Contents

List of Figures

List of Tables

Chapter 1

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 1.1 and all the software implemented in this robot is described in Section 1.2.

1.1 Hardware

The hardware used in this work was a mobile 3D scanner called "lemonbot", shown in Figure 1.1. 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 1.1, all three laser scanners can be compared and in Figure 1.2 are shown.

Camera

The camera used in this work was a PointGrey Flea3 FL3-GE-28S4 Camera (Figure 1.3), 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 1.2.

Figure 1.1: Lemonbot mobile 3D scanner

Table 1.1: 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 1.2: 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 1.4), which is a compact and light module with the characteristics in Table 1.3.

1.2 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 1.2.1. The other one is the software used to process the data recorded by the 3D scanner, which is described in Section 1.2.2.

Figure 1.3: PointGrey Flea3 FL3-GE-28S4

Table 1.2: 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	$29\,\text{mm} \times 29\,\text{mm} \times 30\,\text{mm}$

1.2.1 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 1.3: 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

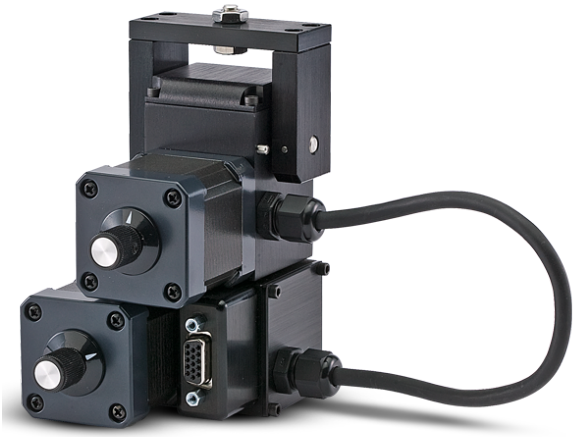


Figure 1.4: FLIR PTU-D46

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.

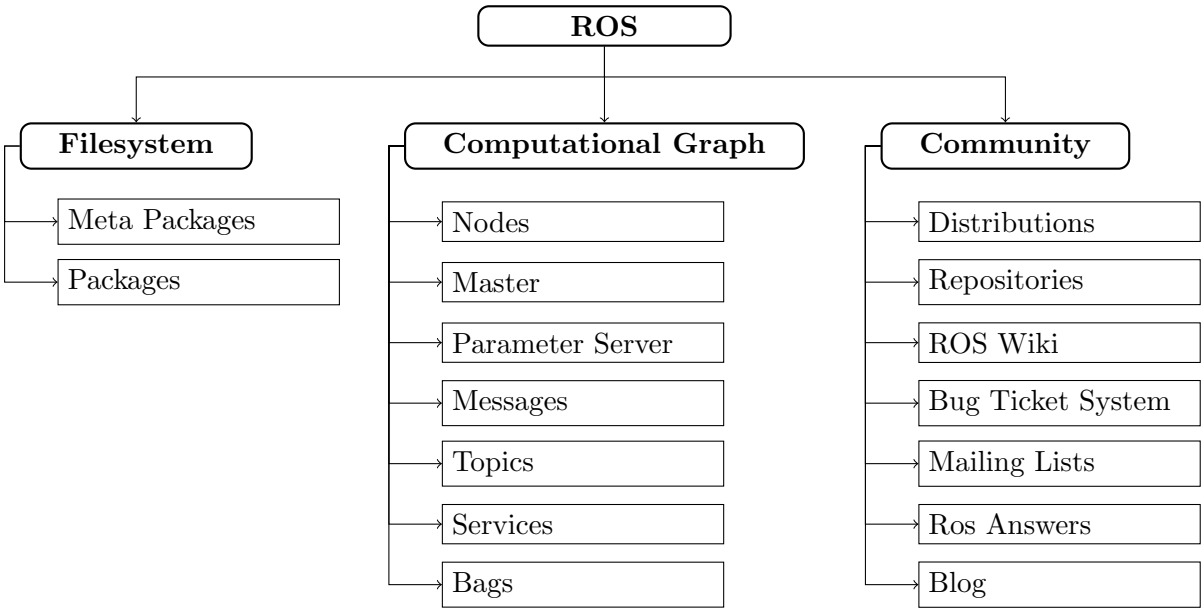
Easier to develop . Each component can be developed be separate developers, with different languages and independent release cycles, because they do not have any dependency between each over and only a message specification needs to be agreed on between developer teams, making collaborative development 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 [1]: 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 1.2.1. Some components that were required for this work are explained next.

