Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digital reconstruction of three dimensional scenes is a field that gained a high importance in areas like architecture, robotics, archeology, autonomous driving and virtual reality. The process of digitalization of the environment into a 3D model still poses a lot of challenges, requiring a lot of processing and merging data from different sensors to get an accurate representation.

1.1 Objectives

Despite all the work done, there is still no perfect solution. 3D reconstruction is still a challenge, because real scenes are very complex and sensors have 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.

So, we aim to develop a 3D sensor that captures both range scans as well as color images, which is now a very recurrent and successful approach. This is because both sensors complementing each other, one providing high definition and accurate spatial resolution (Laser scanner) and the other providing the high resolution color map.

The main objectives are:

1. Development of an algorithm capable of creating a 3D colorized point cloud. It is required that all geometry and color registration be accurate and precise and the model need to have a high density of points, in order to register most of the details of the scene.

2. Creating a dataset consisting of multiple acquisitions for subsequent use by other projects in 3D processing and vision algorithms.

3. Development of an automatic and easy-configurable mobile robot for easy and reliable captures, to use for subsequent work.

What is not covered in this work is:

1. Processing and vision algorithms for 3d models. We do not include this in this work because the main objective is creating dense and accurate point clouds, not processing them. However, we hope that this work provide ways to obtain reliable data for 3D-based algorithms.

2. Generating 3D mesh models. Mesh models are better for representing a scene, and algorithms like Delaunay triangulation and Ball-pivoting triangulation algorithms allow it to easily create a mesh from a point cloud. Nonetheless, it requires huge amounts of preprocessing and post processing to get a good 3D mesh model. Yet, the 3D point clouds resulting in this work can be processed to create a 3D mesh model.

Chapter 2

Theoretical background

2.1 Technologies

Many technologies were developed to capture tridimensional information of the environment. In the following section, we aim to describe 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 remain the most popular method for 3D sensing, also because it's 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 [1]. 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 [2].

This appealing characteristics lead to a huge research and development in 3D reconstruction using this camera, culminating in the KinectFusion algorithm [3], 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. This technique, however, yields results far from perfect: the depth values from structured light have significant error or can be missing, specially from objects with darker colors, specular surfaces or small surfaces [4].

2.1.3 Time of Flight

Time of flight 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 some advantages comparing to both previous approaches [2]:

1. Is less computationally intensive, because the measurement is directly measured by a specialized sensor.
2. Is partially independent of the lightning conditions because the light detected is emitted by the device itself.
3. Is capable of a dense and accurate depth values, even for continuous or irregular surfaces, unlike the stereoscopic approach.
4. It is much faster that any other method, capable of acquisition rates of hundreds of Hz.

However, it has some disadvantages as well:

1. Unlike stereoscopic, which is a passive method, ToF sensors interact with the scene, so are not possible to be implemented in certain environments.
2. The properties of the material, like the reflectivity, color and roughness can have significant effects on the accuracy of ToF sensors.
3. Multi-path reflections are a common problem of ToF sensors, caused by multiple reflections of the light, causing errors in the measurements.
4. Interference if multiple ToF sensors share the same environment. However, it is possible to mitigate this effect.

A new popular ToF sensor today is the Photonic-Mixer-Device camera, which looks similarly to a normal image sensor measures the phase shift of incoming light. This sensor is now being used in new generation RGB-D camera, replacing the structured light approach, mainly because it is more resilient to background light, allowing the sensor to work in outdoor environments [2]. 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, it's 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 aircraft to create high-precision and dense earth's surface models. Nowadays, it's 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 it's light source. Lasers are capable of emitting beams of light that are:

1. **Monochromatic**. Lasers emit light in a narrow spectrum of light, so they can produce a single color of light. It improves the resilience against background light, making it independent to sun radiation.

2. **Narrow**. 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.

3. **Polarized**.

2.2 Related 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 we describe such systems that were developed so far.

In [5], 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 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 [6] 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 $1 million cash prize, to the fastest autonomous unmanned vehicle that completed a 300 miles track. In [7], a 3D laser scanner 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 [8], a swinging platform [9]. This sensor also became lighter, compact and modular, making it possible to integrate in multiple systems easily. One of this systems is *KaRoLa*, described in [10]. This laser scanner was then applied to several system, specially in search and rescue robots.

