Universidade de Aveiro Departamento de Engenharia Mecânica
2019

Afonso de Jesus Cardoso Pinheiro de Castro

Multi-Modal Sensor Calibration on board the ATLASCAR2

Calibração Multi-Modal de Sensores a bordo do ATLASCAR2

Universidade de Aveiro Departamento de Engenharia Mecânica
2019

Afonso de Jesus Cardoso Pinheiro de Castro

Multi-Modal Sensor Calibration on board the ATLASCAR2

Calibração Multi-Modal de Sensores a bordo do ATLASCAR2

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica do Doutor Miguel Riem de Oliveira, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro.

o júri / the jury

presidente / president

Vítor Manuel Ferreira dos Santos
Professor Associado C/ Agregação do Departamento de Engenharia Mecânica da Universidade de Aveiro

vogais / examiners committee

Artur José Carneiro Pereira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (arguente)

Miguel Armando Riem de Oliveira
Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Quero, desde já, deixar uma palavra de agradecimento genuíno ao meu orientador, Professor Miguel Riem de Oliveira, que me devolveu a paixão e o interesse pela busca, investigação e pelo consequente desenvolvimento. Que, "apenas" com a sua experiência e postura, me transmitiu valores preponderantes para a notória evolução na minha metologia de trabalho, na minha reação a obstáculos encontrados, na minha gestão de tempo, e, sobretudo, na minha visão relativamente ao futuro próximo, assunto que tantas vezes habita na mente de um estudante nesta fase do percurso académico.

Também aos meus amigos João, Danny e Diogo, que nesta intensa jornada sempre me potenciaram e reencaminharam todas as vezes que foram necessárias. A vocês devo muito do que sou e do que alcancei. Um sincero agradecimento, não só pelos contextos críticos solucionados pela vossa ajuda, mas principalmente pela contínua presença e partilha que consequenciou num crescimento valioso para o meu futuro. No meio do mar de experiências que com alta frequência lotam o quotidiano, a segurança existencial que o vosso companheirismo suporta tornou-se essencial em todos os momentos. Obrigado pelo sentido humano que sempre vos caracterizou e me ensinou.

Por fim, e por ser a base de tudo, um agradecimento à minha família. Ao meu pai pelo excelente exemplo e pela referência que é em muitas decisões que tomo. À minha mãe, pelo carinho demonstrado dos mais diversos modos e por toda a preocupação. A todos os meus irmãos, por todas as peripécias que, desde sempre, completaram a minha pessoa. Reconheço a sorte que tenho por todas as oportunidades que a minha família me permite ter, que, para tantas outras pessoas, são oportunidades muito difíceis de alcançar.

palavras-chave Calibração Extrínseca; Sensores Multi-Modais; ROS; Optimização; Veículos Inteligentes

resumo Os mais complexos sistemas robóticos possuem vários sensors de diferentes modalidades. Com o objetivo de se estimar a posição e orientação destes vários sensors multi-modais, existem alguns trabalhos que propõem calibrações sequenciais par a par: calibrações essas com alguns problemas inerentes. ATLASCAR2 é um veículo inteligente com vários sensores de diferentes modalidades. O objetivo principal deste projeto é calibrar todos os sensors a bordo do ATLASCAR2. Foi desenvolvida uma abordagem interativa e semi-automática, que funciona para qualquer robô em ROS, mesmo os mais complexos. Depois da etapa de identificação de quais as transformações geométricas, de entre toda a descrição do robô, devem ser estimadas e da coleção da informação recolhida por cada sensor, optimiza-se, através do método dos mínimos quadrados, os parâmetros de posição e orientação de cada um dos sensores do robô. Os resultados mostram que a calibração simultânea dos quatro sensores é tão boa quanto os procedimentos par a par usados pelas ferramentas de calibração padrão, como as do OpenCV. Assim sendo, a solução proposta apresenta uma nova e vantajosa metodologia, uma vez que se adequa a qualquer sistema robótico complexo e que calibra todos os seus sensors ao mesmo tempo.

keywords Extrinsic Calibration; Multi-Modal Sensors; ROS; Optimization; Intelligent Vehicles

abstract Complex robot systems have several sensors with different modalities.
In order to estimate the pose of these various multi-modal sensors, some works propose sequential pairwise calibrations, which have some inherent problems. ATLASCAR2 is an intelligent vehicle with several sensors of different modalities. The main goal of this work is to calibrate all sensors on board the ATLASCAR2. A ROS based interactive and semi-automatic approach, that works for any robot system, even the most complex ones, was developed. After the step of identifying which geometric transformations, between all robot description, should be estimated and collecting the detected data from each sensor, a least-squares optimization occurs to enhance the position and orientation of each one of the robot sensors. Results show that the four sensors simultaneous calibration is as good as the pairwise procedures used with the standard calibration tools, such as the OpenCV ones. In that way, the proposed solution brings a novel and advantageous methodology, since it fits to any complex robot system and calibrates all sensors at the same time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robot development is one of the most researched fields in today's world. With this evolution, more and more jobs can be performed by robots, turning many processes automatic, quicker and more efficient. In fact, the Fourth Industrial Revolution is being driven by technological breakthroughs in fields such as quantum computing, biotechnology, autonomous vehicles, artificial intelligence, and robotics. All these boosts will transform the way people interact with machines. According to the IFR (International Federation of Robotics), the World Robotics Report from 2017 shows that there was an increase of 30 per cent in robot units shipped globally compared with the year before (2016). More than that, the annual sales volume of industrial robots increased by 114 percent between the years of 2013 and 2017.

With all this interest of industry, robotic research communities are getting bigger and more advanced, always looking for new features and testing new solutions. Robots evolve with many cutting-edge technologies. Some of these are vision recognition, skill learning, failure prediction, new concept of man-machine-collaboration and so on. All these concepts will help to expand the field of robot application.

Calibration is an essential procedure to ensure a well and successfully performed task by any robot. Calibration results can lead to significant accuracy improvement and cost-saving for every company or particular robot user. Because of all of this, robot calibration plays an increasingly important role in robot implementation and operation.

Since robots are similar to other mechanical devices, they can be affected by slight changes or drifts caused by the usage or component replacement. Calibration procedures enhance the position and orientation (pose) of the calibrated robot components. Thus, it can minimize the risk of having to change applications. According to [7], robot calibration is a process by which robot positioning accuracy can be improved by modifying the robot positioning software instead of changing the mechanical structure or design of the robot itself. To fulfill this, the procedure involves developing a model whose parameters accurately represent the real robot.

There are some robots, specially in industry scenarios, which perform successive jobs repeatedly, without having information about what is happening on the surroundings. On the other hand, robots with sensor components only operate after processing the data received by the sensors. That means that the robots actions depend on environmental events.

The majority of robotic research relies on the relation between pose of the sensors and the detected information with the body and the actions to execute. That is why the works in the field of robotic of today and the near future always approach the sensor role. The robots that can actually understand the environment, and act accordingly, are known as Intelligent Robots. In order to ensure that an intelligent robot can be successful in its task, it is imperative to have the sensors calibrated as well as possible. In other words, the robot must know exactly where each one of its own sensors is with respect to the others. That is the only way for the robot to identify the real and correct pose of every detected object by the sensors.

In the first decade of the twenty-first century, many research communities developed different calibration methods, for distinct types of sensors. Several calibration procedures are already available, as we will see in the next chapter, but only for the case of a robot with only two sensors (pairwise calibration).

The creation of robots capable of performing more complex tasks, with more variables, of diverse nature, is growing nowadays. The aim is to get robots to do human-like tasks, like driving a car, for example. These more complex type of robots, obviously, have more than just two sensors acquiring information. Thus, it is truly important to have a procedure that could calibrate all sensors of a complex robot at once. This idealization poses some inherent problems.

1.1 Problem Definition

Sensor calibration plays an essential role in intelligent robot development. With the purpose of understanding the problems associated to multi-sensor calibration, it is important to, first of all, perceive what calibration means.

A robot system is composed with links and joints, where each joint connects two different links. Links, usually, represent physical robot body components, and each one has its own coordinate frame. A joint symbolizes a geometric transformation between two link frames. Commonly, robots have a reference frame where each chain of transformations (like for a robot arm) starts. Figure 1.1 [1] shows a conceptual example of a small robot constitution. All links and joints could be represented in a transformation graph. In this transformation graph, nodes are reference frames, represented by ellipses, and the arrows (edges) symbolize the transformations between reference frames.

[1] `http://wiki.ros.org/urdf/XML/model`

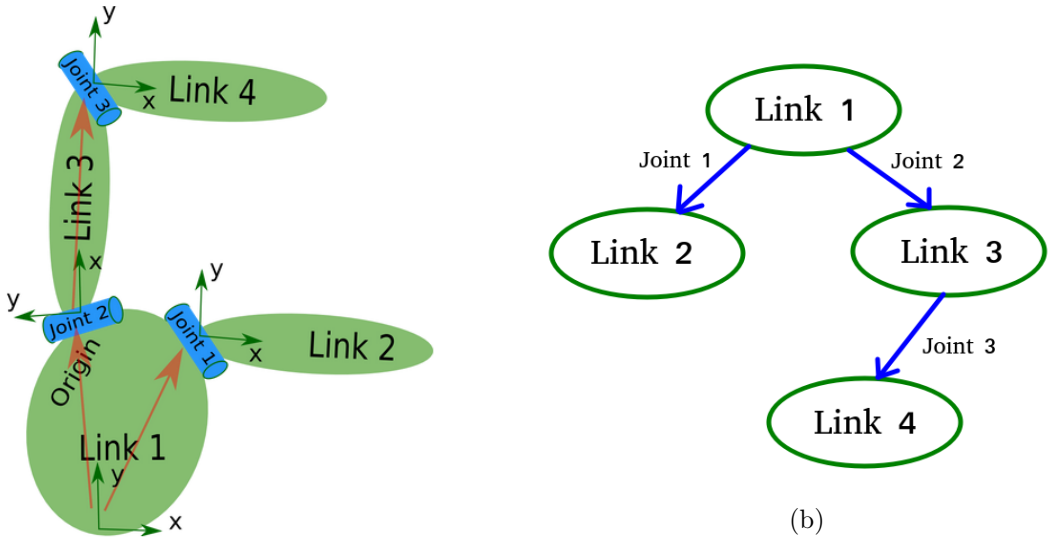


Figure 1.1: Conceptual robot example: a) links and joints that constitute the robot structure; b) associated transformation graph.

Sensor elements are normal links, that are related to the remainder of the robot links. The goal of sensor calibration procedure is to estimate the pose of each one of the sensors. That means that the calibration produces an accurate geometric transformation (i.e., the translation and rotation parameters) from any other link to the sensor. A successful calibration results in a sensor pose very close to the reality, consequently giving very reliable information to the robot about the distance and orientation of any detected obstacle.

In order to calibrate two sensors, an association of the data from one sensor to the data of another is required. Only by knowing this association, it becomes possible to estimate the transformation parameters between those sensors. After having a first guess, or initial estimate, of these transformation parameters, an optimization procedure can be implemented with the objective of enhancing those parameters by minimizing the error of the data associations. The relation between both data requires a common known object that could be detected and recognized by both sensors. Since the error of the data associations is the guideline for the optimizer, the more accurate the error is, the better performance the optimizer will have. With the purpose of getting a true and reliable error, most of the calibration procedures make use of objects that are robustly and accurately detected by all studied sensors, even if they have different modalities. These objects are called *calibration patterns*.

Thus, when the sensors from a robot have distinct modalities, like 2D LiDARs and RGB cameras, developing a methodology that relates the same pattern object correspondent detected data, in the same virtual environment is needed. Distinct sensor modalities give information with distinct nature. For example, lasers return a group of measurements, and the cameras detect image data. This is one of the first problems founded in multi-modal sensors calibration: relating data of different nature to identify the same object, in the same virtual space, and, thus, computing the error of that association. Fortunately, all the works about pairwise calibration already fought this problem and generated a lot of solutions depending

on the modalities of the sensors in consideration.

The big issue, yet to be consistently tackled, is calibrating complex robot systems with a lot of links and several multi-modal sensors scattered by different parts of the robot. Normally, the error to minimize by the optimization is due to the association between two sensor data. When the robot has more than just two sensors, calibration procedures can not be performed using, directly, a pairwise approaches. For those cases, a different methodology to estimate the pose of the sensors is required. One way to solve this problem is to execute a sequence of pairwise calibrations. One example of a sequential pairwise calibration occurs when the first sensor is calibrated with the second one, and then the second relates to the third, and the third to the fourth, and so on. Another sequential pairwise configuration could be the calibration between each one of the sensors with a reference one. That is, one of the robot sensors is the reference and then the pose of each of the remaining ones will be estimated relatively to the reference sensor. In one case or another, the implicit problems of this type of approach are still present.

The major shortcoming of the sequential pairwise approaches is related to the fact that the transformation between two sensors are estimated only considering the error between the data from the selected sensor tandem. In fact, if two sensors are calibrated using only their own detected data, the pose of each one relatively to the other will end up well estimated (if there is sufficient data with enough variety) but their pose relatively to the rest of the sensors (and the robot) will be compromised, causing lack of accuracy in the overall final calibration procedure. This means that the data and the pose from additional sensors must also be considered when it comes to estimate the position and orientation of each sensor. Figure 1.2 shows an example where a sequential pairwise calibration occurs. In this case, there is a reference sensor, with which the sensors one, two and three were calibrated, and then the transformation between sensor three and sensor four were estimated.

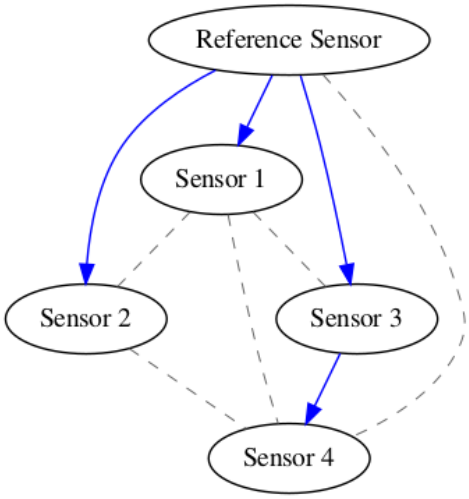


Figure 1.2: Calibration setup example: solid blue arrows represent the estimated transforms by a pairwise approach and dashed gray lines stand for other possible arrangements (and respective data) that were not considered by the calibration ([19]).

The error between the data associations relative to the blue solid lines will be, at the end of the calibration procedure, minimal. On the other hand, the error associated to the dashed gray line combinations could be very high, since they are not considered. That is caused by the wrong final configuration lead in by only considering a subset of the available data combinations.

Another known problem of sequential pairwise calibrations is the cumulative error that this transformation parameters sequential estimation entails. Successive calibrations lead to recursive errors, generating, in that way, a non acceptable deviation of the overall sensor pose, at the end of the calibration procedure.

In addition to the problems already mentioned, multi-sensor calibration procedures often change the robot transformations graph. Complex robot systems are characterized for having multiple links and sensors on different transform chains (robot arms), at distinct levels. In order to get the specific geometric transformation, between two sensors, that is supposed to be estimated, there is the need of grouping the extra information of the robot transformation graph, so that the sensors links could be at the same level and, thus, directly related. For example, if a robot has a sensor connected to the reference frame and another at the end of an arm with several links, it becomes hard to get the relative position between those two sensors and knowing which geometric transformation should be estimated. The change of the transformation graph could put all the sensors at the same level, but, as it sounds, changing the structure will probably bring other problems. The difference between the original transformation graph and the one that was derived from sequential calibration procedure is another common issue related to multi-sensor robots calibration.

1.2 Objectives

ATLASCAR2 is an intelligent vehicle developed at the Department of Mechanical Engineering of the University of Aveiro in Portugal. This electric vehicle (Mitsubishi i-MiEV from 2015) is one more prototype of many autonomous cars developed from the ATLAS project[2]. ATLASCAR2 has four sensors: two 2D LiDARs and two RGB cameras. Figure 1.3 shows the sick LMS1xx lasers and the pointgrey flea3 cameras that are on board of the ATLASCAR2.

(a) (b)

Figure 1.3: Sensors on board the ATLASCAR2: a) sick LMS1xx 2D LiDAR; b) point grey flea3 RGB camera.

[2] http://atlas.web.ua.pt/

As an autonomous vehicle, ATLASCAR2 is considered a complex robot system, since it has several sensors with different modalities. In Figure 1.4, we can see the ATLASCAR2 with its two RGB cameras and its two 2D LiDARs.

Figure 1.4: ATLASCAR2: Autonomous vehicle from the Department of Mechanical Engineering of the University of Aveiro; the sensors are indicated by the yellow ellipses.

Because of the position of each sensor relatively to the main body of the car, from now on, the ATLASCAR2 lasers will be called *left_ laser* and *right_ laser* and the cameras will be called *top_ left_ camera* and *top_ right_ camera*.

The main objective of this work is to calibrate all sensors with different modalities (lasers and cameras) on board the ATLASCAR2. In other words, the ambition is to get the position and orientation parameters of each sensor relatively to any other robot link. These parameters should be accurate, i.e., as close as possible to the real world ATLASCAR2 configuration. In the case of the cameras, besides this extrinsic calibration (estimation of the sensors body pose), there is also the target of enhancing the intrinsic parameters. Within this main objective, there are a few more specific purposes for the development of the presented project.

The first one is to create a calibration procedure capable of enhancing the pose of every sensor simultaneously. That is, in order to contour the obstacle related to sequential pairwise calibration, all sensors pose must be estimated at the same time.