Figure 1.5: 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.

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.

1.2.2 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 1.6.

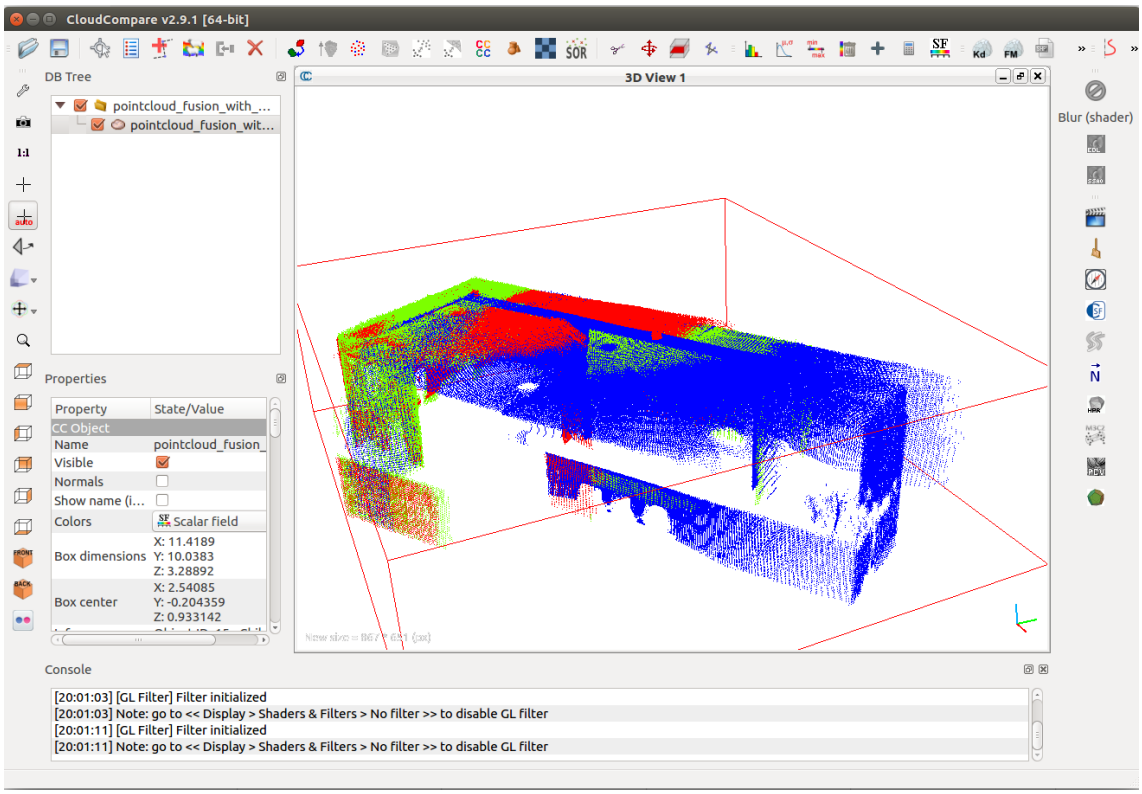


Figure 1.6: Cloud Compare screenshot.

1.3 System Integration

The section describes the principal steps of the implementation of the mobile robot using the ROS framework. The principal components of the robot are, as said, the

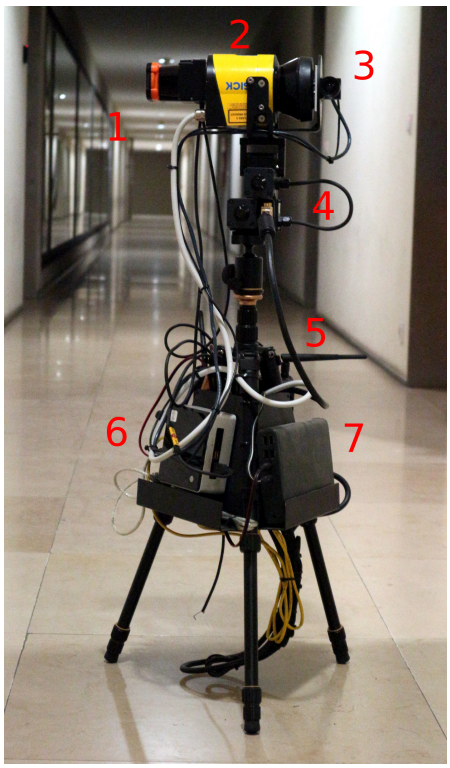


Figure 1.7: Principal Components of the Lemonbot.

Bibliography

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