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 area like architecture or archeology, where color information is very important. Such work can be seen on [11], where a 3D sensor like the one described in [5] was paired with a camera, to generate a 3D reconstruction with color. Another technique was created in [12], 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 [13], 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.

2.3 Comercial Solutions

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 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 it's flexibility and wide use, graphic cards are specially optimized to render this meshes and the result 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 it 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 it's 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 it's 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:

1. 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 [14], it will always be a limiting factor for this model.

2. 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.

3. 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.

This problems are usually solved by performing a triangulation of the point cloud, a process where a mesh is produced, therefore solving all the problems inherent to the simplistic point cloud model. 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 are 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 constitutes a mobile 3D scanner called ”lemonbot”. This scanner’s core components are a 2D laser scanner and a camera mounted on top of a pan and tilt unit. The other components are a mini computer, a battery and a wireless router. All the components are assembled on a tripod, according to the [esquemático do tripé]: the ptu is connected to the top of the tripod shaft, and the laser and camera is connected to the ptu moving link; the battery, computer wireless router and ptu driver is placed on a support in the legs of the ptu; a cable harness runs through the tripod’s shaft connecting all parts to the host computer. The host computer can be remotely controlled, via a remote connection (ssh), by connecting the client device (usually a computer) through the wireless network created by the router. This is specially useful to make the repetitive task of acquisitions faster and more agile.

3.1.1 2D Laser Scanner

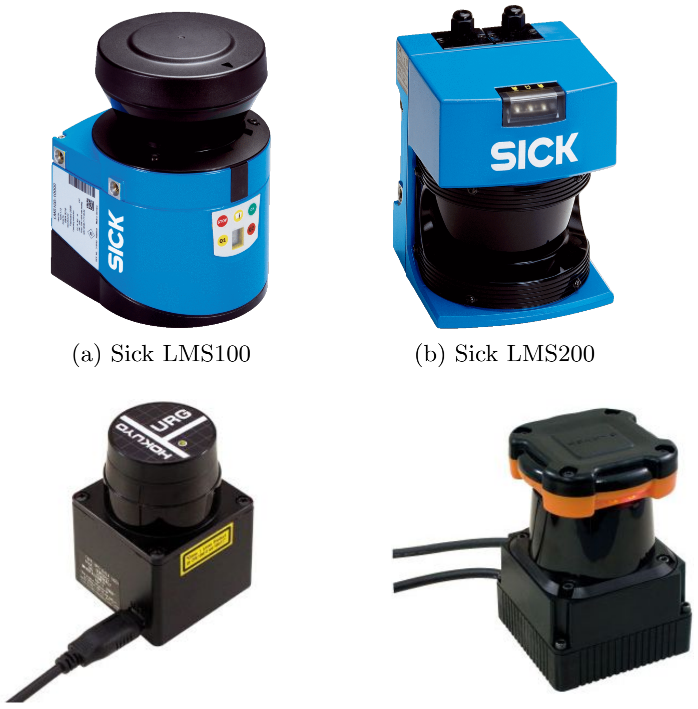
One of the objectives of this work is to try different 2D laser scanners, to try to find the one that fits best for this kind of application. We chose four 2D laser scanners, trying to get a representative sample of all the market. This laser scanners differ in different aspects, as for example price, size, weight and working range. In table 3.1, a comparison of the characteristics of this laser scanners are specified.

3.1.2 Camera

The camera used in this work was a PointGrey Flea3 FL3-GE-28S4 Camera (fig. 3.2), 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 3.3.