The second specific goal is not to change the original transformation graph whenever the calibration occurs. The implemented approach should preserve the graph topology. This will ensure that other robot functionalities will not be compromised.

Robot Operating System (ROS) is one of the most used open sources for robot development, by many research communities all over the world. Building a project in ROS framework

is always profitable in a way that it can easily be used and improved by others robot developers. So, it could be said that a third objective is to implement this calibration procedure in a ROS environment.

In summary, the most important target of this work is to create the first straightforward software package for the calibration of intelligent vehicles, or robots with multiple sensors in general.

1.3 Document Structure

This document is divided in seven chapters.

The first is this one (*Introduction*) where all the information needed to understand the purpose of this work is exposed. *Introduction* is composed by the problem context, some required concepts explanation, the robot system where the solution will be implemented and the objectives of the project.

The second chapter, *Related Work*, approaches several studies and solutions already achieved, from the past, in the robot sensor calibration field. It also mentions the shortcomings of these first works, giving, in that way, the knowledge of what was not yet accomplished.

Experimental Infrastructure, the third chapter of this document, presents diverse tools used for the calibration procedure. Some of these tools already existed before, like the ROS packages, and others were created and implemented for this concrete work, such as all utilities for the optimization stage.

The fourth and the fifth chapters, *ROS Based Calibration Setup* and *Optimization Methodology*, respectively, are the core of the proposed approach to calibrate multi-modal sensors on board the ATLASCAR2. In an overview, the suggested solution is decomposed in two distinct stages: calibration setup (chapter four) and optimization (chapter five). The configuration of the calibration has the purpose of collecting all required data for the optimization. Chapter four describes how this interactive, semi-automatic and ROS based methodology for gathering the indispensable information works. On the other hand, chapter five exposes the optimization procedure, explaining how it will enhance the sensor pose parameters.

The sixth chapter, called *Results*, presents the errors obtained from the application of the proposed approach in ATLASCAR2 sensors. In order to took some conclusions relatively to other distinct calibration procedures, *Results* compare the errors related to the final pose of the same sensors estimated with different calibration tools. In that chapter, an accuracy level comparison is uncovered, allowing to take some conclusions about the efficiency and reliance of the proposed approach.

Conclusions and Future Work is the seventh and last chapter of this document. It contains conclusions about the usage of the suggested idea to calibrate the sensors on board the ATLASCAR2 both in results and in the human-friendly appliance methodology. It also indicates some further possibilities to boost this proposed calibration procedure.

Chapter 2

Related Work

Sensors extrinsic calibration of a complex robot system is, increasingly, one of the major issues among robot develop communities. Over the past several years, a large number of studies and works in the field of calibration were done, bringing many new features and options for the different purposes of each project.

Most of the studies rely on a sensor to sensor pairwise calibration: [2] calibrates a 3D and a 2D laser range finders; [16] reports an online stereo calibration (camera to camera) without any marker; [18] also addresses the stereo calibration without a specific calibration pattern; [26] presents a solution to calibrate a camera with a laser range finder; and [27] describes the calibration of a binocular stereo vision. These are just some examples among the big world of pairwise calibration. These works have their own value since they present several ways to relate the pose between two sensors: with or without a reference pattern for calibration, offline and online, and even if the sensors have distinct modalities.

In fact, the problem of extrinsic calibrating multi-modal sensors has plenty of solutions when it comes to estimate the pose between only two sensors (pairwise).

In [6], for example, a method to calibrate RGB to RGB cameras, with the particularities of working for different type of cameras with varying focal lengths and tolerating the alignment variation problem from the standard stereo systems, is proposed.

For the case of two depth cameras (RGB-D cameras), [3] presents a human-friendly, reliable and accurate calibration framework between a color-depth sensor couple. This calibration ensures that the intrinsic and extrinsic resulted parameters could be used in some robotics applications that require some accuracy, such as highly accurate 3D environment reconstruction and mapping and high precision object recognition and location. In [4], besides the intrinsic and extrinsic parameters estimation of RGB-D cameras, the mean and variance of the depth error at different measurement distances are also estimated. More RGB-D cameras and hand-eye calibrations methodologies are described in [14] and [28].

If the problem is to find the relative pose between a 2D laser-rangefinder (LRFs) and a camera, [10] proposes an approach which relies on the observations of corners in human-made scenarios. This approach does not need any specific calibration pattern, which makes the process faster and simpler to perform. Besides this work, there are other studies that manner to solve the extrinsic calibration of a camera - 2D LiDAR system: [29] was the first published calibration tool for this problem. By studying [5] and [13], we notice that these works have distinct objectives, since the first purpose is to project a special 3D calibration target in order

to enable an automatic detection procedure, and the second work describes some extensions to an already existing calibration tool so that the range of applications becomes wider.

Another example of pairwise calibration that relate different sensor modalities is [11], where the poses of a 3D LiDAR to camera system are estimated without user intervention.

More related to the intelligent vehicle work space, [9] features a spacial calibration of a radar and camera sensors that turn out to be very useful for road detection with real-time information.

As we can see, there are diverse approaches to estimate the poses of a pair of sensors with various modalities. The majority of the presented works achieved excellent results, by providing the most accurate and reliable position and orientation parameter calibration. Because of these good results, many of the created and studied tools for detection or for pose enhancement could be used.

The problem starts when the objective is to calibrate a system with more than two sensors, all at the same time.

There are some works that address the calibration of more than two sensors. In [15], for example, a method to estimate the relative pose between a Kinetic (RGB-D camera) and three RGB cameras simultaneously is proposed. This study could actually get better results than the manufacturer's calibration.

Intelligent vehicles are known as complex robot platforms with several sensors of different modalities. ATLASCAR2 (see [25]) is one of these robots with fours sensors: two RGB cameras and two 2D LiDARs. For the purpose of calibrating ATLASCAR2, [20] propose a methodology that applies multiple pairwise approaches. In other words, this work estimates the pose of a first sensor related to a reference sensor, and then estimate the pose of a second related to that same reference sensor, and so on. Although being a proven solution to the mentioned problem, this sequential calibration has a critical disadvantage.

The shortcoming of the approach proposed by [20] to calibrate the multiple sensors on board of the ATLASCAR2 is that the transformation tree that describe the robot system is altered during the procedure. In this particular case, the output transformations graph is a simple tree with one reference sensor link and all other sensor links connected to it. That means that the calibrations were all between the reference sensor and one of the others. This approach to calibrate all the sensors of a complex robot system has the need of changing the structure of the robot description, so that it could be possible to easily identify where the sensor links are and which joint should be estimated. This is a critical factor for any calibration procedure: they should not change the structure of the original transformation tree. The reason is that this tree, besides the composition of the robot links and the relative position within each others, also supports additional functionalities such as robot visualization or collision detection. If the transformation graph changes due to the calibration, those functionalities are compromised or have to be redesigned. It would be great for research communities all over the world if there were some project capable of reading any robot description and independently from how many or where the sensors are and, without changing the original transformations tree, could calibrate all sensor poses at once, without compromising the results.

Taking all this into account, we are convinced that there is still room for improvement, since no method for the calibration of several sensors with different modalities, adapted to any configuration, is already available.

The major problem related to sequential pairwise calibration is that the pose of the two sensors in consideration is estimated using only the data provided by those same two sensors. Since fused sensor data is often needed for complex robots to perform precise tasks, the detected data from additional sensors, which usually is available, is essential for the complete calibration procedure. In order to estimate the pose of more than two sensors simultaneously, a calibration with a bundle adjustment methodology is required.

In this way, the proposed optimization of the ATLASCAR2 sensors poses, in this project, is similar to these type of approaches. In [1], the implementation of an algorithm capable of solving large-scale problems with tens of thousands of images is presented. More than just solving, [1] also evaluates the performance of distinct bundle adjustment algorithms by using several community photo collections and a related dataset. This mentioned study offers a good state of the art performance for large-scale bundle adjustment problems.

The approach, explained in this document, for estimating the pose of all ATLASCAR2 sensors is, in some points, close to the methodologies used in [22] and [24].

A framework for calibrating all of the sensors and actuators of a robot together is presented in [22]. Complex robots bring many challenges for calibration procedures, one of them being, for example, the different error characteristics of each sensor. In fact, a camera's precision in re-projecting points to its own image is very dependent on its focal length, and a laser detects 3D points in a distinct way than two cameras, for instance. More than this, there could be an extra uncertainty that should be considered in each sensor error: the linkage error. This linkage error depends on the arm where the sensor in question is mounted. In [22], the PR2 mobile manipulation platform was used as a tested example of the developed general optimization procedure. This optimization was modeled after the bundle adjustment framework of computer vision. And here is the similarity to the promoted project of ATLASCAR2 calibration. Despite being used for a specific robotic platform, the optimization methodology of [22] is general and the calibration procedure adopts a bundle adjustment philosophy: two characteristics that are also promoted by the present project.

A general approach to joint estimation of temporal offsets and geometric transformations among all robot sensors is presented in [24]. There are several properties of known multi sensor calibration approaches that make difficult to generalize the work to other applications or different combinations of sensors. In [24] an idea without relying on unique properties of a specific sensors configuration and, therefore, fitting for any temporal calibration procedure in robotics is developed. This general work was demonstrated by calibrating, in space and time, a setup with a stereo camera and a laser-rangefinder and a camera with respect to an inertial measurement unit. This calibration flexibility for any robotic system is also a purpose of ATLASCAR2 multi sensors calibration project, since it represents a big advantage for robot developers.

Finally, [23] describes one of the most used open sources for robot development: ROS. As was mentioned in chapter 1, creating a ROS software to calibrate the ATLASCAR2 multi-modal sensors is very advantageous, since it allows for the use of several updated tools and working in the same framework as the large majority of robot developers communities. This homogenization also makes the developed software usable for other researchers, who might need it. For a better understanding of how ROS system works, chapter 3 will briefly explain the ROS framework philosophy and mention the most important tools used for the purpose

of calibrating the ATLASCAR2 multi-modal sensors.

As part of this project, [19] was derived from the work described in this document. This published paper also explains the proposed approach here presented.

Chapter 3

Experimental Infrastructure

To achieve the objective of calibrating all multi-modal sensors on board the ATLASCAR2, a procedure mainly composed by two stages, that are detailed in the next two chapters, was developed. The first developed procedure stage operates in a ROS framework. On the other hand, the second moment consists of an optimization strategy implemented in Python language.

Before explaining the complete proposed approach of this project, it is essential to identify and understand some concepts and tools that were used for the purpose.

3.1 Robot Operating System (ROS)

ROS is an open source frame work for robot development. This system is structured in a way that even the most complex and large robot platforms end up to be simply manipulated. Besides this, ROS framework is easy to implement in any modern programming language like Python, C++, for example. For all these reasons, ROS is used by many robotic research communities all over the world, that also contribute to its growth.

The first phase of the presented project was implemented in a ROS system environment. For a better understanding of the programs, tools and features developed for the purpose of calibrating the ATLASCAR2 multi-sensors, or the sensors of any robotic system in general, it is important to be aware of some ROS properties.

ROS nodes

A ROS node is a process that performs computation[1]. In other words, each task in ROS frame work is always executed by a node. A node can subscribe a ROS topic and listen to the messages that are being published within it; besides listening, the node can also publish ROS messages in that specific subscribed ROS topic. In this way, the information can be received, processed, and published by each node. In a system with several nodes, this communication philosophy allows to have many processes being performed simultaneously. For example, a node could be publishing all transformations between each connected pair of the robot links, other node could be providing a graphical view of the simulated scene, other node could be controlling the robot's wheel motors, and so on.

[1] `http://wiki.ros.org/Nodes`

ROS topics

ROS topics are named buses over which nodes can exchange messages[2]. A specific topic can be subscribed by several nodes, without each one of them knowing who the other subscribers are. Not only subscribe, but multiple ROS nodes could publish generated data to a ROS topic, once again, without each node knowing who the other publishers are. This is the way nodes can exchange messages: those who want to provide information should publish it in a ROS topic. On the other side, a node that needs to acquire data should subscribe the interested ROS topic. ROS topics only communicate in one direction, which means that they should not be used by a node that needs a response to its request (like in services).

ROS messages

ROS messages are simple data structures[3] that ROS nodes could listen to from the subscribed topics or could publish to a topic. The simplified ROS messages description makes it easier to generate the code of a message in several distinct programming languages. ROS messages content is composed by fields and constants. Fields are the data that the ROS message has and constants are the values that define that data. Messages support various types such as *integer*, *boolean*, *floating point*, etc.

RQT

RQT is a ROS software framework that provides several Graphical User Intervention (GUI) tools[4]. Each graphic tool is a plugin that RQT shows in a screen window, allowing to monitor the information that is being provided by the plugin. For example, one of the most used GUI tools in RQT is node graph. Node graph, as it sounds, is a graph where all working ROS nodes are represented as well the ROS topics involved in their communication. Figure 3.1[5] exhibits an example of a simple node graph displayed by the RQT window.

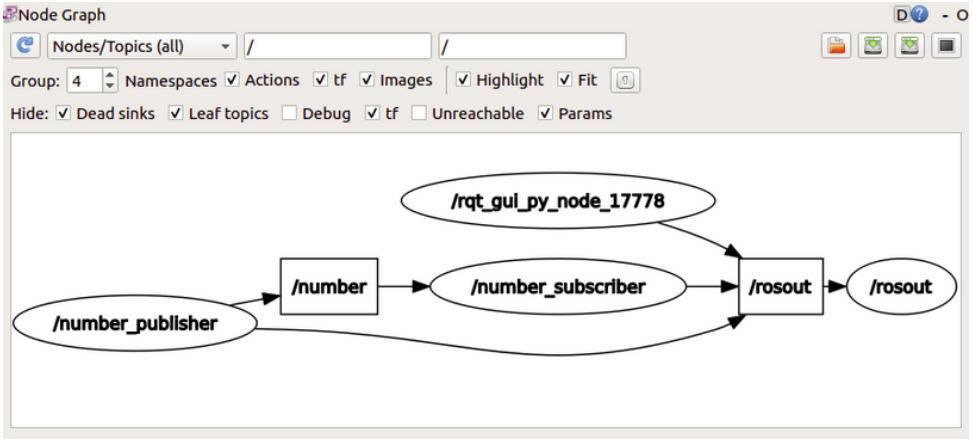


Figure 3.1: RQT window showing a simple node graph: each rectangle represents a ROS topic and each ellipse a ROS node.

[2] http://wiki.ros.org/Topics
[3] http://wiki.ros.org/Messages
[4] http://wiki.ros.org/rqt
[5] https://roboticsbackend.com/rqt-graph-visualize-and-debug-your-ros-graph/

In Figure 3.1, each node is represented by an ellipse and the topics are the squares. For example, *number_publisher* is a ROS node that publish ROS messages for *number* topic; *number_subscriber* is a node that subscribes the *number* topic and publish to the *rosout* topic.

For the development of complex robot systems, RQT provides another very useful GUI tool: the *tf* tree. *Tf* tree is the transformation graph tree that represent all robot structure, since it shows every robot link and the joints that connected them. In Figure 1.1 we had already seen a small example of how a transformation graph looks like. *Tf* tree tool is very important in the sense that it allows to keep track of the robot coordinate frames and to know the distributed information sources, in order to apply transforms to this data properly. [8] describe the complexity of all problems that *tf* tree (and *tf* library, in general) solve in robot development. Figure 3.2 shows a simple example of a *tf* tree displayed by RQT.

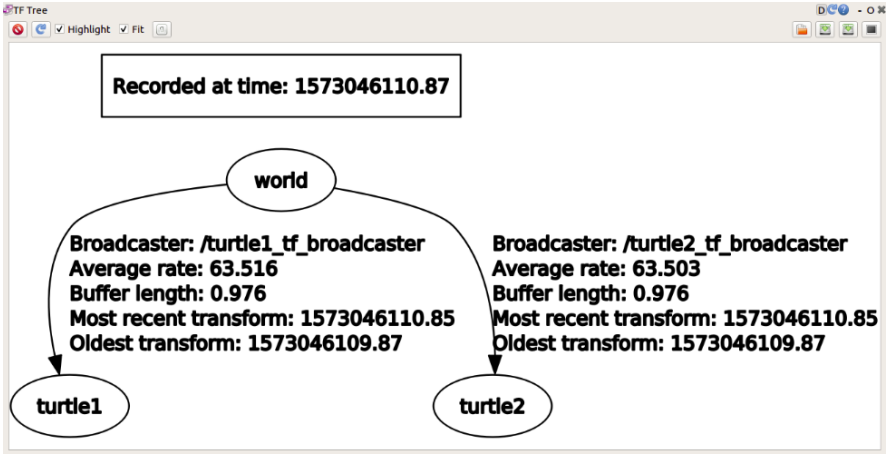


Figure 3.2: RQT window showing a simple *tf* tree.

In this example, *world*, *turtle1* and *turtle2* symbolize the robot links. In the present work, the transformation graph will be approached and studied several times, since sensors data collection from complex robot systems turns out to be one of the biggest issues that robot developers try to solve.

RViz

RViz is a 3D environment visualization for ROS framework. In RViz it is possible to see a simulated 3D scene with the robot model, with each one of the link coordinate frames, with sensor detected data (like point clouds for lasers, or images for cameras) and many more features. This 3D environment is very useful for comparisons between the real world and the simulated one. For this particular reason, RViz will be essential in the first main stage of the presented procedure for calibrating all multi-modal sensors on board the ATLASCAR2.

