \lceil v \rceil} \\ &+ (u - \lfloor u \rfloor)\,(v - \lceil v \rceil)\; I_{\lceil u \rceil, \lfloor v \rfloor} \\ &+ (u - \lfloor u \rfloor)\,(v - \lfloor v \rfloor)\; I_{\lceil u \rceil, \lceil v \rceil} \end{aligned} \tag{6.8}$$

6.2 Color Fusion

In an capture with N_a acquisitions, each one with N_i images, the total number of images account to $N_a \times N_i$. Each one of this images will yield a partial colorized point cloud, according to Section 6.1, and the point clouds need to be merged into a final point cloud.

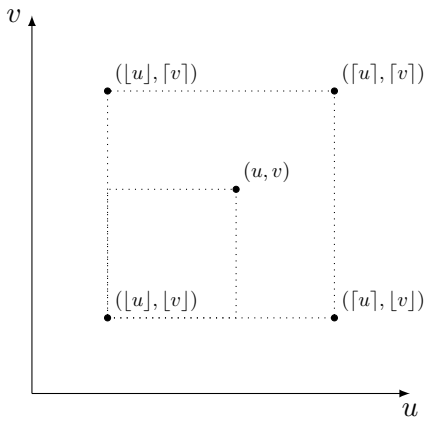


Figure 6.5: Bilinear interpolation in an image

More specifically, each point p_i has multiple correspondent colors, one for each registered image. The method here described determines the final color in a point-wise fashion and does not account for the neighbor points.

Let us admit that the point p has a set $C = \{c_i | i = 1 \dots k\}$ of k registered colors. The final color of this point c should be a combination of the colors in C.

The first approach is to colorize the point with one image only, for example, the first or last image. This method is the easiest and the faster, but does not consider the other images for the colorization.

The second approach is to average the colors to obtain the color c, as seen in Equation (6.9). However, this is a poor heuristic as it considers that all colors have the same error, which is not true. For example, an image taken closer to an object is more precise than one taken away from it.

$$c = \frac{1}{k} \sum_{i}^{k} c_i \tag{6.9}$$

A common solution for the mean limitation is to use an weighted mean, shown in Equation (6.10). The w_i are the weights for each color and should reflect the quality of each color, because colors with bigger weight have a bigger influence in the final color.

$$c = \frac{\sum_i^k w_i c_i}{\sum_i^k w_i} \tag{6.10}$$

In this work, the quality measurement was determined based on an heuristic that depends on two factors, that are obtained in the color registration phase (Section 6.1).

The first factor f_1 depends on the distance d from the camera to the point and on the optimal focus point d_f. f_1 is smaller the bigger the distance between d and d_f. The function used was the gaussian centered on d_f. The second factor f_2 depends on the distance from the pixel coordinates (u, v) to the center of the optical center (c_x, c_y). Again, a gaussian distribution was used to calculate f_2, and a bigger distance also yields a smaller f_2. In brief, both factors f_1 and f_2 are calculated according to Equations (6.11) and (6.12). The parameters α and β determine how wide the gaussian function is, so points farther from the peak point influence more or less.

$$f_1 = \exp\left(-\frac{(d-d_f)^2}{2\alpha^2}\right) \tag{6.11}$$

$$f_2 = \exp\left(-\frac{(u-c_x)^2+(v-c_y)^2}{2\beta^2}\right) \tag{6.12}$$

$$\tag{6.13}$$

The two factors are then combined into the weight w factor of the color, based on a linear combination, dependent on a parameter s, which determines the influence of each factor, as seen on Equation (6.14).

$$w = sf_1 + (1-s)f_2 \tag{6.14}$$

In conclusion, for each point p_i the color c_i is attributed, based on the registered colors of each image. The fusion of all this colors is based on a weighted mean, where the weight of each color is determined by an heuristic that considers the location of the color in pixel coordinates and the distance of the point to the camera, in order to benefit points that have a better quality in the measurement, for example, points that are in focus or points that are closer to the camera center. This process is repeated for all the points of the point cloud until every point has a color (however, some points have no color registered, because no color was registered before).

6.3 Camera Intrinsic Calibration

The intrinsic calibration determines the intrinsic parameters of the camera, such as the focal lenght ($f = (f_x, f_y)$), the optical center ($c = (c_x, c_y)$) and the distortion coefficients to model the lens distortion. This parameters are required to create the distortion matrix, as well as the undistortion function, which is essencial for the point projection (as described in **??**).

The calibration procedure used in this work is a standard procedure for cameras with low distortion and is known as the chessboard camera calibration. This method calibrates a monocular camera with fixed focus using a sequence of images taken from a chessboard with known dimensions. In order to improve the calibration results, the chessboard should rotate and move, in order to occupy the entire image size.