Table 3.1: 2D laser scanners used and their characteristics

	Sick LMS100	Sick LMS200	Hokuyo's URG04LX	Hokuyo UTM-30LX
Application	Indoor			
Safety Class	Class 1 (eye-safe)			
Light source	Infrared (905 nm)	Infrared (905 nm)	Red (785 nm)	Infrared (905 nm)
Aperture angle	270°	180°	240°	270°
Scanning Frequency	10 Hz	50 Hz	10 Hz	40 Hz
Angular Resolution	0.25°	0.25°	0.36°	0.25°
Working Range	0.5 m to 20 m	0 m to 80 m	0.02 m to 5.6 m	0.1 m to 30 m
Systematic Error	±20 mm	±15 mm	NA	NA
Statistical Error	±12 mm	±5 mm	±3 %	0.1 m to 10 m: ±30 mm; 10 m to 30 m: ±50 mm
Size				
Price	5000€			



(a) Sick LMS100 (b) Sick LMS200

(c) Hokuyo URG04LX (d) Hokuyo UTM-30LX

Figure 3.1: 2D Laser scanners used

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

Table 3.3: Characteristics of the PointGrey Flea3 FL3-GE-28S4 Camera

Figure 3.2: PointGrey Flea3 FL3-GE-28S4

3.1.3 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 (fig. 3.3), which is a compact and light module with the following characteristics:

- Small form factor,
- Maximum payload weight of 4 kg,
- $\pm 159°$ pan range and $-47°$ to $31°$ tilt range.
- 60 °/s maximum speed.
- Angular resolution of 0.0032°,
- Communication to the host via serial interface.

3.2 Software

This section presents the software developed or used in this work. The code was split into four repositories by their scope: the 3D mobile scanner runtime (section 3.2.1) and the data processing algorithms (section 3.2.2).

3.2.1 Lemonbot software

One of the key components of robots is the software and most robots require large amounts of code to function. In older robot architectures, the code was a monolith responsible for every system of the robot, however this proven to be an inefficient approach, as it required a high understanding of the overall code by the developer, created dependencies and collisions between separated parts of the code and required a high maintenance of the code. This also created a single point of failure, because if just one component failed everything failed.

This required a paradigm shift towards distributed systems. In a distributed system, each node runs isolated and interact through message passing. This systems are known to be:

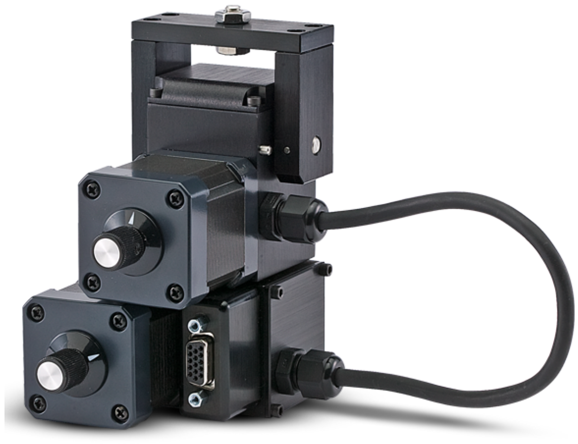


Figure 3.3: FLIR PTU-D46

- 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 development. 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 with the "real environment".
- Decoupled. Each node runs in a isolated environment, to mitigate all the conflicts that can appear in a unknown environment, like resource utilization and shared library versioning. This can go as far as running each node in a separate virtual machine, by using containers.

However, this systems now requires an underlying infrastructure, responsible for:

- Orquestration. Nodes run inside this distributed environment, so a special orquestration software is responsible to start and keep track of all the running nodes.
- Communication. Each node communicates to other components by message passing, with different topologies, like publisher-subscriber or client-server. However, this requires a standardization on the protocol of communication and message serialization.

- Discovery. Each components needs to find the location on the network of the other nodes, to be able to communicate to then.
- Configuration. Each parameter of the system is saved in a key-value store, accessible to all the nodes.

This paradigm can be seen all across computer systems and a great example of this paradigm is the Erlang language runtime system, which was designed to be distributed, fault-tolerant and highly available. This runtime system was used in the production of a high-reliable ATM switch - the Ericsson AXD301 - with achieved an outstanding 99.999 999 9 % reliability. In robotics, such a system is being developed and it's quickly becoming the new standard: the Robot Operating System.

ROS

ROS is a software architecture for robot development, providing a collection of tools, libraries and conventions to simplify the development of complex robot systems. It was originally created by the Stanford University in the mid-2000s and now is being widely adopted by most research communities.