ROS launch files

ROS launch files are responsible for launching multiple ROS nodes[6]. This means that if the purpose is to have different processes operating at the same time, they can be all launched by one ROS launch file. These files are written in XML configuration and have a *.launch* extension.

ROS bag files

ROS bag files are used to store ROS messages data[7]. When a bag file is created, one or more ROS topics are subscribed. The serialized message data that is being received by those topics is stored in the created bag file. Then, each *.bag* extension file could be played back and the information it contains will be published in the same ROS topics or even remapped new ones, at ROS framework. This feature is very helpful in the way that it allow to record and store all detected data from a real scene and only process, analyse and visualize it later. For the ATLASCAR2 multi-modal sensors calibration, a ROS bag file was created. In this way, the project could be developed remotely and not necessarily on the computer on board the ATLASCAR2.

ROS packages

ROS packages are useful modules that constitute the organization of ROS framework[8]. A package could contain ROS nodes, ROS libraries, configuration files, and so on. For example, *rosbag*, *roslaunch*, *rviz* and *rqt* are some ROS packages that must be installed to use all the functionalities that were already mentioned in this section. Beside this, sensor packages (*LMS1xx* and *pointgrey_camera_driver*) were also installed for this work, in order to have the characteristics and body model of each laser and camera.

3.2 Optimization Utilities

The second phase of the proposed approach to calibrate the multi-modal sensors on board the ATLASCAR2 consists in optimizing the first guess of each sensor position and orientation parameters. Thus, an optimization procedure was implemented with the goal of minimizing the error between the coordinates of some known detected points and the coordinates of that same points re-projections. Further in this document (chapter 5), the way this computation works is explained and all the steps performed by the optimizer are detailed. It is important to notice that in [17] was already developed the optimization methodology that is going to be adapted and used in this work.

For the optimization workflow, a Python class called *Optimizer* with several functions was created. These functions process the requested data to enable the optimizer to actually minimize the residuals (errors). To help to understand the implementation of the optimization

[6] `http://wiki.ros.org/roslaunch`
[7] `http://wiki.ros.org/Bags`
[8] `http://wiki.ros.org/Packages`

procedure, it is important to have an overview of the functions that compose the *Optimizer* class:

Optimizer.addModelData function

This function is called right at the beginning of the optimizer setup, since it is responsible for importing data models needed for the optimization. This dataset should be a dictionary containing every static information to be used by the cost function.

Optimizer.pushParamVector function

Each group of parameters to optimize is pushed by this function. There are different types of vectors to push: the translation vector, the rotation vector and, for the cameras case, the vector with the intrinsic parameters. In section 5.1, all the optimization parameters are explained.

Optimizer.pushResidual function

This function adds a new residual to the existing list of residuals. Every residual computed by this optimization is detailed in section 5.3.

Optimizer.computeSparseMatrix function

The sparse matrix, presented in section 5.4, is computed by this function. Since *Optimizer.computeSparseMatrix* needs the optimization parameters and the residuals, it should be called only after setting both of them.

Optimizer.setVisualizationFunction function

During the optimization procedure, there is the possibility to have graphical feedback of the sensors pose being enhanced, the residuals being minimized and more data plotted. This function was implemented for that purpose. Figure 5.3, Figure 5.4 and Figure 5.1 are examples of the output of this function.

Optimizer.startOptimization function

This is the function that initializes the optimization procedure. As explained in section 5.5, the least squares optimization function from scipy library is used.

Optimizer.printParameters function

In order to monitor the values of the optimization parameters as well as the corresponding data models, *Optimizer.printParameters* was created. It can be called whenever the user wants to.

Chapter 4

ROS Based Calibration Setup

The purpose of this work is to calibrate as many as possible different type of ATLASCAR2 sensors. The proposed road to this objective has two main stages in the procedure: Calibration Setup and Optimization. The calibration setup is essential in the way that it will give the optimizer all the information that it needs to enhance the pose of each sensor. As will be further detailed, the optimization process needs the parameters to be optimized, a function that computes the error produced by the values of those parameters and first values of those same parameters. Thus, the Calibration Setup is composed by four moments (Extend and Parse Robot Description, Interactive Sensors Configuration First Guess, Interactive Data Labeling and Interactive Data Collecting) that will gather the optimization requested data and make it useful to that end.

Before anything else, with the purpose of finding the intrinsic values of the robot cameras, some conventional camera intrinsic calibrator is used. This introductory step has only the objective of giving the first value of the intrinsic parameters, since they are going to be optimized later (as will the extrinsic parameters). Thus, whichever calibrator system is used is not important: any of the conventional should result in good and reliable results.

For the ATLASCAR2 robot system, the intrinsic matrix of each camera was computed by ROS camera calibrator [1] (Figure 4.1).

Figure 4.1: ROS Camera Calibrator example.

After this, we are able to study the proposed approach for the calibration configuration. The four moments that compose the initial setup of the calibration procedure are explained in this chapter.

[1] http://mirror.umd.edu/roswiki/camera_calibration(2f)Tutorials(2f)MonocularCalibration.html

4.1 Step 1: Extend and Parse the Robot Description

In this section, it will be detailed how the ROS robot models are described and how they can be parsed. This parsing of the robot description will end up to be essential to configure how the calibration should be carried out.

4.1.1 Robot Description

In ROS, there is a specific format to describe a robot: it is called Unified Robot Description Format (URDF). This URDF allows you to describe all robot physical properties in an *XML* file[2]. A well known and much used, by robot development communities, ROS package is *xacro*[3]. The word *xacro* stands for *XML macro*, which means that, with *xacro* package, it is possible to describe the exact same robot but in a smaller and compact, thus easier to human reading, *XML* file by using macros.

The URDF has some different elements to describe the robot. These elements should be declared in the *XML* file. Usually, a robot description consists of a set of links connected by a set of joints. Beside these two (link element and joint element), there are a few more URDF elements like robot, transmission, gazebo, sensor, model_state and model. Each element has its own attributes and elements to be specified. For example, a joint element has a 'name' and a 'type' as attributes and *origin*, *parent link* and *child link* as elements. The *origin* is the geometric transformation from the *parent link* to the *child link*. Here is a general structure of a robot description *.xacro* document:

```
<robot name="any_robot_name">
  <link> ... </link>
  <link> ... </link>
  <link> ... </link>
  <link> ... </link>

  <joint name="..." type="...">
    <origin xyz="..." rpy="..." />
    <parent link="..." />
    <child link="..." />
  </joint>
  <joint>  ....  </joint>
  <joint>  ....  </joint>
  <joint>  ....  </joint>
</robot>
```

The root element of a URDF is always the robot element.

[2] https://www.ros.org/core-components/
[3] http://wiki.ros.org/xacro

Two major benefits of using URDF are that *.xacro* files allows to:

- call other *xacros*. Thus, when it comes to describing complex robot systems, it gets a lot easier and much more organized by having separated documents for distinct parts of the robot platform.

- import meshes. If a body link hasn't a standard geometric shape (for example, cubic or cylindrical), a *.dae* file could be included in the robot description leading to a more realistic visualization of the project simulation, in RViz.

The ATLASCAR2 is an intelligent vehicle with several sensors which make it a complex robot. In order to access and study its complete description, all sensor packages had to be installed so that the *xacro* file could import them to the ATLASCAR2 environment. The cameras are both of the same type, as well as the LiDAR sensors: *pointgrey_ camera_ description* package [4] and *sick_ ldmrs_ description* package [5].

Considering all this, when launched the ATLASCAR2 project, the simulated robot model in RViz environment (Figure 4.2) as well as the correspondent *tf* tree in RQT (Figure 4.3) could be seen.

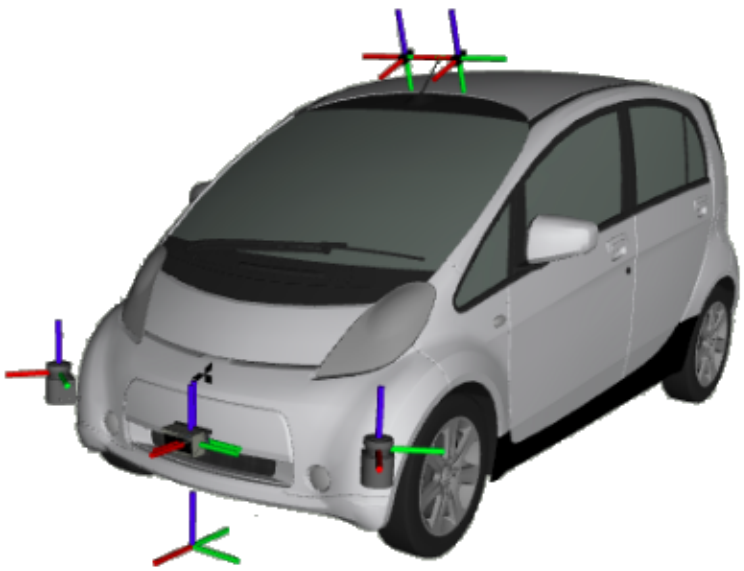


Figure 4.2: ATLASCAR2 model in RViz, with the links coordinate frames.

[4] `http://wiki.ros.org/pointgrey_camera_driver`
[5] `http://wiki.ros.org/sick_ldmrs_laser`

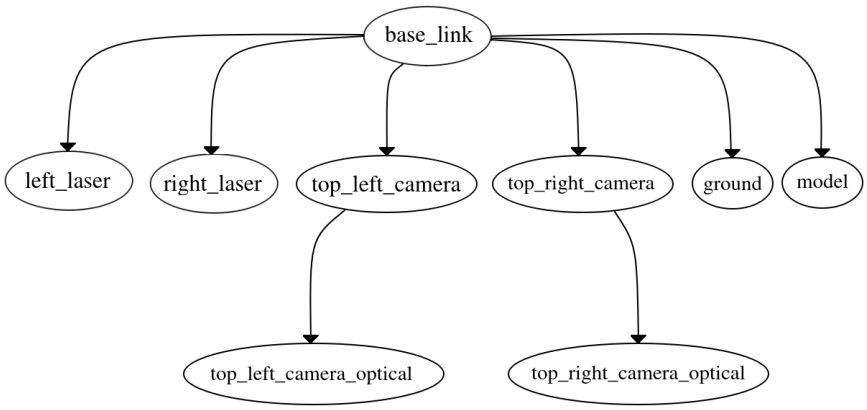


Figure 4.3: *tf* tree of ATLASCAR2, shown by RQT. The *ground* and *model* frames are components of the ATLASCAR2 system which are not related to each sensor transformation chain.

4.1.2 Adding a Calibration *xacro* and Parsing the Robot Description

The purpose of the calibration procedure is to find the real position and orientation of each sensor relatively to a reference coordinate frame or any other robot link. For this, it is essential to know which are the transformations to access between all the available transforms within the robot description (and represented in the *tf* tree).

In order to know which joints should be calibrated, a new *xacro* document was created; this document is composed by a new element called *sensor* that has also been created. *sensor* element is composed, among others, by an associated sensor link (parent link), calibration_parent link and a calibration_child link:

```
<sensor name="..." update_rate="...">
    <parent link="..."/>
    <origin xyz="..." rpy="..."/>
    <calibration_parent link="..."/>
    <calibration_child link="..."/>
    <topic topic="..."/>
</sensor>
```

The transformation to be optimized is the one which expresses the *calibration_child* link pose relatively to the *calibration_parent* link pose. The *parent* link is the frame related to all detected data from each sensor.

Beside the new *sensor* element, the novel *XML* file also includes the previous, standard, *xacro* files. That's why this step is considered an extension of the robot description (see Figure 4.4).

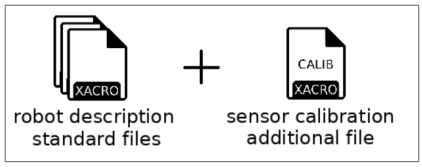


Figure 4.4: Representation of the proposed, ready to calibrate, complete robot description contents.

In this project, *atlascar2_calibration.xacro* is the new calibration document, manually written by a user that configures which are the transformations, related to each sensor, that are going to be calibrated. That is the necessary information that should be written in *sensor* elements in order to configure how the calibration should be carried out.

urdf_parser_py is a ROS package capable of parsing URDF models from a file and mapping the URDF information to a Python structure [6]. All the important robot model content to continue the calibration procedure (like the number of sensors and each sensor type and name) is obtained by this module. Since the *sensor* element was freshly created, the parser package had to be extended accordingly[7].

Thus, the process of finding the joint related to each geometric transformation becomes direct: the *calibration_parent* link and the *calibration_child* link should match with the joint *parent* link and the joint *child* link, respectively.

Figure 4.5 shows the ATLASCAR2 case of study: because of the four sensors considered, there are four joints (four transformations) to be optimized.

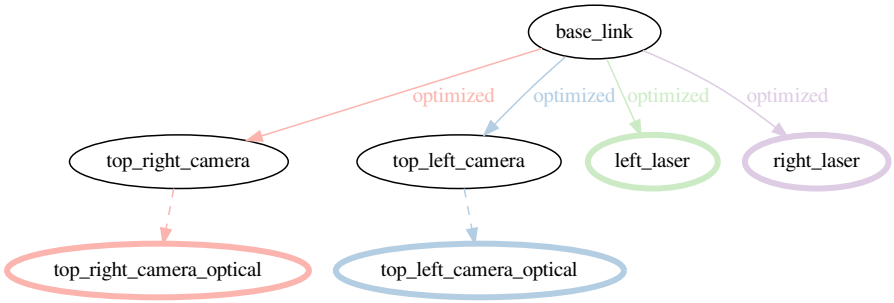


Figure 4.5: Portion of ATLASCAR2 *tf* tree with the identified transforms to be optimized.

[6] `http://wiki.ros.org/urdf_dom_py`

[7] It is known that already exists an element called *sensor* for Gazebo. However, the parser does not read Gazebo elements. In later versions of this project, the name of this new element was changed to *calibration*, in order to avoid any kind of problems.

Notice that the sensor link is not necessarily the child link of the joint to calibrate: in ATLASCAR2 cameras case, the data collected (as will be further explained) by these sensors is related to the optical link pose of each camera. Therefore, besides which transformations should be optimized, there is also the need of storing all the transforms that belong to the same sensor path. Those ones which are between the root link and the *calibration_parent* link are saved as pre-transforms, and the ones after the *calibration_child* link of the sensor path will be the post-transforms. This proposed approach allows to identify and gather all the important data, enabling the sensor calibration of any robotic system, even the most complex ones with a huge *tf* tree like the iconic worldwide known PR2 robot (Figure 4.6[8]).

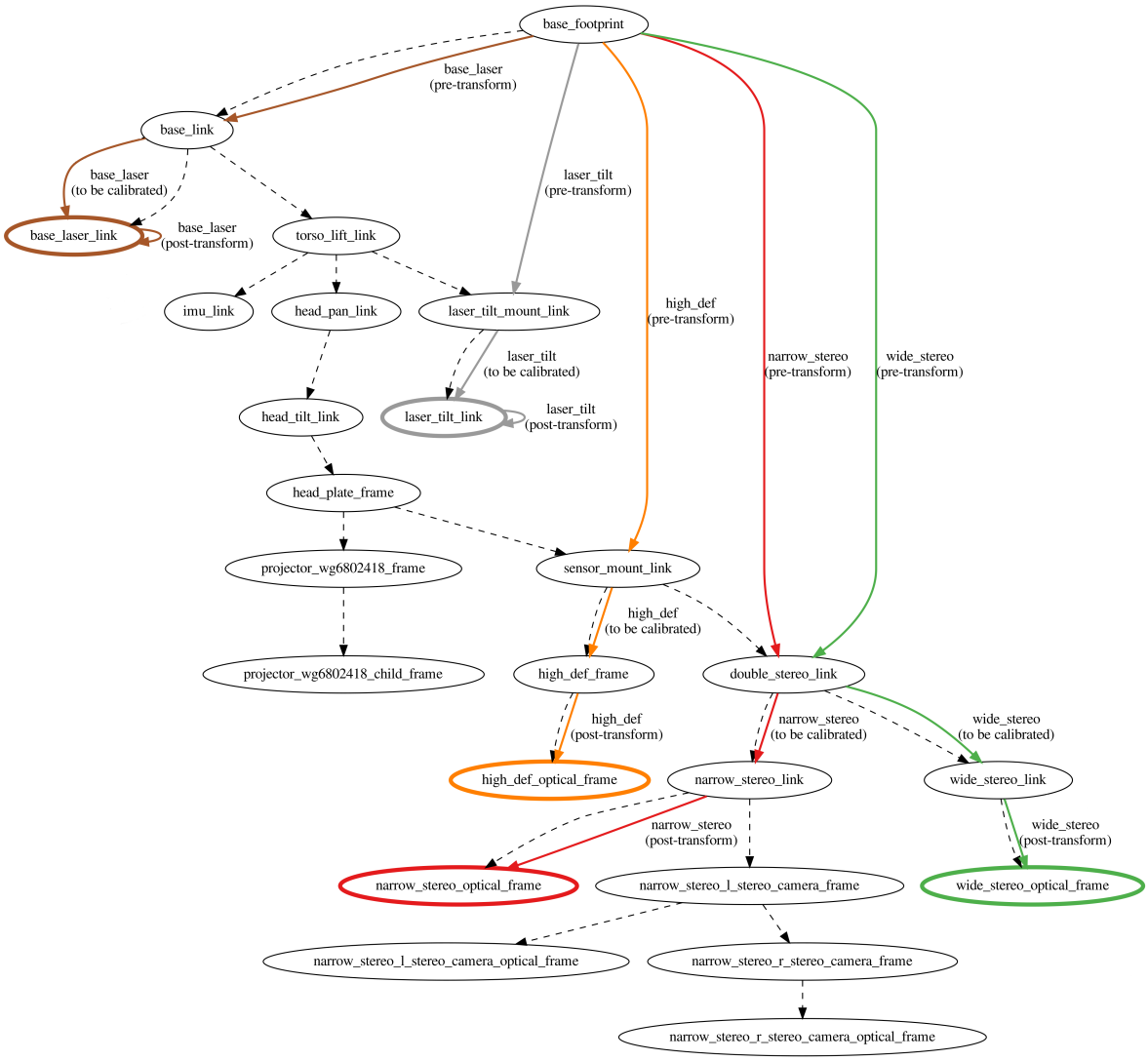


Figure 4.6: Part of the PR2 robot transform tree: each color represents the transformations path of each particular sensor; the link with a colored border is the sensor link.