After all the images are obtained, the corners of the chessboard are extracted and the re-projection error is minimized to obtain the the intrinsic parameters. The results of this calibration are more accurate if the corners of the chessboard are well defined in the image, so the chessboard should have an appropriate size. Also, the chessboard poses should be enough and should be well distributed spatially.

In the end, the accuracy of the calibration should be measured for new images, with the re-projection error. This value should be as low as possible and, as a rule of thumb, a value less than 0.01 is acceptable.

In ROS, this calibration is easily obtained with the *cameracalibrator.py*, which includes a graphical interface, and provides feedback about the corner detection and the state of the calibration. The interface is shown on Figure 6.6. In this system, this data is first saved into a ROS *camera_info* file. Then, this file is also saved in each capture in the parameters file.

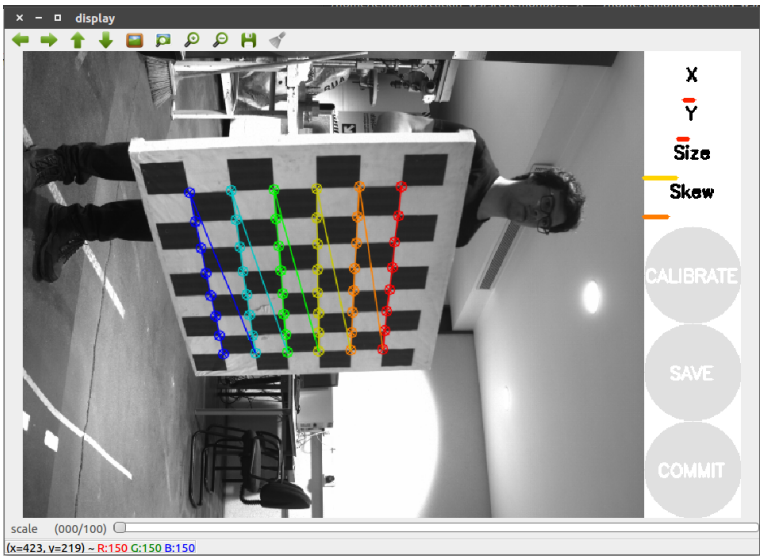


Figure 6.6: Interface for the *cameracalibrator* node

6.4 Camera Extrinsic Calibration

The extrinsic parameters of the camera the position and orientation of the camera in the robot. In this case, the camera was mounted statically on top of the PTU, just like the laser scanner. This calibration that determines this extrinsic parameters is known as the eye-in-hand calibration, described in [7].

This calibration relies on a static calibration object, whose pose can be estimated in the camera frame. Hence, four coordinate frames and four transformation exist. The four frames are the *camera* frame, the *world* frame, the *PTU* frame and the *object*. The four transformations are the extrinsic transformation of the camera, or the *PTU* to the *camera* transformation ${}^{camera}_{ptu}\mathcal{T}$, which is static and unknown, the *camera* to *object* transformation ${}^{object}_{camera}\mathcal{T}$, which is obtained by the object pose estimation algorithm, the *world* to *PTU*, which is known and, finally, the *world* to *object* transformation, which is static and unknown. The overall transformation graph is shown is in Figure 6.7, with the unknown transformations in red and the known transformations in green.

The inspection of the transformation graph determines an equality, because there are two possible ways to transverse the graph from one node to another, which yields the Equation (6.15). This equality is the base of this optimization: ${}^{camera}_{ptu}\mathcal{T}$ can be obtained from multiple pairs of synchronized ${}^{object}_{world}\mathcal{T}$ and ${}^{object}_{camera}\mathcal{T}$ transformations.

$$ {}^{object}_{world}\mathcal{T} = {}^{ptu}_{world}\mathcal{T} \cdot {}^{camera}_{ptu}\mathcal{T} \cdot {}^{object}_{camera}\mathcal{T} \tag{6.15} $$

In this work, the object used for detection was an ArUco marker, which is comprised of a pattern which can be detected and also allows for precise pose estimation, as seen in Figure 6.8. One of the biggest advantages over other markers is that the implementation for detection and pose estimation is already implemented in the ROS package *aruco_detect*. The calibration is also implemented in the ROS package *visp_hand2eye_calibration*, as a node that receives multiple transformations in the topics */world_effector* and */camera_object*, which correspond respectively to the ${}^{ptu}_{world}\mathcal{T}$ and ${}^{object}_{camera}\mathcal{T}$ transformations. To publish the transfor-