It's design principles follow the distributed system paradigm. In ROS, each node performs a certain function and communicates by a message-passing. Messages are exchanged in topics, described by both a topic name and a topic message, and each node can advertise (to publish new messages) or subscribe (to receive new messages). The advantage of this system is that multiple nodes can publish or subscribe to the same topic, which can be very useful. Each message has a header with a *timestamp* and *frame_id*, to localize the source of the message both in space in time. A server-client architecture is also possible in ROS, but was used in this work.

This system also provides a hardware abstraction. For example, despite this work using different 2D laser scanners, the integration was really easy, because all the laser drivers publish the same message type (*sensor_msgs/Laserscan*), so all the other software communicating to this different lasers work without any change in the code.

Coordinate Frames

3.2.2 Data Processing Algorithms

Chapter 4

Laser Extrinsic Calibration

In order for this system to work, it requires the position and rotation of the laser to be known, in order to transform the laser scans into a static referencial, also known in robotics as the base link. This transformation, in this sustem is between the mount link of the pan-tilt (*lemonbot_ptu_mount_link*) to the laser optical sensor (*lemonbot_laser_optical_link*).

This transformation needs to be known to a high degree of accuracy, because any small error scales with the distance of the target to the object. For example, an error of $0.1°$ in the rotation of laser translates, in a target at 1 m, into an absolute error $\epsilon = 1\,\text{m} \times \tan 0.1° = 17\,\text{mm}$, which is already significant. So, the calibration method used in this system needs to be very accurate, in order to produce viable results.

4.1 Known Methods

Methods to calibrate a laser mounted in a moving frame were not found, so initially we used a combination of two methods, by using a camera between: the *Hand2Eye Calibration*, which calibrates a camera mounting into a moving frame, and the *RADLOCC Calibration*, which calibrates a laser to a camera. So the calibration required the following steps:

1. Firstly, the intrinsic parameters of the camera are found, using the well-known chess-board calibration method. (Missing Reference)

2. Then, the extrinsic calibration of the camera to the PTU is found using the *Hand2Eye Calibration*. (Missing Reference)

3. After, the camera-laser extrinsic calibration was found using the *RADLOCC Calibration*. (Also Missing Reference).

4. Lastly, this two extrinsic transformation given by the extrinsic calibrations are merged into one: the ptu-laser transformation.

This method was however not successful in finding an accurate calibration, as can be seen in (Insert a figure with the not so good calibrations). Even so, we used it as an initial guess and to compare it to our proposed method. The reasons this method fail are suspected to be the following:

1. (enumerate the reasons....)

4.2 Proposed Method

The method hereby shown relies on that a bad calibration does not produce a perfect geometry, in particular, points that belong to a plane, with a good calibration, should lie close to the corresponding plane formed by them. So this method relies on a minimization of a function that has as its inputs the parameterized transformation and as it's output, the score of the heuristic said above. Therefore, the lower the function value, the better the calibration, so we will try to find a minimum for this function.

The optimization function is composed of the following steps, which are explained in more detail afterwards.

- A point cloud is generated, using the input parameters to construct the ptu-laser transformation.
- A plane segmentation is performed, creating an array of point clouds, each one representing each plane. This segmentation is one time before hand manually and each subsequent segmentation follows this one as the ground truth. (Insert figure with the segmentation)
- Each cluster of points is then processed in a plane-fitting algorithm that outputs the normalized square distance of the points to the corresponding plane, also known as the mean square error (MSE).
- Then, an heuristic function combines the MSE of all the point clouds into a single value, that should reflect how good the transformation is.

4.2.1 Parameterization

The parameters required for the optimization functions should be the ptu-laser transformation. A transformation is composed by a translation and a rotation. Since a rotation matrix is composed by 9 elements to express 3 degrees of freedom other parameterization methods are used: Euler angles, axis/angle or quaternions. Quaternions are the standard representation, mainly because of its algebra, which is makes it easy to interpolate and create smooth movements, unlike Euler angles.