Summing up, the robot description should result in what is already usual by many robot research communities all over the world plus the new *xacro* document that will provide the

[8] `http://wiki.ros.org/Robots/PR2`

information of which are the links of the sensors to calibrate and the corresponding transform to optimize. Each sensor has a chain of transforms from the reference link (usually called *base_link* or *world* in robotics) to the sensor link. By manually writing the new *xacro* calibration file, the user is allowed to choose any of the transforms within the path to be reformed. This means that the geometric transformation to optimize could be any of the sensor transforms path. In fact, if any of the sensor path joints changes its configuration, then the final sensor pose is going to be different too. So, despite the calibrated transform, this is always a sensor calibration procedure.

Figure 4.7 shows an abstract example of what could be a robot *tf* tree, with the sensor links and the correspondent calibration transform (solid arrows).

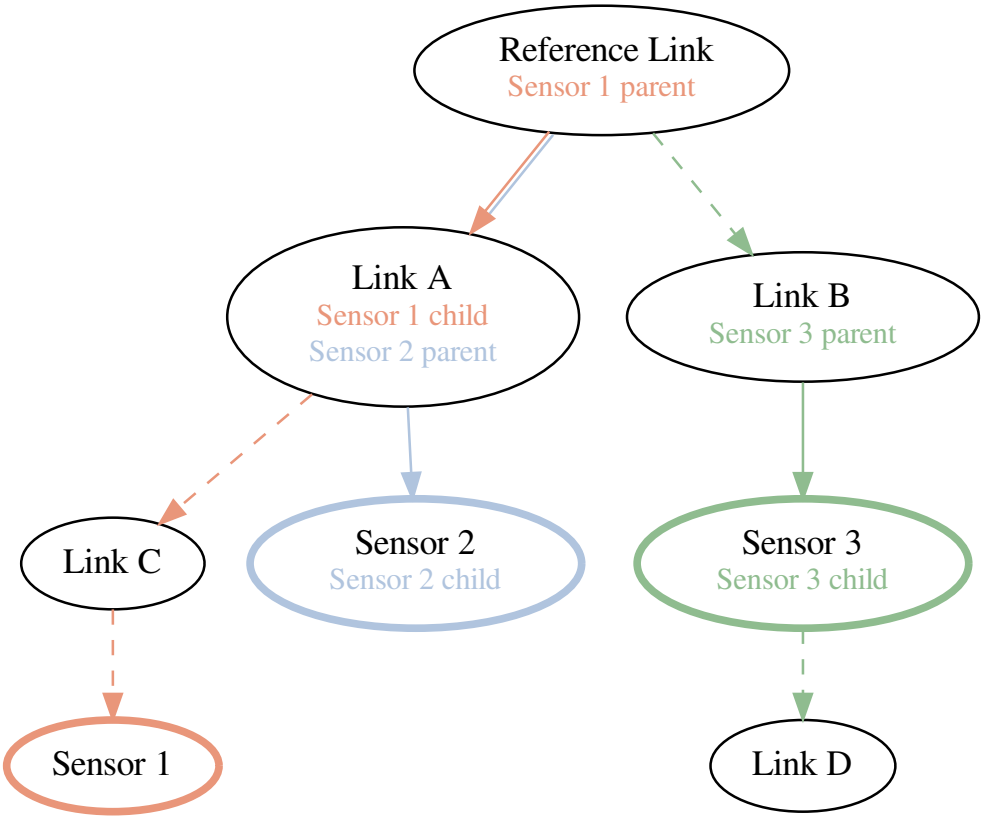


Figure 4.7: A conceptual example of a calibration setup: the transformations considered by the calibration are the solid arrows; each sensor to calibrate is in a link with the same color as its corresponding path.

As we can see in Figure 4.7, sensor link doesn't have to be at the end of the path either, like sensor 3. All this flexibility that comes from the user created new *xacro* file, in its own robot description, turns the calibration methodology general in the sense that the number of

sensors and their location are not restricted. Besides that, the presented methodology does not imply a change in the transformation tree of the robot, as most of the usual and known calibration procedures. These are huge advantages of using this proposed approach: it does not compromise additional robot functionalities (by changing *tf* tree) and it could be applied to any ROS compliant robot system.

4.2 Step 2: Generating a First Guess of a Configuration of the Sensors Using an Interactive Approach

The information provided by these four stages of the calibration setup phase are strictly demanded by the optimizer. Only the access to this data enables the possibility of finding the optimal solution, which corresponds to the global minimum of the optimization objective function (detailed in the next chapter). A well known problem for algorithms that search the global minimum of a function is being stuck in a local minimum value.

4.2.1 Problem of Local Minima

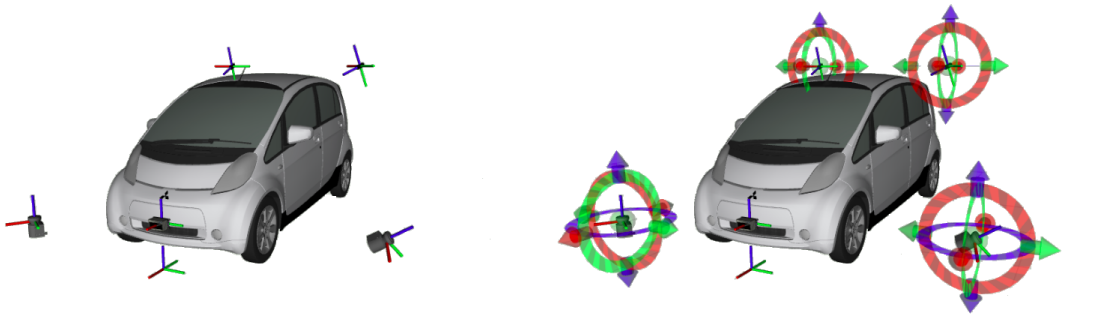
Standard optimization algorithms begin their search in a random parameters configuration and then run trough the function graph in the direction pointed by the differentiability (jacobian matrix). If a local minimum point was found, then the optimizer has the order to stop searching because if he gives a step into any direction it always reach to a higher value than before. This could happen even if there is a much lower (global minimum) point in other function section. This is called the local minima problem of optimization algorithms. In [21], there are some explained techniques to overcome this problem. The majority of them encourage variety in the potential solution set.

To make sure that the optimization will converge into the optimal solution, the proposed method in this work is different from the techniques mentioned before. If the first guess of the parameters is near to the optimal solution, then, intuitively, it is more difficult to run into the local minima issue. Thus, the goal of this step is to ensure an accurate first sensors pose configuration. In this way, the optimization will easily found the best possible configuration for the sensors poses.

4.2.2 Interactive First Guess of the Sensors Pose

An interactive way of setting the sensors pose that will be considered in the first iteration of the optimization procedure was developed. By parsing the robot description files, it is directly known how much and where the sensors in consideration are. One RViz Interactive Marker[9] was created for each sensor (the ROS package *interactive_ markers* [10] must be installed). Each interactive marker has the exact same pose as the associated sensor, and the user could freely translate and rotate the marker using just the mouse cursor. Figure 4.8 shows the misplaced sensor bodies and the interactive markers created in the same position, ready for the hand pose correction.

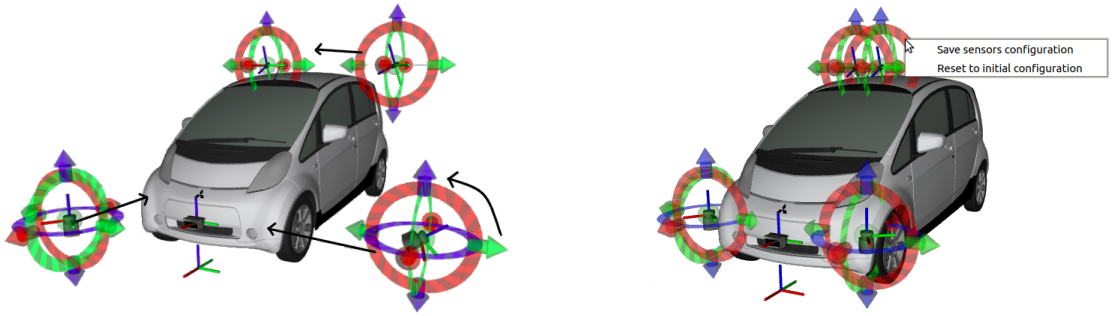
[9] `http://wiki.ros.org/rviz/Tutorials/Interactive%20Markers%3A%20Basic%20Controls`
[10] `http://wiki.ros.org/interactive_markers`



(a) (b)

Figure 4.8: ATLASCAR2 with the misplaced sensors in RViz environment: a) before creating the interactive markers; b) after creating interactive markers.

Along with the creation of the interactive markers, a ROS node responsible for continuously publishing the pose of all markers was created. By having this information published and always updated, the system can find the sensor bodies position and orientation in the robot model. Figure 4.9 shows that by dragging the RViz interactive marker, the sensor body will move accordingly.



(a) (b)

Figure 4.9: ATLASCAR2 and the interactive markers in RViz environment: a) before correcting sensor positioning; b) after correcting sensors positioning.

The fact that the sensor configuration is changing in accordance to the interactive marker pose, defined by the user, allows a real time comparison with the real world scene. This is a clear advantage of this interactive way of defining the sensor pose first guess: as is shown in Figure 4.10 , sensor configuration in RViz is very similar to the real position relatively to the car model.

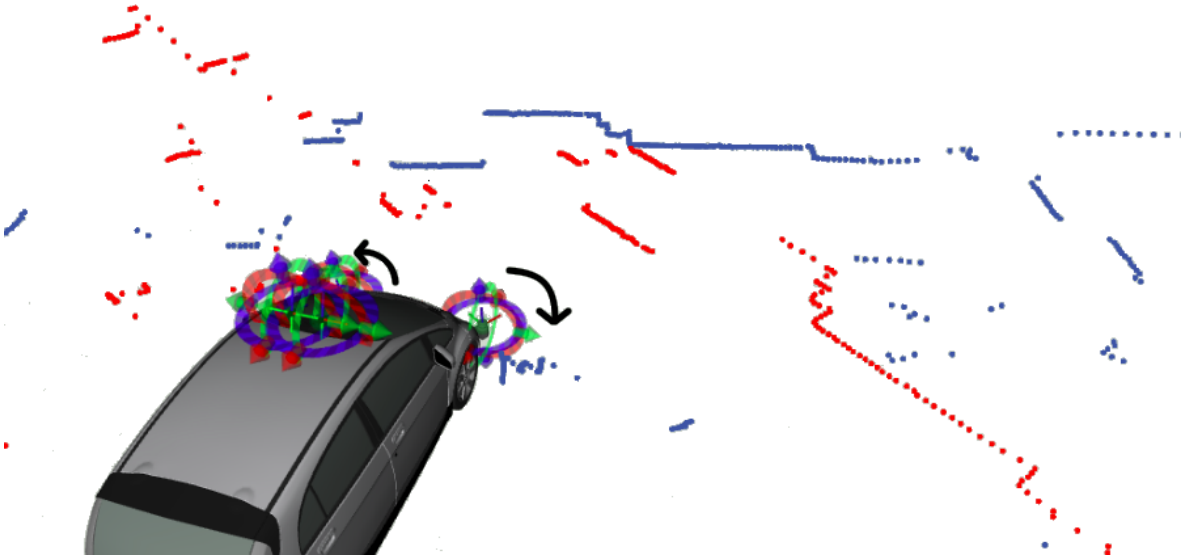
(a) (b)

Figure 4.10: Sensors configuration comparison: a) Real world scene; b) RViz environment.

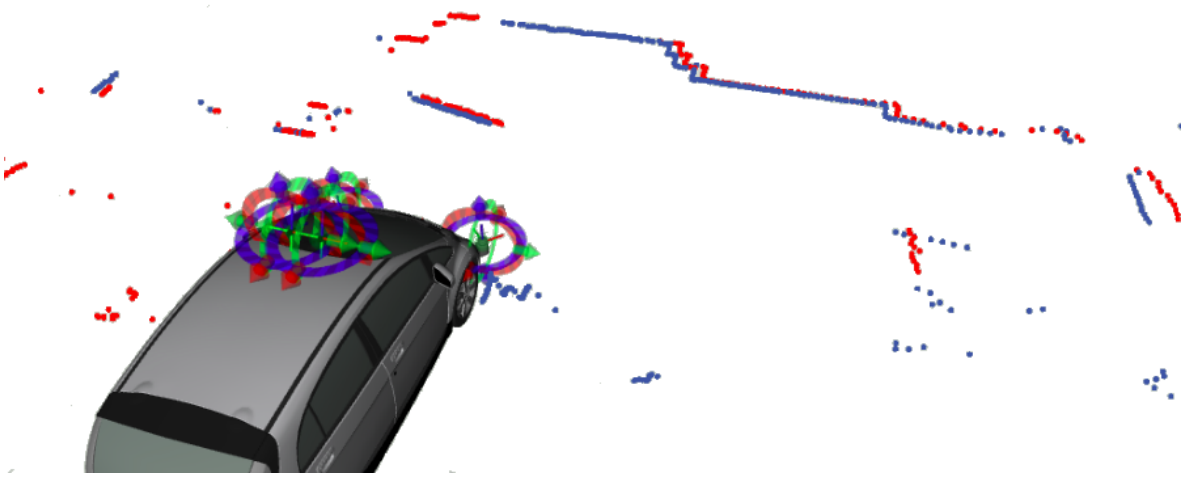
RViz provides the visualization of each sensor captured data at each time stamp. This happens because the novel *sensor* element, used in the new calibration *xacro* file of robot description, has a *topic* element which is the name of the topic where sensor data information is stored. When launching the ATLASCAR2 project, RViz node subscribe this topic to have access to the real time data and be able to display it in RViz environment. If sensor is a camera, the user can see the moment image that is being captured; if sensor is a LiDAR, the respective captured point cloud is displayed. It is not only the sensors bodies that move accordingly, the interactive markers pose, so does the sensor data.

For an even better first guess of sensor configuration, besides the eye comparison with the real world scene, it is also possible to take the overlap of sensor data in consideration. For example, different point clouds should translate the same psychical obstacles in the same place: Figure 4.11 translates the two point clouds of the two LiDARs before and after the small correction in each sensor orientation. In the second image, we can see that both laser data are in tune since they describe the same real obstacles. It is possible to see here[11] how all this interactive first guess tool works.

[11] `https://youtu.be/zyQF7Goclro`



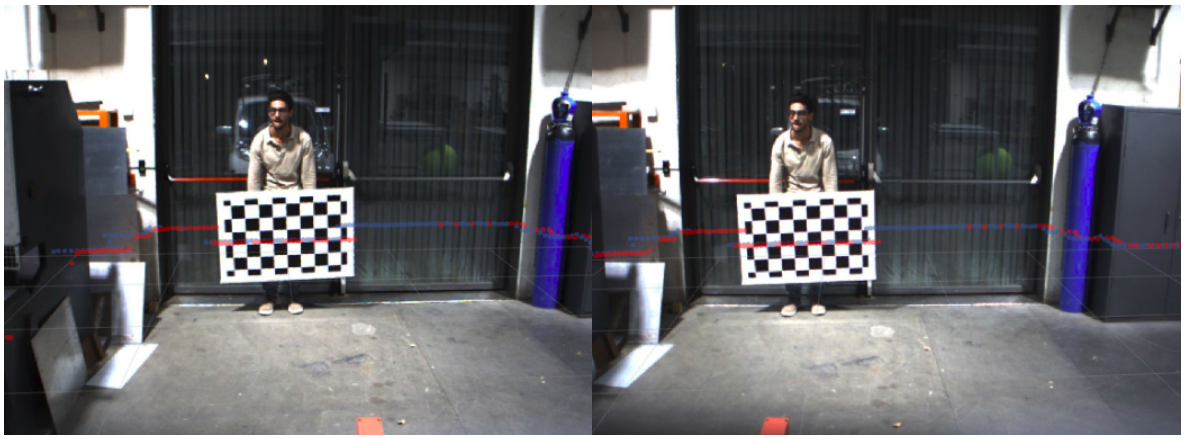
(a)



(b)

Figure 4.11: ATLASCAR2, the interactive markers and the laser data in RViz environment: a) before correcting sensor pose; b) after correcting sensor pose.

In the image of both cameras, the two different point clouds are also enabled (Figure 4.12), providing another excellent real time feedback resource so that the user could reach to the best possible sensor configuration.



(a) (b)

Figure 4.12: Point clouds displayed in camera images: a) top left camera; b) top right camera.