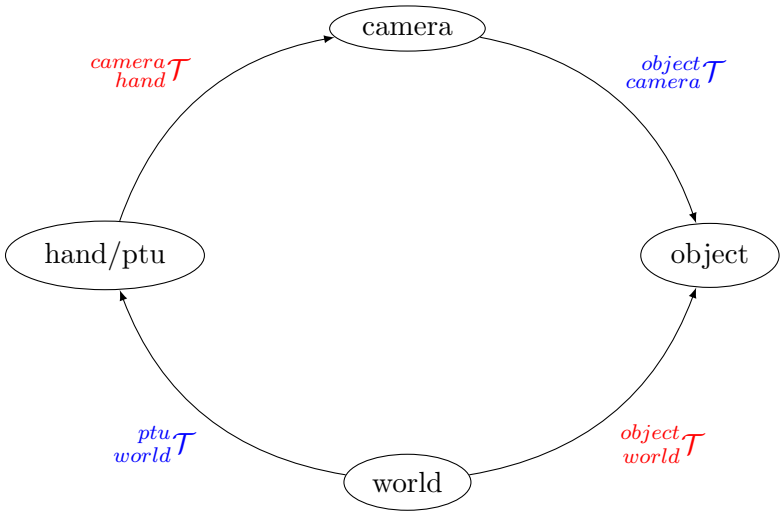


Figure 6.7: Hand-in-eye transformation graph

mations on this topics, a node was developed, the *hand2eye_simple_client*, which publishes both the transformations synchronously at the keypress of the user. The control of the PTU was also manual.

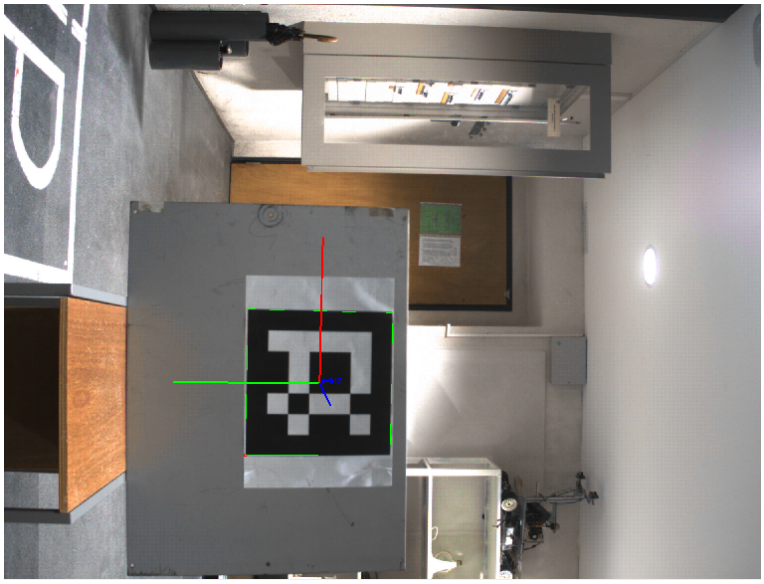


Figure 6.8: ArUco maker detection and pose estimation.

Bibliography

[1] *Building safety and security - SICK*. URL: https://www.sick.com/ag/en/industries/building-safety-and-security/c/g288283 (visited on 09/02/2018).

[2] Zi-xing Cai et al. "A 3-D perceptual method based on laser scanner for mobile robot". In: 2005 (Jan. 2005), pp. 658–663.

[3] Paulo Dias, Miguel Matos, and Vitor Santos. "3D Reconstruction of Real World Scenes Using a Low-Cost 3D Range Scanner". In: 21 (Oct. 2006), pp. 486–497.

[4] *Estimating Surface Normals in a PointCloud*. URL: http://pointclouds.org/documentation/tutorials/normal_estimation.php (visited on 09/02/2018).

[5] *FARO Focus 3D Laser Scanner*. URL: https://www.faro.com/en-gb/products/construction-bim-cim/faro-focus/ (visited on 09/02/2018).

[6] Enrique Fernandez. *Learning ROS for Robotics Programming - Second Edition*. Packt Publishing, 2015.

[7] Radu Horaud and Fadi Dornaika. "Hand-Eye Calibration". In: 14 (June 1995), pp. 195–210.

[8] Joachim Hornegger and Carlo Tomasi. "Representation Issues in the ML Estimation of Camera Motion". In: *ICCV*. 1999.

[9] "Junior: The Stanford Entry in the Urban Challenge". In: 2008.

[10] Sagi Katz, Ayellet Tal, and Ronen Basri. "Direct visibility of point sets". In: 26 (July 2007).