For the parameterization, we choose to use the axis/angle representation, because:

- It is a *fair parameterization*, meaning that it does not introduce more numerical sensibility to the problem than what is inherent to it.
- It has only 3 elements to express the 3 degrees of freedom, unlike quaternions, which have 4 elements.
- It does not contain any singularities, like the *gimbal lock* in Euler angles.

Axis/angle representation

Any rotation R can be expressed as a rotation around an axis $a \in IR^3$ by an angle θ. Since only the direction of a matter, both θ and a can be combined into a vector ω, such that:

$$\theta = |\omega|, \qquad a = \frac{\omega}{|\omega|}. \tag{4.1}$$

Because we use the quaternion representation for every rotation, this representation is converted into a quaternion q for the subsequent steps and back, using eqs. (4.2) and (4.3):

$$q = (\cos\frac{\theta}{2}, \sin\frac{\theta}{2} \cdot a) \tag{4.2}$$

$$\begin{cases} \theta = \arccos(q_w) \\ a = \arcsin(q_w) \cdot (q_x, q_y, q_z) \end{cases} \tag{4.3}$$

Conclusion

In conclusion, the input parameters are going to contain 6 elements, containing 3 elements referent to the translation $r = (r_x, r_y, r_z)$ and 3 elements referent to the rotation $\omega = (\omega_x, \omega_y, \omega_z)$, using a angle/axis representation. The heuristic is going to be expressed as $h = f(r_x, r_y, r_z, \omega_x, \omega_y, \omega_z)$.

4.2.2 Plane Segmentation

The plane segmentation step was made manually in

4.2.3 Plane Fitting

4.2.4 Heuristic

4.2.5 Optimization Method

4.3 Data

4.4 Results

4.5 Conclusions

Chapter 5

Results

Chapter 6

Conclusions

Bibliography

[1] Denis Klimentjew, N Hendrich, and Jianwei Zhang. Multi sensor fusion of camera and 3d laser range finder for object recognition. pages 236 – 241, 10 2010.

[2] Michael Zollhöfer, Patrick Stotko, Andreas Görlitz, Christian Theobalt, Matthias Nießner, Reinhard Klein, and Andreas Kolb. State of the art on 3d reconstruction with rgb-d cameras. *Comput. Graph. Forum*, 37:625–652, 2018.

[3] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

[4] Ju Shen and Sen-ching Cheung. Layer depth denoising and completion for structured-light rgb-d cameras. pages 1187–1194, 06 2013.

[5] 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. 45:181–198, 12 2003.

[6] Zi-xing Cai, Jin-xia Yu, Xiao-bing Zou, and Zhuo-hua Duan. A 3-d perceptual method based on laser scanner for mobile robot. 2005:658 – 663, 01 2005.

[7] Francesco Maurelli, David Droeschel, Thomas Wisspeintner, Stefan May, and Hartmut Surmann. A 3d laser scanner system for autonomous vehicle navigation. pages 1 – 6, 07 2009.

[8] Z. Nemoto, H. Takemura, and H. Mizoguchi. Development of small-sized omni-directional laser range scanner and its application to 3d background difference. pages 2284–2289, Nov 2007.

[9] Tomoaki Yoshida, Kiyoshi Irie, Eiji Koyanagi, and Masahiro Tomono. 3d laser scanner with gazing ability. pages 3098 – 3103, 06 2011.

[10] Lars Pfotzer, Jan Oberländer, Arne Rönnau, and Rüdiger Dillmann. Development and calibration of karola, a compact, high-resolution 3d laser scanner. *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, pages 1–6, 2014.

[11] Paulo Dias, Miguel Matos, and Vitor Santos. 3d reconstruction of real world scenes using a low-cost 3d range scanner. 21:486–497, 10 2006.

[12] Ioannis Stamos and Peter Allen. 3-d model construction using range and image data. 1:531 – 536 vol.1, 02 2000.

[13] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The Digital Michelangelo Project: 3D scanning of large statues. pages 131–144, July 2000.

[14] Michael Wimmer and Claus Scheiblauer. Instant points: Fast rendering of unprocessed point clouds. In *SPBG*, 2006.