In an overall view, the biggest change in the sensor pose is made by comparing the model in RViz with the real world system, and then there are some slight adjustments to be made in accordance to the data real time feedback. Following this, it is expected that by ensuring an accurate first guess for the sensor poses, the optimizer will start searching closer to the best parameters configuration, and, thus, there is less likelihood of running into local minima.

After interactively defining the pose of the sensors, this first guess configuration is saved in a new xacro description file, so that the original robot description is never lost. From now on, this new description with the first guess will be read. Note that, at any time within this step, the user could return to the original configuration.

Summing up, in order to avoid the local minima problem of optimization procedures, several interactive tools were developed which allow the user to more easily produce a more accurate initial estimate of the pose of the sensors. This accurate first guess is achieved by a human interactive tool with big advantages, such as:

- being user friendly thanks to its human intuitive way of working;
- allowing the comparison, instantaneously, with the real physical robot system;
- allowing the real time feedback from the sensors data.

4.3 Step 3: Interactive Data Labeling

With the two previous steps of the calibration setup, the geometric transformations, between all the robot system *tf* tree, that are going to be optimized are already provided. That is, the translation and orientation specifications of each transform will be the parameters to optimize, as will be further detailed. Beside knowing which they are, the values of these parameters are not so far from the real configuration which makes it unlikely for the optimization procedure to run into a local minima problem.

4.3.1 Choice of Calibration Pattern

In order to analyse the accuracy of the sensor pose, and to compute the associated error, it is necessary to collect information about the sensor data. Since the main goal of this project is to calibrate multi-modal sensors of a robot system, a calibration pattern that could be recognized by all type of sensors must be used.

A usual pattern used for calibrating RGB and RGB-D cameras is a chessboard. LiDARs only detect 2D shapes of the objects in the surroundings. That means that any planar surface would be recognized, or, for example, any spherical or cylindrical object.

Taking this into account, we conclude that the chessboard is one of the best solutions for the purpose of same pattern detection by all different sensor modalities.

4.3.2 Interactive Labeling of Data

There are already several chessboard detectors for image data. In this work, the *Find Chessboard Corners* OpenCV function[12] is used.

For 2D LiDAR data, there is not a direct tool to recognize the points related to the chessboard. As can be seen in Figure 4.11, there are multiple planes in the scene because of the walls and the doors detected by the laser scan. In order to solve this issue, an interactive way of indicating which are the points, from the complete point cloud, that belong to the chessboard and, thus, are meant to be saved, was created. The interactive data labeling procedure is shown in Figure 4.13.

[12] `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#findchessboardcorners`

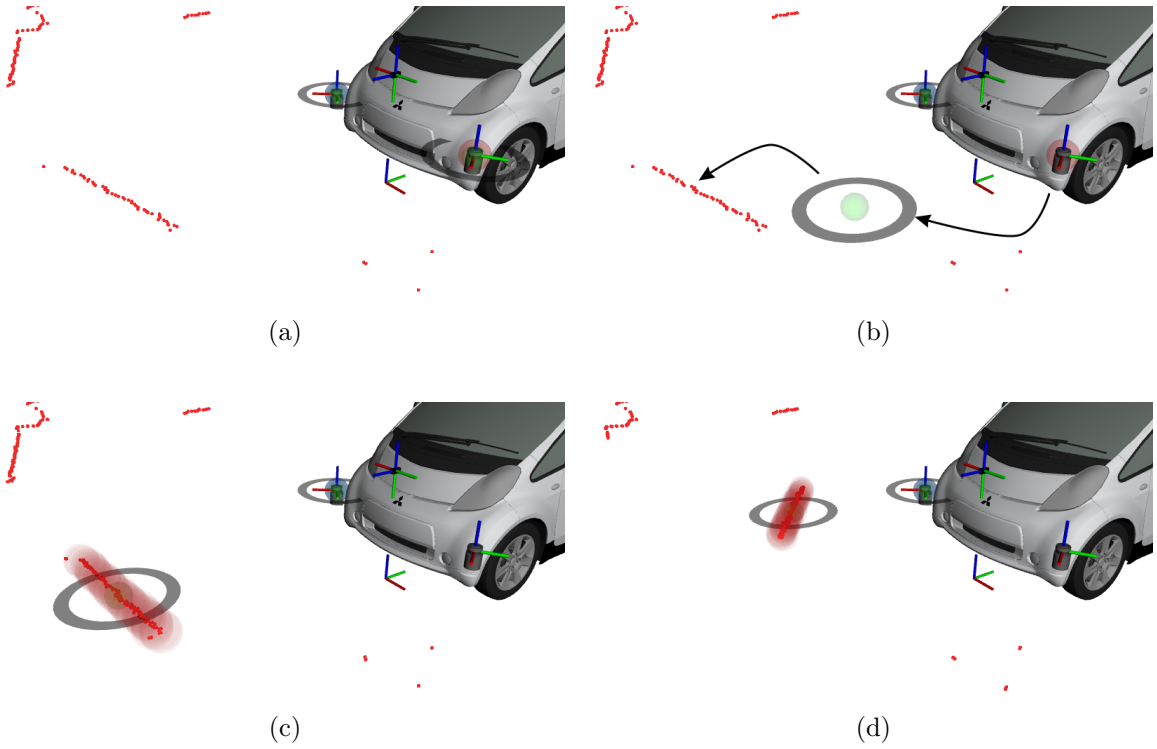


Figure 4.13: Interactive Data Labeling procedure: a)interactive markers creation; b) dragging the markers to the chessboard cluster; c) the marker selected the closest cluster; d) the marker tracks this same cluster over time.

The user should manually select which one of all the point segments translate the chessboard position. For that, once again, RViz interactive markers appear to be the best solution to implement: they start in the same position of the sensor, along each 2D LiDAR data (Figure 4.13 a)). Then, the user should drag the marker to the related chessboard measurement points. By clustering the LiDAR measurements and knowing that the distance between the limit of the chessboard surface and the background scene is big enough to be visually detected, it turns to be very easy for human interactive choosing the correct point cluster (Figure 4.13 b)). The marker will stick to the closest cluster (Figure 4.13 c)) and, from the moment that the user drop the marker, it will move according to the cluster (see[13]). Along the chessboard movement, the correspondent point group of the 2D LiDAR data is moving too, and so does the interactive marker that is connected to this cluster (Figure 4.13 d)).

In this way, the human intervention to label the scan laser data is only required once: at the beginning of this calibration configuration step. From that moment on, it is possible to track the chessboard robustly, because, even if the number of measurements that compose the cluster changes with time (which will happen due to the distinct chessboard orientations and distance to the sensor), the interactive marker will always be searching for the closest

[13] `https://youtu.be/9pGXShLIEHw`

cluster to consider. This approach is a semi-automatic process, since the first definition of which is the interest cluster is done manually and the over time tracking of this same cluster is automatic.

This data labeling procedure works for any number of 2D LiDARs (and RGB or RGB-D cameras): in the ATLASCAR2 case of study, there are two lasers which mean that it will be two interactive markers, one for each point cloud (each LiDAR data), needing, both of them, the manual selection of the chessboard detected measurements. In Figure 4.13, we can see this interactive LiDAR data labeling for just the left laser and its red point cloud data.

Summarizing, in order to compute the value of the optimization objective function, some data information is requested to test the parameters to optimize (geometric transformations specifications). A multi-modal sensors calibration requires a calibration pattern that could be detected by all different sensor modalities: this is the only way to enable the correspondence of the same detected object by all collected data information. The advantages of the proposed calibration pattern and the interactive data labeling procedure are that this approach provides:

- Automatic detection for RGB cameras;
- A very intuitive framework suported by the RViz environment;
- Semi-automatic chessboard detection for 2D LiDARs.

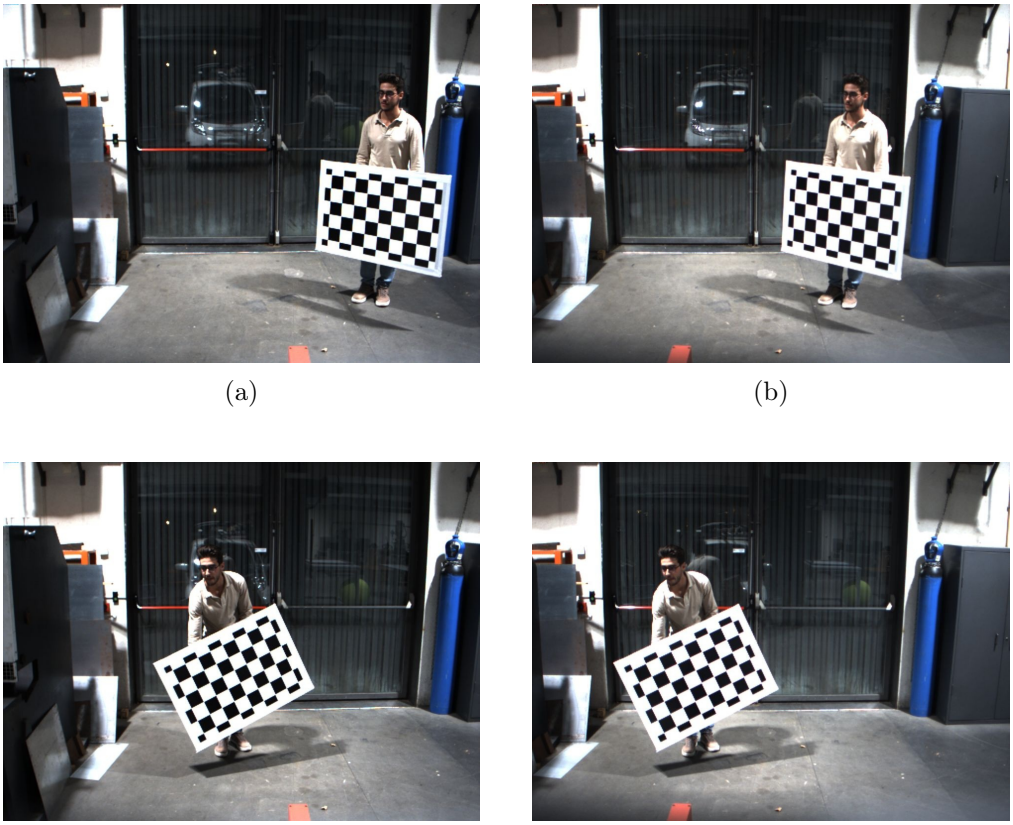
4.4 Step 4: Interactive Collection of Data

After adding the recently created *sensor* element to the robot description, all robot XML files were parsed in order to find those sensors and enable the creation of interactive markers, in RViz environment, related to each ATLASCAR2 sensors. In an interactive way, all sensors poses were improved, putting them closer to the real world situation, and all sensors data were labeled, providing the information about which measurements are concerned with the chessboard detection.

Now, all of this material must be gathered and saved in an accessible format, in order to be used afterwards by the optimization. Since the optimization procedure will work on the re-projection error (as will be detailed in chapter 5), it is necessary to acquire multiple sensor data from the exact same scene frame, i.e., for the ATLASCAR2 case, all of the 2D LiDAR point clouds and the camera images, that are going to be saved, must describe the same real world chessboard pose. For solving this concern, temporal synchronization and equal stream frequency between all sensors is required.

The tool created for saving the information is interactive. With the real time feedback of the point clouds of each laser scan and of the camera images, the user can see if, at the captured moment, all sensors are detecting the chessboard at the same world referenced pose. To guarantee this, at the time of having the chessboard, in different positions and orientations, in front of the sensors, it was payed a particular attention to let the chessboard static for a few seconds in each distinct configuration. This will facilitate the job when it comes to saving the sensors data of the same real scene. The few static chessboard seconds are enough to nullify the effect in data content caused by the difference between the streaming frequencies of all sensors (if there were any). Figure 4.14 demonstrates two captured moments (13 and

18) by both cameras: top left and top right. As can be seen, for each moment, both cameras are detecting the same real world frame.



(a) (b)

(c) (d)

Figure 4.14: Images detected by both cameras at two distinct moments: a) top left camera at moment 13; b) top right camera at moment 13; c) top left camera at moment 18; d) top right camera at moment 18.

In order to ensure an accurate pose estimation, the chessboard pattern should be detected in as many distinct distances and orientations as possible. In fact, most of the calibration procedures must take this concern into consideration, so that they can be well succeeded. This is why the chessboard should be moved around and the user should save several *collections*. *Collection* is the proposed term for the information saved at the same time stamp. This saved information is composed by ROS messages, that are constantly being published, as well as the camera images (see[14]).

The developed project creates a Python dictionary [15] where all the contents, from the ros messages, that are necessary for the optimization procedure are stored. To convert ROS messages to Python dictionary, the *rospy_ message_ converter* ROS package and Python library[16] was used.

[14] `https://youtu.be/9pGXShLIEHw`
[15] `https://realpython.com/python-dicts/`
[16] `http://wiki.ros.org/rospy_message_converter`

The dictionary is saved in a JSON file that will be accessed by the optimizer. This JSON stored dictionary has two keys:

```
- collections {29}
- sensors {4}
```

The *collections* key is composed by several dictionaries with data about the different moments of data collection.

In the studied case of the ATLASCAR2 calibration, 29 collections were taken. Each collection key is another dictionary with three keys:*data*, *labels* and *transforms*. The value of *data* key is another dictionary with four keys corresponding to the four sensors. *data* field provides information, among other, about the detected data of each sensor (intensities and ranges measured by all points of the LiDARs, size and source of the image captured by both cameras):

```
- collections {29}
    - 0 {3}
    - 1 {3}
        - data {4}
            - left laser {10}
            - right laser {10}
                - angle_increment : 0.0087
                - angle_max : 2.3562
                - angle_min : -2.3562
                - header {3}
                - intensities [541]
                - range_max : 20
                - range_min : 0.01
                - ranges [541]
                - ...
            - top_left_camera {7}
            - top_right_camera {7}
                - data_file :
                - header {3}
                - height : 724
                - width : 964
                - ...
        - labels {4}
        - transforms {6}
    - 2 {3}
    - ...
- sensors{4}
```

The *label* key in each *collection* is where the chessboard detected points of each sensor are, if there are any. Each sensor has a *detected* boolean value that is *True* when the calibration pattern is recognized and *False* when it is not. In case of success, all the distances detected from the laser to the chessboard plane (for LiDARs) and all the pixels related to the chessboard

detected corners (for cameras) are stored. If the *detected* value is *False* for a specific collection, then that same collection is neglected from the optimization process. Note that the number of pixels for both cameras has to be the same, assuming that there is the same number of chessboard corners. LiDARs data does not have this feature, since each one could have distinct number of points in the chessboard detected cloud:

```
- collections {29}
    - 0 {3}
    - 1 {3}
        - data {4}
        - labels {4}
            - left laser {2}
                - detected :
                - idxs [33]
            - right laser {2}
                - detected :
                - idxs [40]
            - top_left_camera {2}
                - detected :
                - idxs [54]
                    - 0 {2}
                    - 1 {2}
                    - ...
            - top_right_camera {2}
                - detected :
                - idxs [54]
        - transforms {6}
    - 2 {3}
    - ...
- sensors{4}
```

The robot could be moving during the process of collecting data. The transformations between the links that belong to each sensor chain in the robot *tf* tree are stored in every captured moment (pre-transforms, transform to optimize and post-transforms, as mentioned in 4.1). This is the proposed way of giving all the information that the optimizer needs to compute the error associated to the calibration pattern detection by each sensor at each moment. All transformations are defined by the three translation values and the four quaternions:

```
- collections {29}
    - 0 {3}
    - 1 {3}
        - data {4}
        - labels {4}
        - transforms {6}
            - base_link-left_laser {2}
                - quat [4]
                - trans [3]
```

```
        - base_link-right_laser {2}
        - base_link-top_left_camera {2}
        - base_link-top_right_camera {2}
        - top_left_camera-top_left_camera_optical {2}
        - top_right_camera-top_right_camera_optical {2}
```

Beside the *collection* key, there is a *sensor* key, in the main Python dictionary, which stores time independent information. The value of *sensor* key is another dictionary composed by the four sensors, where each one has its name, calibration parent link and calibration child link (mentioned in the novel *xacro* created in 4.1), the *tf* tree link chain, the message type, the parent link (link related to the sensor) and the ros topic where all detected data by each sensor is being published:

```
- collections {29}
- sensors{4}
    - left_laser {7}
    - right_laser {7}
        - _name:
        - calibration_child:
        - calibration_parent:
        - chain [1]
        - msg_type:
        - parent:
        - topic:
    - top_left_camera {9}
    - top_right_camera {9}
        - _name:
        - calibration_child:
        - calibration_parent:
        - camera_info {11}
        - camera_info_topic:
        - chain [1]
        - msg_type:
        - parent:
        - topic:
```

Note that the cameras have two more saved parameters: *camera_info*, where the intrinsic parameters, computed at the beginning of the procedure with the OpenCV calibrate tool (see the beginning of chapter 4), are saved and *camera_info_topic*, the ROS topic where the collected intrinsic values are being published.

Summing up, in order to have a reliable calibration, it is necessary to have data related to different calibration pattern poses. The user has the responsibility to choose, with real time feedback from all detected sensor data, which are the best moments to capture information. The stored information is composed by listened ROS messages and images captured by both cameras. The ROS messages, that were continuously published, in specific ROS topics, are organized and saved in a human readable JSON file, to be later accessed by the optimization procedure. This proposed approach of collecting the data has the advantages of:

- handling the problem of different stream frequencies for the sensors;
- being, once again, interactive with real time feedback (thus, easy to use);
- storing all important information well organized in just one file that is human readable.