[11] Denis Klimentjew, N Hendrich, and Jianwei Zhang. "Multi sensor fusion of camera and 3D laser range finder for object recognition". In: (Oct. 2010), pp. 236–241.

[12] Marc Levoy et al. "The Digital Michelangelo Project: 3D Scanning of Large Statues". In: (July 2000), pp. 131–144.

[13] *Matterport 3D Space Gallery*. URL: https://matterport.com/gallery (visited on 09/02/2018).

[14] *Matterport Pennsylvania Craftsman Home*. URL: https://matterport.com/3d-space/pennsylvania-craftsman-home/ (visited on 09/02/2018).

[15] *Matterport website*. URL: https://matterport.com/ (visited on 09/02/2018).

[16] Francesco Maurelli et al. "A 3D laser scanner system for autonomous vehicle navigation". In: (July 2009), pp. 1–6.

[17] Z. Nemoto, H. Takemura, and H. Mizoguchi. "Development of Small-sized Omni-directional Laser Range Scanner and Its Application to 3D Background Difference". In: (Nov. 2007), pp. 2284–2289. ISSN: 1553-572X. DOI: 10.1109/IECON.2007.4460106.

[18] Richard A. Newcombe et al. "KinectFusion: Real-time dense surface mapping and tracking". In: *2011 10th IEEE International Symposium on Mixed and Augmented Reality* (2011), pp. 127–136.

[19] Lars Pfotzer et al. "Development and calibration of KaRoLa, a compact, high-resolution 3D laser scanner". In: *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)* (2014), pp. 1–6.

[20] *Potree examples showcase - Retz*. URL: http://www.potree.org/potree/examples/showcase/retz.html (visited on 09/02/2018).

[21] M. J. D. Powell. "An efficient method for finding the minimum of a function of several variables without calculating derivatives". In: *The Computer Journal* 7.2 (1964), pp. 155–162. DOI: 10.1093/comjnl/7.2.155. eprint: /oup/backfile/content_public/journal/comjnl/7/2/10.1093/comjnl/7.2.155/2/070155.pdf. URL: http://dx.doi.org/10.1093/comjnl/7.2.155.

[22] Jochen Schmidt and Heinrich Niemann. "Using Quaternions for Parametrizing 3-D Rotations in Unconstrained Nonlinear Optimization". In: (Jan. 2001).

[23] *Scipy reference - minimize(method='Powell')*. URL: https://docs.scipy.org/doc/scipy/reference/optimize.minimize-powell.html (visited on 09/02/2018).

[24] *Self Driving Car - Google*. URL: https://www.google.com/selfdrivingcar/ (visited on 09/02/2018).

[25] Ju Shen and Sen-ching Cheung. "Layer Depth Denoising and Completion for Structured-Light RGB-D Cameras". In: (June 2013), pp. 1187–1194.

[26] Prarinya Siritanawan, Moratuwage Diluka Prasanjith, and Danwei Wang. "3D feature points detection on sparse and non-uniform pointcloud for SLAM". In: *2017 18th International Conference on Advanced Robotics (ICAR)* (2017), pp. 112–117.

[27] Ioannis Stamos and Peter Allen. "3-D Model Construction Using Range and Image Data". In: 1 (Feb. 2000), 531–536 vol.1.

[28] *Stanford Scanning Repository*. URL: graphics.stanford.edu/data/3Dscanrep/ (visited on 09/02/2018).

[29] *Streambend Laser Scanning*. URL: https://streambend.net/laser-scanning/ (visited on 09/02/2018).

[30] Hartmut Surmann, Andreas Nuchter, and Joachim Hertzberg. "An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments". In: 45 (Dec. 2003), pp. 181–198.

[31] *The Stanford Bunny*. URL: https://www.cc.gatech.edu/~turk/bunny/bunny.html (visited on 09/02/2018).

[32] *Velodyne VLS180*. URL: https://velodynelidar.com/vls-128.html (visited on 09/02/2018).

[33] Michael Wimmer and Claus Scheiblauer. "Instant Points: Fast Rendering of Unprocessed Point Clouds". In: *SPBG*. 2006.

[34] Tomoaki Yoshida et al. “3D laser scanner with gazing ability”. In: (June 2011), pp. 3098–3103.

[35] Qilong Zhang and R. Pless. “Extrinsic calibration of a camera and laser range finder (improves camera calibration)”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. Sept. 2004, 2301–2306 vol.3. DOI: 10.1109/IROS.2004.1389752.

[36] Michael Zollhöfer et al. “State of the Art on 3D Reconstruction with RGB-D Cameras”. In: *Comput. Graph. Forum* 37 (2018), pp. 625–652.