This chapter presented all developed features that are required in the calibration procedure, since they are responsible for collecting the necessary information for the optimization of the sensors pose. The detailed methodology allows the robot system, that is being calibrated, to have several sensors with different modalities, and they all are taken into account at the same time. Not only this, but the transformations that are going to be optimized can be anywhere in each correspondent sensor path, not destroying neither compromising the original composition of the system. These are the two big news for the robot development and research world. In addiction, the majority of the setup procedure is done in an interactive way, always with real-time feedback. This brings many advantages already mentioned above.

For ATLASCAR2, in the first place, after finding the first guess of the intrinsic parameters of both cameras with the ROS Camera Calibrator a ROS bag file was saved. The ROS bag file contains all the information that was being published, by the working ROS nodes, at the time that it was recorded. Afterwards, it can be reproduced and ROS environment will operate regularly and so will RViz. This work method simplifies a lot all the setup procedure because it avoids the concern of having two persons working on the scene: one for moving the chessboard in front of the sensors and the other to taking care of the interactive actions (defining the pose first guess, data labeling and data collecting). This small detail helped the development of this project a lot since it enabled the possibility of working outside the ATLASCAR2 real world environment.

It is important to notice that all this explained calibration configuration procedure and the optimization that will be approached in the next chapter were developed with the contribution of the other authors of [19].

Before diving into the optimization procedure, Figure 4.15 exhibits a summary schema of the calibration setup proposed approach.

When the JSON file with the collected data and the camera images are ready, it is time to optimize all sensor poses.

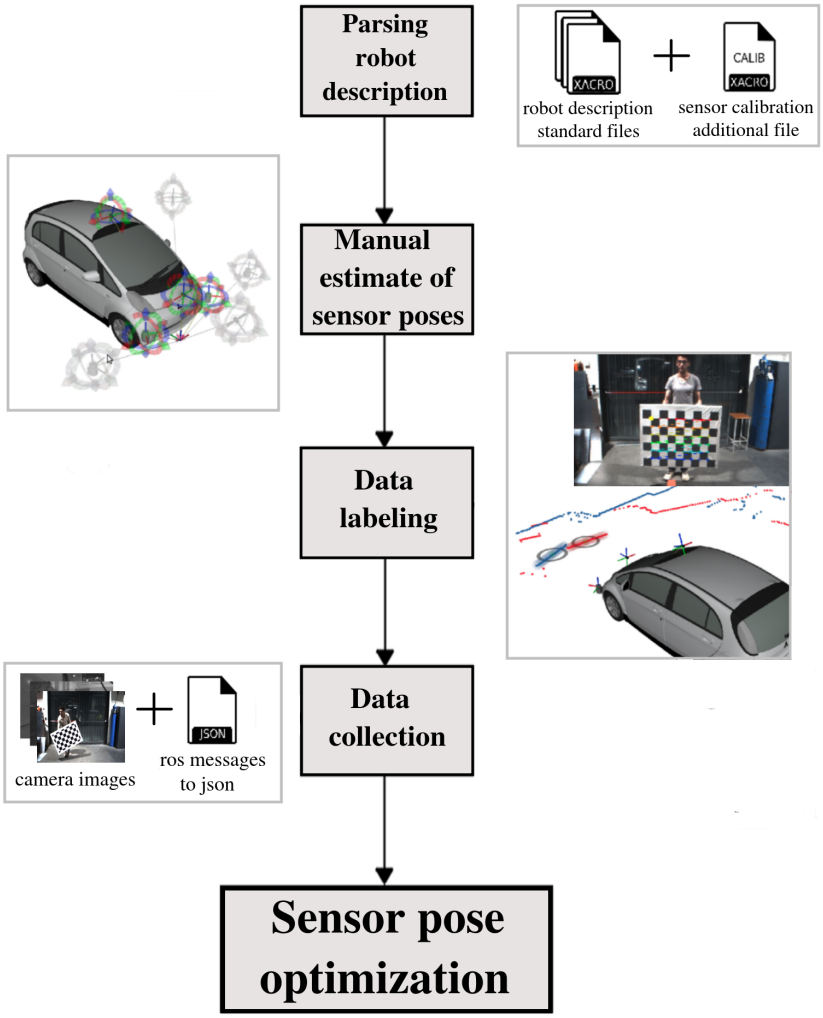


Figure 4.15: Summary schema of the four steps of the calibration setup configuration that leads to the optimization procedure.

Chapter 5

Optimization Methodology

The goal of optimization procedures is to find the parameter values that return the smallest error. The result is derived from a function, created for the purpose. This function is known as the *objective function*.

The desired function results by the optimizer could be the minimum or the maximum, depending the problem nature. Usually, when the searched is the minimum, the objective function is also called *cost function*, and the procedure is called a minimization.

For the purpose of calibrating all multi-modal ATLASCAR2 sensors, the objective of this optimization is to estimate the position and orientation of each particular sensor.

This chapter will detail the optimization methodology used, and explain all the elements that compose it. This optimization is an adaptation of the already developed methodology in [17], as was said before.

5.1 Optimization Parameters

An extrinsic calibration translates into a pose estimate. That means that the ambition is to accurate the position and orientation parameters of each sensor.

The position, relatively to any other link of the robot description, is characterized by three values: x-axis translation, y-axis translation and z-axis translation. These translation values compose the translation vector (3 x 1 dimension vector):

$$t = \begin{bmatrix} tx \\ ty \\ tz \end{bmatrix}. \tag{5.1}$$

The orientation, relatively to any other link, could be represented in several ways: 3 x 3 matrix, axis/angle parameterization, quaternions, an others. It is possible to translate these different formulations together. The three values of axis/angle parameterization to characterize each rotation between the *calibration_parent* link and the *calibration_child* link were used as optimization parameters. So, as the translation vector, the rotation vector is

composed by these three values (3 x 1 dimension vector):

$$r = \begin{bmatrix} r1 \\ r2 \\ r3 \end{bmatrix}. \tag{5.2}$$

The optimization parameters must, for any optimization, be linearly independent between each other. Since the rotation of a robot link has three degrees of freedom (because of the three axis), it is not possible to optimize more than three parameters. In the case of having more than three values to represent a rotation, it means that some of them depend on other parameter value, which could make it impossible for the optimizer to get to the best solution.

The axis/angle parameterization was chosen because it has 3 components: two values to define an axis and one value to define an angle, making it a fair parameterization [12].

Thus, for each sensor, since the objective of the extrinsic calibration is to get as close as possible to the real pose, these six values are due to be optimized and, therefore, are included in the optimization variables.

As seen in 4.4, the JSON file that the optimizer will import and read, in order to have the necessary material to start the process, has all transformations described by the translation vector and the rotation vector in quaternions. For that reason, it is essential to translate the quaternion representation to the axis/angle parameterization.

At this point, there are six parameters per sensor to be enhanced. These values compose the geometric transformation that will be calibrated, as was explained in 4.1.

The cost function will compute the residuals based on an error (in pixels for RGB cameras and in millimeters for LiDARs) between the re-projected position of the chessboard, estimated by all transformations, and the position of the calibration pattern detected by each sensor.

In order to project a 3D point to a sensor data, it is required to know the coordinates of that point relatively to the sensor frame. The goal of the present work is to calibrate several sensors with distinct modalities. For that to be done all at the same time, the information about the 3D coordinates of the chessboard corners should not only be relative to one sensor link but should represent the pose of the calibration pattern with respect to the world. That is the only way of estimating the pose of each sensor and the chessboard in the same scene, since they all are described relatively to the *base_ link* link.

Thus, the pose of the calibration pattern must also be optimized, since it has, physically, to be the same, for all sensors, in each collection. All of this approach is detailed in the section 5.2.

Identical to the sensors configuration, the chessboard pose will also be characterized by the six parameters. It is important to remember that several collections were saved, where the chessboard was always with different positions and orientations. For that reason, the optimizations parameters, beside the pose values of each sensor, also include the values of each collection pose of the chessboard.

In this case, 29 different moments were collected, which results in 174 optimization variables only related to the chessboard.

For the specific case of cameras, to enable the projection of a 3D point to the image detected by the sensor, it is indispensable to know the intrinsic matrix and the distortion parameters. In order to get to the best possible results, the calibration procedure also enhance the value of these parameters. This means that for each camera sensor type, besides the six variables that express the translation and the rotation between the *calibration_parent* link and the *calibration_child* link, there is also nine more optimization variables: four intrinsic parameters and five distortion parameters.

The focal length (fx and fy) and the principal point offset (cx and cy) values compose the intrinsic matrix of a camera:

$$K = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.3}$$

The distortion vector has the five distortion coefficients:

$$D = \begin{bmatrix} d1 & d2 & d3 & d4 & d5 \end{bmatrix}. \tag{5.4}$$

All variables must have some initial value, so that the optimizer could compute the first error and start to refine them in order to get to the minimum of the cost function. The first guess of the sensors pose was defined as explained in section 4.2. The first guess of camera sensor intrinsic values was taken from the ROS Camera Calibrator, as mentioned at the beginning of chapter 4. This first guess information was saved in the JSON file: every single interactive estimated position and orientation of each sensor and the camera sensors intrinsic parameters and distortion coefficients. The optimization methodology starts by accessing all data in the stored dictionary.

The first guess of the chessboard position, for each collection, is not present in the JSON file. In fact, it will be computed right before the optimization starts. The calibration pattern pose guessing is detailed in the next section.

Summing up, the set of optimization parameters Φ, has the following arrangement:

$$\begin{aligned} \mathbf{\Phi} = \Big[& \overbrace{\mathbf{t}_{m=1}, \mathbf{r}_{m=1}, \mathbf{i}_{m=1}, \mathbf{d}_{m=1}, \ldots, \mathbf{t}_{m=M}, \mathbf{r}_{m=M}, \mathbf{i}_{m=M}, \mathbf{d}_{m=M}}^{\text{Cameras}}, \\ & \overbrace{\mathbf{t}_{n=1}, \mathbf{r}_{n=1}, \ldots, \mathbf{t}_{n=N}, \mathbf{r}_{n=N}}^{\text{LiDARs}}, \quad \overbrace{\mathbf{t}_{k=1}, \mathbf{r}_{k=1}, \ldots, \mathbf{t}_{k=K}, \mathbf{r}_{k=K}}^{\text{Calibration objects}} \Big], \end{aligned} \tag{5.5}$$

where m refers to the m-th camera, of the set of M cameras, n refers to the n-th LiDAR, of the set of N LiDARs, k refers to the chessboard detection of the k-th collection, contained in the set of K collections, $\mathbf{t}$ is a translation vector $[t_x, t_y, t_z]$, $\mathbf{r}$ is a rotation represented through the axis/angle parameterization $[r_1, r_2, r_3]$, $\mathbf{i}$ is a vector of a camera's intrinsic parameters $[f_x, f_y, c_x, c_y]$, and $\mathbf{d}$ is a vector of camera's distortion coefficients $[d_0, d_1, d_2, d_3, d_4]$.

5.2 Cost Function

The scalar cost function of this optimization is the sum of the squares of the returned values from a vector function, divided by two. Each different sensor has an inherent sub-function, that depends the sensor modality. The value of all these sub-functions is a vector with the errors (residuals) associated to the re-projection of the calibration pattern points. The sub-functions will be detailed in the next sections. By putting all sub-functions together, a new vector function is born, that embraces the errors of all studied sensor pose.

In the ATLASCAR2 case, since it has four sensors (two cameras and two LiDARs), the objective function is composed by the vector values of four sub-functions, two of each type.

The optimizer will search for the best set of parameters Φ that minimize the error, related to each sensor pose, returned by each sub-function. Thus, the cost function could be written as:

$$\begin{aligned} \min_{\mathbf{\Phi}} F(\mathbf{\Phi}) &= \frac{1}{2} \cdot \sum_{r=1}^{R} f_r(\mathbf{\Phi})^2 \\ f(\mathbf{\Phi}) &= \left(f^{\text{ camera 1}}, f^{\text{ camera 2}}, ..., f^{\text{ lidar 1}}, f^{\text{ lidar 2}}, ...\right) \\ f^{\text{ camera}}(\mathbf{\Phi}) &= (f_1, f_2, ..., f_C) \\ f^{\text{ lidar}}(\mathbf{\Phi}) &= (f_1, f_2, ..., f_L), \end{aligned} \tag{5.6}$$

where r refers to the r-th residual, of all the R residuals, f is the vector function that congregates the errors of all sensors sub-functions, f_r is the value of the r residual, $f^{\text{ camera}}$ is the cost sub-function for cameras that return a vector with C residual values, $f^{\text{ laser}}$ is the cost sub-function for lasers that return a vector with L residual values.

So, for instance, for a robot system with n cameras and m lasers, the number of total residuals R it would be $R = n \cdot C + m \cdot L$. Even though they are two different sub-functions, the first concept to be computed, in both of them, is the sensors poses relatively to the reference frame. The explanation of how this is done is right below. A geometric transformation is represented by a matrix with the translation and rotation parameters. By definition, this matrix has a size of 3 x 4 but, in order to be multiplied by others transform matrices, it could be homogenized, turning it into a 4 x 4 matrix:

$$T = \begin{bmatrix} r11 & r21 & r31 & tx \\ r12 & r22 & r32 & ty \\ r13 & r23 & r33 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{5.7}$$

The purpose of the optimization is to accurate the pose of each robot's particular sensor. The position and orientation of each sensor is given by the geometric transformation from the reference link (*base_ link* for the example of ATLASCAR2) to the sensor link (*parent* link).

This transform is simply obtained by orderly multiplying all the partial transformations that constitute the *tf* path of the pretended sensor. In other words, the chain of transformations, extracted from the topology of the *tf* tree, composes an aggregate transformation that indicates the sensor pose relatively to the world reference frame.

When it comes to describe a geometric transformation from a link A to a link B, the usual representation is ${}^{A}\mathbf{T}_{B}$.

As we had seen in 4.1, each sensor chain is composed by the pre-transforms, the transform to be optimized, and the post-transforms. The pre-transforms are those which describe the position and orientation of the *calibration_parent_link* from the reference link; the post-transforms express the sensor pose from the *calibration_child_link* and the transformation to be calibrated is that relating the *calibration_child* link to the *calibration_parent* link.

Generally, the aggregated geometric transformation that characterizes the sensor positioning from the world frame is given by:

$$ {}^{\text{world}}\mathbf{T}_{\text{sensor}} = \prod_{i=\text{world}}^{\text{sensor}-1} {}^{i}\mathbf{T}_{i+1} = \overbrace{\prod_{i=\text{world}}^{\text{parent}-1} {}^{i}\mathbf{T}_{i+1}}^{\text{prior links}} \cdot \overbrace{{}^{\text{parent}}\mathbf{T}_{\text{child}}}^{\text{to be calibrated}} \cdot \overbrace{\prod_{i=\text{child}}^{\text{sensor}-1} {}^{i}\mathbf{T}_{i+1}}^{\text{later links}} , \tag{5.8} $$

where ${}^{i}\mathbf{T}_{i+1}$ represents the partial transformation from the i-th to the i-th $+1$ link, and *parent* and *child* are the indexes of the *calibration_parent* and *calibration_child* links in the sensor chain, respectively.

Notice that the optimization variables are only the ones that describe the partial transformation written in blue in Equation 5.8. Since the sensor pose w.r.t. the world link also depends on this transformation, the cost function can compute the re-projected error, as will be explained further. The fact that the values to accurate by the optimizer are chosen by the user allow to maintain the structure of the original robot description.

The transformation that is actually being optimized could relate two consequent links or more links of the chain, and, besides that flexibility, the remaining transformations (pre and post) could be fixed, if the robot is static, or could be different over time, since they were all saved in each distinct time stamp (*collection*), as was already mentioned in 4.4. This is one of the features that contributes to make the proposed approach to the calibration general for every type of robots.

Before the first call of the cost function, it is essential to have a first value of all the optimization parameters. The optimization variables related to the sensors already have their first guess, as was explained in the previous section.

In order to get the calibration pattern pose relatively to the reference frame, for each collection, a chessboard detector was used for the images saved of one of the available cameras. More specifically, when the optimizer is launched, after it accesses and reads all the collections information (JSON file containing ROS messages data and the images detected by

both cameras in each capture moment), it will use *findChessboardCorners*[1], *cornerSubPix*[2] and *solvePnP*[3] functions of OpenCV library, so that the transform from the camera to the chessboard, for each collection, could be computed.

For ATLASCAR2, since the transformation from the camera sensor to the *base_link* (reference frame) is already known (Equation 5.8), it is possible to find the relation between the chessboard and the *base_link* by just multiplying the transformation from the camera to the calibration pattern computed right before:

$$ {}^{\text{world}}\mathbf{T}_{\text{chess}} = \overbrace{{}^{\text{world}}\mathbf{T}_{\text{camera}}}^{Equation\ 5.8} \cdot \overbrace{{}^{\text{camera}}\mathbf{T}_{\text{chess}}}^{\text{chess detection}}, \tag{5.9} $$

By having all of these three transforms, the relation between the pose of each sensor to the chessboard, for each collection, is easily figured out.

Summarizing, before the optimization procedure starts, all the parameters have to be defined and, because of that, to estimate the first guess of the chessboard pose in the scene for each collection, one of the cameras is used.

After that, when the optimization is already running, the objective function is called, and, depending on the modality of the sensor in question, one of the two distinct sub-functions (created by the two different type of sensors) will be computed. Since ATLASCAR2 has four sensors, each time the objective function will be computed, four sub-functions will be gauged, before. Each different sub-function is explained in the next sections.

5.2.1 Camera Sub-Function

In the case of the considered sensor being a camera, the cost sub-function will measure the distance, in pixels, between each chessboard detected corner and the correspondent projected corner.

The data imported from the JSON file that was generated by the calibration setup stage (see section 4.4) contain, for each collection, the pixels values of all detected chessboard corners. In this project, the calibration chessboard has the size of 10 x 7 squares, which results in a 9 x 6 points grid. This means that, for each collection within each camera, 54 pair of pixels values, where each one represents the position in the image where was detected a chessboard corner, are stored. These points are mentioned as *ground_truth*, since there will be the reference for computing the error of the projected corners.

The parameters to optimize (camera pose, intrinsic parameters and chessboard pose) are required when it comes to project the 3D points to the camera image. For the projection procedure, it is necessary to know the 3D position of each corner relatively to the camera link.

If we see the board from the top, the origin of the chessboard frame is the first (top left) corner, and the x-axis has the direction form left to right, while the y-axis points from the

[1] `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findchessboard#cv2.findChessboardCorners`

[2] `https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=cornersubpix#cv2.cornerSubPix`

[3] `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=solvepnp#cv2.solvePnP`

top to the bottom of the board, which makes the z-axis point to the back of the chessboard. This is the calibration pattern frame which already has its first pose guessed right before the optimization started (see Equation 5.9).

By knowing the real size of the chessboard squares, the 3D coordinates of all corners relatively to the chess frame are directly known. Note that the z value will be, for every point, zero, since the board (where all the points are) is in the XoY plan.

After getting the 3D coordinates of all corners in reference to the chessboard frame, the objective function compute the coordinates of the points relatively to the camera link by just multiplying the geometric transformation between the *base_ link* (reference frame for the ATLASCAR2 example) and the calibration pattern frame and by the transform between the camera link and the *base_ link*:

$$\mathbf{p}_{camera} = {}^{\text{camera}}\mathbf{T}_{\text{world}} \cdot {}^{\text{world}}\mathbf{T}_{\text{chess}} \cdot \mathbf{p}_{chess} \tag{5.10}$$

where $\mathbf{p}_{chess}$ refers to the x, y, z coordinates of a chessboard corner, defined in the local chessboard coordinate frame, and $\mathbf{p}_{camera}$ refers to the x, y, z coordinates of the same chessboard corner, defined in the camera link. In fact, both $\mathbf{p}_{chess}$ and $\mathbf{p}_{camera}$ are the homogenized matrices of the coordinates so that Equation 5.10 could be mathematically correct. The homogenized coordinates matrices have dimension 4x1:

$$p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} . \tag{5.11}$$

The relation between the 3D coordinate points defined in the camera frame and the image pixels of these same points, taking into account the distortion parameters, are described in the following equations:

$$\mathrm{x}' = \mathrm{x}/\mathrm{z} , \tag{5.12}$$

$$\mathrm{y}' = \mathrm{y}/\mathrm{z} , \tag{5.13}$$

$$\mathrm{x}'' = \mathrm{x}' \cdot (1 + \mathrm{k1} \cdot \mathrm{r} + \mathrm{k2} \cdot \mathrm{r}^2 + \mathrm{k3} \cdot \mathrm{r}^3) + 2 \cdot \mathrm{p1} \cdot \mathrm{x}' \cdot \mathrm{y}' + \mathrm{p2} \cdot (\mathrm{r} + 2 \cdot \mathrm{x}'^2) , \tag{5.14}$$

$$\mathrm{y}'' = \mathrm{y}' \cdot (1 + \mathrm{k1} \cdot \mathrm{r} + \mathrm{k2} \cdot \mathrm{r}^2 + \mathrm{k3} \cdot \mathrm{r}^3) + \mathrm{p1} \cdot (\mathrm{r} + 2 \cdot \mathrm{y}'^2) + 2 \cdot \mathrm{p2} \cdot \mathrm{x}' \cdot \mathrm{y}' , \tag{5.15}$$

$$\mathrm{u} = \mathrm{fx} \cdot \mathrm{x}'' + \mathrm{px} , \tag{5.16}$$

$$\mathrm{v} = \mathrm{fy} \cdot \mathrm{y}'' + \mathrm{py} , \tag{5.17}$$

where $r = x'^2 + y'^2$, x, y and z are the coordinates of each chessboard corner defined in the camera frame, $k1, k2, p1, p2$ and $k3$ are the five distortion parameters that are optimized, fx, fy, px and py are the intrinsic parameters that are also optimized and, finally, u and v are the coordinates of the projected point in pixels. This projection of each chessboard corner from the 3D coordinates to the image pixels was inspired in the *Camera Calibration and 3D Reconstruction* documentation of OpenCV [4].

[4] https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

The vector returned by this sub-function is the distance between the coordinates of the detected corners and the coordinates of the projected corners. This distance error is computed using the Euclidean norm:

$$f^{\text{camera}} = \left(\ell^2(\mathbf{x}_{c=1}, \hat{\mathbf{x}}_{c=1}), \ell^2(\mathbf{x}_{c=2}, \hat{\mathbf{x}}_{c=2}), ..., \ell^2(\mathbf{x}_{c=C}, \hat{\mathbf{x}}_{c=C})\right) \ , \tag{5.18}$$

where ℓ^2 represents the Euclidean norm, c denotes the index of the chessboard corners, of all the C chessboard corners, $\mathbf{x}_c$ denotes the pixels coordinates of the measured points (given by chessboard detection), and $\hat{\mathbf{x}}_c$ are the pixel coordinates of the projected points:

$$\hat{\mathbf{x}}_c = \begin{bmatrix} u \\ v \end{bmatrix} . \tag{5.19}$$

The goal of the optimization is to minimize the re-projection errors f^{camera}, by optimizing the parameters that define the chess-to-world transformation, the parameters of one of the partial transformations that compose the aggregated world to camera transform, the intrinsic matrix parameters and the distortion that also influence the result of the cost function.

As is expected, the re-projected points should be closer to the *ground_truth* corners, during the optimization procedure.

Figure 5.1 shows the difference between the initial position of the chessboard corner projected from 3D world to the camera image and the final position of these same projected points.

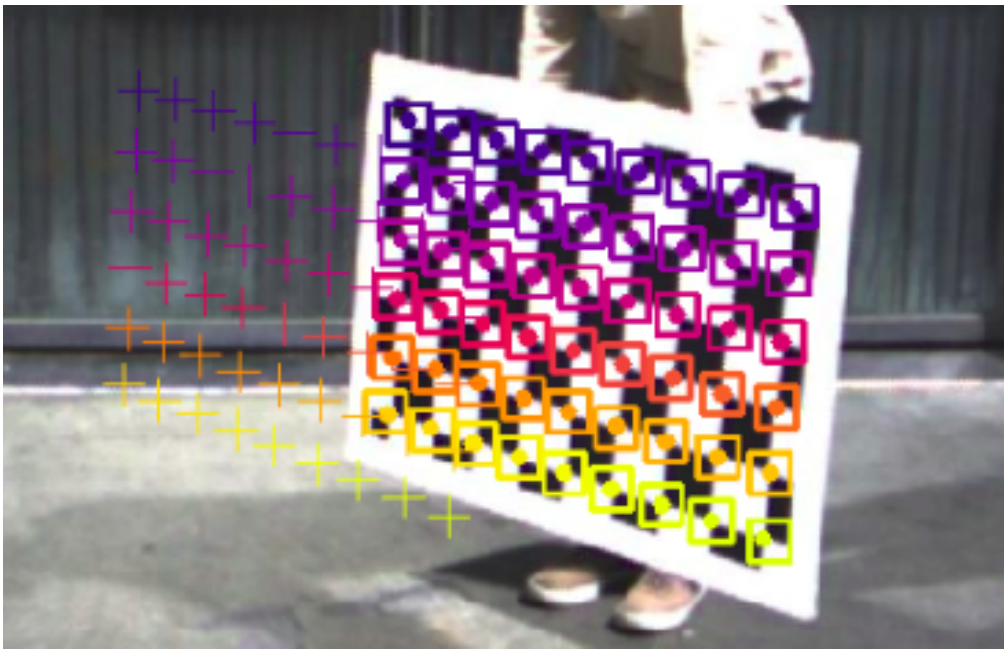


Figure 5.1: Portion of Top Right Camera image with: squares expressing the position of the detected chessboard corners (*ground_truth* points); crosses in the initial position of each projected corner from the 3D world to the camera image and round points expressing the optimized projected corners.

We can see that the final pixels of the projected points (dots in Figure 5.1) almost perfectly match the *ground_ truth* points (squares in Figure 5.1), as was expected from all the parameters being optimized.

5.2.2 2D LiDAR Sub-Function

For the case of 2D LiDARs, the sub-function only considers the two border points, between all the measurements that are related to the chessboard plan, to compute the error associated to the pose of the LiDAR and the chessboard.

In order to calculate the residuals that this cost sub-function should return, it is required to know the detected point 3D coordinates from the chessboard frame.

During the calibration setup stage, when the information of a time stamp is saved, as explained in section 4.4, the ranges of all measurements that the LiDAR is detecting are stored, as well as the information about this same LiDAR and the indexes of the ranges that correspond to the plan where the chessboard is. This means that, for each collection in the JSON file read by the optimizer, in each LiDAR sensor, there is the ranges of the calibration pattern plan measured points (labeled as mentioned in section 4.3) and the minimum angle and the angle increment characteristics of the LiDAR to take in consideration. These values are named *angle_ min* and *angle_ increment*, respectively, as seen in page 35 (section 4.4). This information allows to deduce the 3D coordinates of each measured point from the LiDAR frame.

Knowing that the point cloud detected by the 2D LiDAR is in the XoY plan of the sensor frame and that all angle information is relative to the semi positive x-axis, it is possible to compute the 3D coordinates of each point of the chessboard measurements in the LiDAR frame. The next equations translate the LiDAR measurements data into 3D points:

$$\rho \;=\; \text{ranges[idx]} \;, \tag{5.20}$$

$$\theta \;=\; \text{angle\_min} + \text{angle\_increment} \cdot \text{idx} \;, \tag{5.21}$$

$$\begin{bmatrix} \text{x} \\ \text{y} \\ \text{z} \end{bmatrix}^{\text{lidar}} \;=\; \begin{bmatrix} \rho \cdot cos(\theta) \\ \rho \cdot sin(\theta) \\ 0 \end{bmatrix} \;, \tag{5.22}$$

where ρ represents the closest distance from the origin of the LiDAR frame to the point in 3D world (range), θ describe the angle between the semi positive x-axis and the line that connects the LiDAR origin frame to the point in the point cloud, *ranges*, *idx*, *angle_ min* and *angle_ increment* are all the information previously explained that come from the created dictionary by the calibration setup stage and, finally, x, y and z are the 3D coordinates of each point from the sensor link.

Note that the z value is always zero since all the measured points belong to the XoY plan of the LiDAR frame.

With the optimization parameters of the chessboard pose relative to *base_link* and the LiDAR partial transformation that also allows to compute the pose of the sensor relatively to *base_link* (accordingly to Equation 5.8), the 3D coordinates of each labeled measurement of the point cloud in the chessboard frame is easily known:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{\text{chess}} = {}^{\text{chess}}\mathbf{T}_{\text{world}} \cdot {}^{\text{world}}\mathbf{T}_{\text{lidar}} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{\text{lidar}}, \tag{5.23}$$

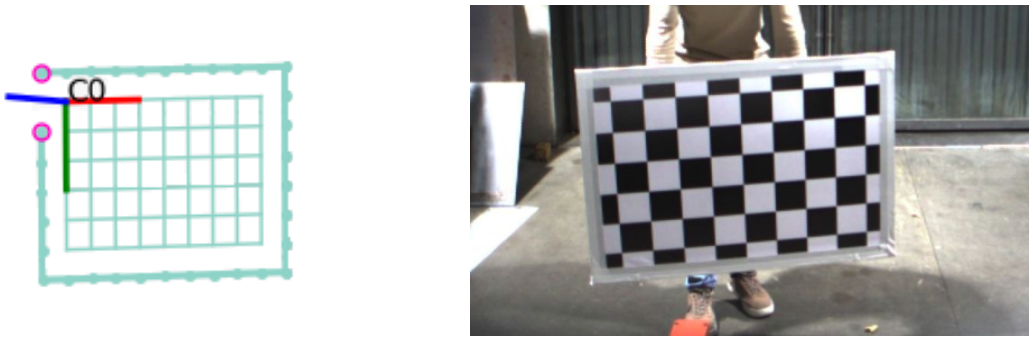
Finally, with the coordinates, from the chessboard frame, of both first and last points of the saved cluster, it is possible to compute the error evaluated by this cost sub-function. The error is based on the distance between each one of the limit points (first and the last index) of the selected ranges and the chessboard surface boundaries.

There are two computed distances for each point: orthogonal and longitudinal.

The orthogonal distance is the z absolute value of the coordinates, in the calibration pattern frame, of the LiDAR data measurement. In an ideal setting, the z value should be zero, since the chessboard plan is on the XoY plan. That is why any value different than zero means that the optimization parameters (sensor pose and chess pose) are not correct yet.

The longitudinal distance is the Euclidean distance between the x and y coordinates, in the calibration pattern frame, of the LiDAR data measurement and the x and y coordinates of the closest point that belong to the limit of the physical board that is being detected. In order to compute this distance, it was essential to create a group of points that represented the boundaries of the chessboard. By knowing the size of the board, the size of each chess square, and that the chess frame origin match with the first (top left) chess corner, the coordinates were deduced and the points of the board boundaries were manually created.

The measure of every border between the chess corner grid and the end of the physical chessboard must be known so that this step could be implemented. In Figure 5.2, we can see the grid of the chess corners and a line all around: that line refers to the limit of the board. This solid line has some points within it, which are going to be compared to the LiDAR data measured ones.



(a) (b)

Figure 5.2: Chessboard: (a) graphics visualization of the created grid and boundary correspondent to the real chessboard; (b) image of the real chessboard.

Again, the optimizer will search for the closest limit point to each one of the studied

LiDAR data measurement coordinates and then compute the longitudinal distance.

Thus, the LiDAR sub-function f^{lidar} is defined as:

$$f^{\text{lidar}} = \left(|z_1^{\text{chess}}|, \ell^2(\mathbf{p}_1^{boardlimit}, \mathbf{p}_1^{chess}), |z_2^{\text{chess}}|, \ell^2(\mathbf{p}_2^{boardlimit}, \mathbf{p}_2^{chess}) \right) , \tag{5.24}$$

where

$$\mathbf{p}^{boardlimit} = \begin{bmatrix} x \\ y \end{bmatrix}^{\text{boardlimit}} , \tag{5.25}$$

$$\mathbf{p}^{chess} = \begin{bmatrix} x \\ y \end{bmatrix}^{\text{chess}} , \tag{5.26}$$

and z^{chess} is the third coordinate value of the range measurement points transformed to the chessboard's coordinate frame. Notice that each laser vector sub-function return four distance values. According to Equation 5.6, $L = 4$, since L denotes the returned residual number by each laser sub-function.

If these four distances are minimized and get closer to zero, then all the re-projected measured point coordinates will be in the plan of the chessboard and within the boundaries of the calibration pattern surface.

5.3 Residuals

The values returned by each vector sub-function that compose the cost function are the computed errors between the sensor detected data and some reference data. Since the goal of the optimization is to find the best parameter configuration in order to minimize the errors cost function, the returned values from the objective function are expected to be close to zero, at the end of the optimization. These errors are also called *residuals*.

In this work, as ATLASCAR2 has two different sensor modalities, two distinct functions were created.

For the case of the sensor in consideration being a camera, the values computed by the associated sub-function are the Euclidean distance between the pixels of each projected chess corner, from the real scene to the camera image, and the pixels of the correspondent, image detected, chess corner. This means that, for each image captured, there are 54 residual values (in pixel units), once the chessboard as 9x6 corners size.

These type of residuals are named by the collection of the image, the sensor that captured the image, and the number of the corner (0 to 53). For example, the error between the projected pixels and the detected ones of the 34^{th} chess corner of the image captured by the *top right camera* at the 8^{th} collection is named *8_ top_ right_ camera_ 34*.

On the other hand, for LiDARs case, the associated sub-function compute the orthogonal and longitudinal distances between the coordinates of the closest point of the chessboard boundary, determined by the chess pose guessing, and some sensor data measurements. The distances are calculated for the first and the last index of the labeled chessboard related cluster, which results in four residuals (two of point) for collection, in meter units, for each LiDAR.

The name of these residuals is composed with the collection number, followed by the sensor name, and zero, if it refers to the first point, or one, if it refers to the last one. For example, the returned value of the f^{lidar} for the 27^{th} collection of the cluster last measurement of the *left laser* is named *27_ left_ laser_ 1*.

At this point, we notice that the residuals of this optimization procedure have distinct units which could cause some trouble to the optimizer. In section 5.5, we explain how this issue is tackled. ATLASCAR2 sensors pose optimization project works with 3364 residuals and none of them depend of all the optimization parameters. Due to the dimension of the problem, some shortcut to help the optimizer to carry the procedure quicker and efficiently is required.

5.4 Sparse Matrix

A very useful tool to guide the optimizer during the optimization procedure is the sparse matrix. The sparse matrix provides the information about which parameters influence which residual values. Each line of this matrix is correspondent to a particular residual, and each column to a particular optimization parameter.

The values that compose the matrix are only zeros and ones. Each zero means that change of the associated column parameter doesn't affect the specific residual of that row. On the other hand, the ones mean that the residual of the row is influenced by the optimization parameter of the respective column. Here is a portion of the sparse matrix created for the purpose of calibrating all multi-modal sensors of ATLASCAR2:

	S_TLC_tx	**S_TLC_ty**	...	**S_RL_r3**	**C_0_tx**	...	**C_29_r2**	**C_29_r3**
0_TLC_0	1	1	...	0	1	...	1	1
0_TLC_1	1	1	...	0	1	...	1	1
...	...	...	...	...	...	...	...	...
0_TLC_53	1	1	...	0	1	...	1	1
1_TLC_0	1	1	...	0	1	...	1	1
...	...	...	...	...	...	...	...	...
28_TLC_53	1	1	...	0	1	...	1	1
0_TRC_0	0	0	...	0	1	...	1	1
0_TRC_1	0	0	...	0	1	...	1	1
...	...	...	...	...	...	...	...	...
0_TRC_53	0	0	...	0	1	...	1	1
...	...	...	...	...	...	...	...	...
29_TRC_53	0	0	...	0	1	...	1	1
0_LL_0	0	0	...	0	1	...	1	1
0_LL_1	0	0	...	0	1	...	1	1
1_LL_0	0	0	...	0	1	...	1	1
...	...	...	...	...	...	...	...	...
28_LL_1	0	0	...	0	1	...	1	1
0_RL_0	0	0	...	1	1	...	1	1
...	...	...	...	...	...	...	...	...
28_RL_1	0	0	...	1	1	...	1	1

where *TLC* stands for *Top Left Camera*, *TRC* for *Top Right Camera*, *LL* for *Left Laser* and *RL* for *Right Laser*.

The parameters to optimize are the position $[tx, ty, tz]$ and orientation $[r1, r2, r3]$ for each sensor and the chessboard. The prefix S means that the parameter refers to one of the sensors and the prefix C means that it is about the chessboard. Note that, for the cameras case, besides the position and orientation values, they also have nine more parameters: the intrinsic and the distortion coefficients. As stated above, the pose of the chessboard is also estimated, for each different collection. Therefore, each chessboard parameter has the number of the collection in consideration and the name of the extrinsic pose specification. For example, *C_17_tz* refers to the z coordinate (from *base_link* frame) of the chessboard origin point, in the 18^{th} collection (since the first collection is the collection number zero).

There are two type of residuals: the camera residuals and the lasers residuals. The residuals of the cameras are the errors of the projected points, and so, for each camera, the number of errors is the number of chess corners per collection. The lasers residuals are the orthogonal and longitudinal distances between each one of the two first and last measurements 3D coordinates from the selected data cluster and the two reference border limit points. Thus, there are four residuals per collection for each LiDAR.

ATLASCAR2 has 2 cameras and 2 lasers as sensors. For this work, the chessboard used has the dimension of 9x6 corners, and 29 collections were saved. Consequently, the sparse matrix has 3364 rows and 216 columns, which makes it a very large matrix, hard to be manually generated. Because of this reason, the sparse matrix is generated automatically, right after defining the optimization parameters, the objective function and the residuals. All these implemented optimization tools were referenced in section 3.2.

This matrix gives the information to the optimizer about which variables should adjust in order to lower a specific residual. This saves a lot of processing time, since it is not necessary to blindly change the parameters several times, hoping that the pretended residuals get to the minimum. This is a major advantage when it comes to optimizing a huge problem with many variables and errors as is the calibration of a complex robot system with abounding sensors.

5.5 Sensors Pose Calibration: Optimization

After having all the parameters to optimize, the cost function, the residuals and the sparse matrix defined, the returned value for the cost function can start to be minimized.

As was mentioned in chapter 2, there are several methods for optimizing different problem types. For the optimization approached in this project, the least-square method was used.

The cost function, $F(\mathbf{\Phi})$, is minimized by the least-square function of the optimize scipy library[5].

This method solves nonlinear problems and finds the local minimum of a scalar function, with the optimization parameters (function variables) constrained. More than that, least-squares requires the residuals as an m-dimensional real residual function of n real variables.

[5] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html`

All of these characteristics make this method the best possible choice for solving any complex robot system, and, thus, the ATLASCAR2 calibration problem, since it fits perfectly the objective of minimizing the cost function with all the information that was already treated and explained so far.

The least-square function of the optimize scipy library requires the cost function ($F(\mathbf{\Phi})$), the vector of the variables (optimization parameters, Φ), the parameter boundaries (infinite by default) and the sparse matrix, in order to start finding the minimum value of the cost function.

For using this minimization tool, it is essential to set some termination conditions and the differential step.

The differential step represents the relative step size that will change the parameter configuration tested in the previous iteration.

There are three termination conditions: function tolerance (*ftol*), variable tolerance (*xtol*) and gradient tolerance (*gtol*). The *ftol* looks for the moment when the differential of the objective function (F) became too small. The condition that satisfies the *ftol* is given by $\delta F < ftol \cdot F$. This means that the function derivative value of a specific variables configuration should be lower than the function value (in that parameter arrangement) multiplied by the *ftol* value chosen argument of the least-square function. This condition will be tested every time the objective function is computed. When the condition is satisfied, the optimization ends and the current parameter configuration is the one that results in a minimum value of the cost function. The *xtol* is the tolerance for termination by the change of the independent variables ($\mathbf{\Phi}$). The condition that *xtol* is hoping to satisfy is $\|\delta\mathbf{\Phi}\| < xtol \cdot (xtol + \|\mathbf{\Phi}\|)$. The *gtol* is the criteria related to the norm of the gradient. The optimizer will stop searching for more variable values when one of these conditions becomes true. When that happens, it means that the least squares founded a (local) minimum and, consequently, that concrete parameter values are the optimal ones.

It is important to remember that the first variable configuration tested is, already, not so far from the best possible solution. That was ensured by the interactive selected sensors pose first guess, explained in section 4.2. Knowing that the optimal solution is the real world scene arrangement, the position and orientation of the sensors could be manually improved, by the human user, in order to get as close as possible to that configuration. This good parameters first guess makes the least-square method find, not only a local minimum, but the cost function global minimum too.

Besides the termination conditions, *x_scale* tool of least-squares function was also used. As mentioned in section 5.3, the residuals have distinct units for different sensors modalities. Because of this, there is a huge difference between the residual values obtained by the camera cost sub-function and the residuals values obtained by the laser cost sub-function. The parameter *x_scale* is responsible for scaling all residuals. By using it, the returned constants by the objective function are all within the same values range, making the sensor pose estimating fair for all sensors. Without this tool, some sensor poses would be more important for the optimizer than the others.

Throughout the implementation of the optimization procedure, a visualization function was created. This function allows the value of each residual as well as the poses of each sensor and the chessboard to be seen in real time.

Figure 5.3 shows the position of each sensor and of the chessboard for collection 0 at the beginning of the optimization and Figure 5.4 shows the residual value graphic, after the optimization ended. In other words, the sensor configuration that can be seen in Figure 5.3 is the one saved in section 4.2 and that is going to be enhanced.

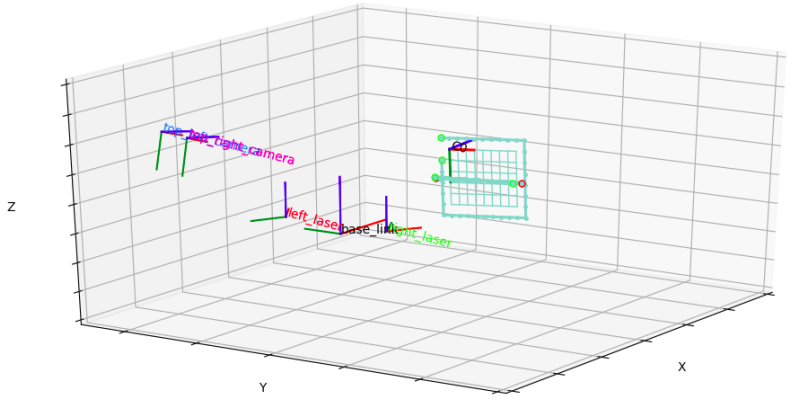


Figure 5.3: Representation of the optimized sensor poses and the estimated chessboard poses per collection.

Both of these graphics are shown during the optimization procedure and both of them are automatically updated every time the optimization parameters change and the respective new objective function value is computed. In Figure 5.4, we can see green dots and blue dots. The green dots stand for the values of each residual resulted from the first variable configuration. The blue dots describe the continuously updating residual values. Figure 5.4 exhibit the obtained residual graphic, at the end of the optimization procedure.

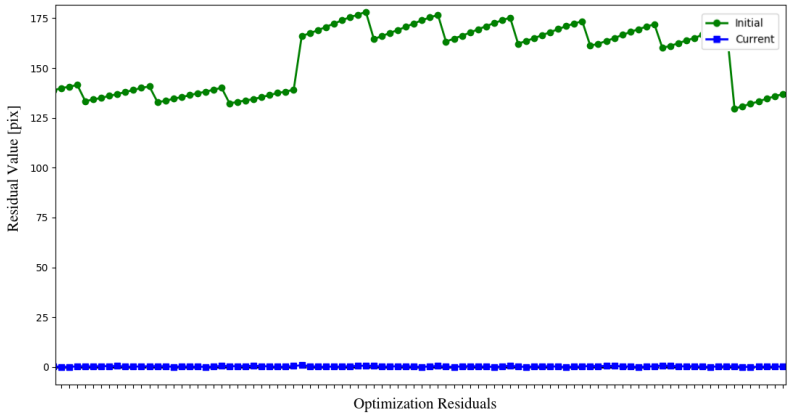


Figure 5.4: Portion of residual values graphic, after the optimization has occurred: the green dots represent the initial value of each residual and the blue dots represent the optimized values of each residual.

As was expected, the final residual values are all almost zero, meaning that the optimization

actually found the minimum of the cost function.

Figure 5.5 has a schema clarifying the way that the optimization procedure works. The termination conditions are what define if the minimum was already found or if the optimizer should keep searching.

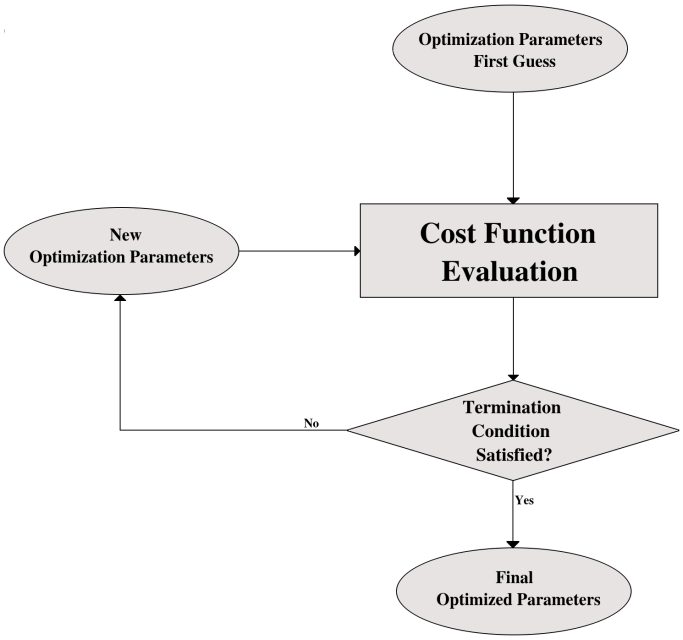


Figure 5.5: Working schema of the optimization procedure cycle.

When the minimum error is found and, therefore, the parameter values are the optimal ones, there is the need of saving the optimized information.

Since the first guess of the optimization parameters was taken from a python dictionary stored in a JSON document, the optimized values are saved at the same way. The new dictionary has the same keys but with different values: the optimized values.

In fact, the geometric transformation parameters (rotation and translation) of the transforms to be calibrated, defined in section 4.1, are accurate now. Besides the sensor pose, the intrinsic parameters of the specific cameras case were also optimized, which makes the *camera_info* key (see section 4.4) values also be updated.

More than storing the new and accurate parameters of the sensor poses, the dictionary was augmented in order to store all the information about the calibration pattern. The new key, named *chessboard*, contains another dictionary with the chessboard pose for each collection, the chess corners number and some more data:

```
- collections {29}
- sensors{4}
- chessboards {7}
    - chess_num_x: 6
    - chess_num_y: 9
    - collections {29}
        - 0 {2}
        - 1 {2}
            - quat [4]
            - trans [3]
        - 2 {2}
        - ...
    - number_corners: 54
    - square_size: 0.101
    - ...
```

Apart from the calibration pattern key, for the cameras case, the indexes of the final projected corner pixels are saved too. Obviously, the pixels values of these points are expected to be similar to the pixels values of the detected chessboard corners, the *ground_ truth* points:

```
- collections {29}
    - 0 {3}
    - 1 {3}
        - data {4}
        - label {4}
            - left_laser {2}
            - right_laser {2}
            - top_right_camera {4}
                - detected:
                - idx [54]
                    - 0 {2}
                    - 1 {2}
                        - x: 252.6875
                        - y: 449.9852
                    - 2 {2}
                    - ...
                - idx_initial [54]
                    - 0 {2}
                    - 1 {2}
                        - x: 93.0471
                        - y: 456.3634
                    - 2 {2}
```

```
                - ...
            - idx_projected [54]
                - 0 {2}
                - 1 {2}
                    - x: 252.5531
                    - y: 450.0138
                - 2 {2}
                - ...
        - top_left_camera {4}
    - transforms {4}
- 2 {3}
- ...
- sensors{4}
- chessboards {7}
```

The new JSON document, generated after the optimization ended successfully, will be referenced as the *augmented JSON* or the *optimization JSON*, for the known reasons. On the other hand, the file with the parameters first guess will be called, from now on, the *original JSON*.

The storage of all this information with the optimized parameters is very useful when it comes to evaluating the results of the optimization procedure, as we are going to see in the next chapter.

Chapter 6

Results

After calibrating all sensors on board the ATLASCAR2, i.e., after performing the two main stages of the presented procedure, it is important to examine the results obtained in order to find out how accurate the general proposed approach is.

In Figure 5.3, it is already possible to see the final position and orientation of all sensors relatively to each one of the remaining ones and the *base_link*. Besides the pose, a large minimization of all residual values is also notable, in Figure 5.4, which induces that the optimization procedure converged into a very good parameter configuration. This very good parameters configuration means that the final achieved values for the sensor poses are very close to the real scene of ATLASCAR2.

However, these two figures only provide a first feedback about the big picture of the calibration procedure. That is why the visualization function of the optimizer (mentioned in section 3.2) was very useful for the implementation of each sub-cost-function. The information that both graphs show, in real time, helped to always understand what could be wrong in the developed optimization code. In other words, this graphical response was an excellent guide for debugging and testing.

In order to deduce the accuracy of the estimated sensor poses by the proposed approach, a very precise tool is required. This tool must be capable of computing the errors related to the final sensor configuration, giving, in that way, some values that could be judged and even compared with errors generated by other standard calibration procedures.

Although all ATLASCAR2 sensors were calibrated simultaneously, the results evaluation is done for each pair of distinct modalities sensors. This kind of results evaluation allow the fair comparison with other pairwise calibration procedures that are so abundant within the works in this field, as was mentioned in chapter 2.

According to all this, and knowing that ATLASCAR2 has two distinct modalities of its sensors (2D LiDARs and RGB cameras), this chapter will explain how camera-to-camera, camera-to-laser and laser-to-laser errors are computed and show the results for each combination of two sensors.

6.1 Camera to Camera Results

The methodology chosen to compute the error related to the final position and orientation between the *top_ right_ camera* and the *top_ left_ camera* is based on the distance between the coordinates, in pixel units, of some known detected points and the coordinates of those same points projected from the other camera.

Specifically, in order to evaluate the optimized pose of both cameras, some deduced relation between the pixel indexes of some common object detected by both cameras is required. In that way, by knowing the real detected indexes of one of the sensors in question, figuring out the expected pixel values on the other camera image, and then comparing them to the real indexes detected by that same camera becomes possible. To achieve this objective, it is essential to have a reference object in both images, whose pixel coordinates could easily be found. The chessboard used for the calibration procedure, which is present in all stored images, and the associated indexes, saved in the JSON files already mentioned, is used for this purpose.

As explained before, besides knowing the difference of the results between the camera poses before and after being enhanced, there is also the interest of comparing the proposed approach to some pairwise calibrations that could have also calibrated the two ATLASCAR2 RGB cameras. Considering this, the present section will be divided in two sub-sections: *ATLASCAR2 Proposed Calibration Approach Results* and *Comparison Between the Proposed Approach and Pairwise Calibration Procedures*.

6.1.1 ATLASCAR2 Proposed Calibration Approach Results

According to the *Camera Calibration and 3D Reconstruction* documentation of OpenCV[1], the projection the 3D world coordinates to the image of a camera, in pixel coordinates, of some point is given by:

$$\begin{bmatrix} \mathrm{u} \\ \mathrm{v} \\ \mathrm{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r11 & r21 & r31 & tx \\ r12 & r22 & r32 & ty \\ r13 & r23 & r33 & tz \end{bmatrix} \cdot \begin{bmatrix} \mathrm{X} \\ \mathrm{Y} \\ \mathrm{Z} \\ 1 \end{bmatrix} \tag{6.1}$$

$$\mathbf{p} = \mathbf{K} \cdot \left[\mathbf{R} \mid \mathbf{t}\right] \cdot \mathbf{P}$$

and

$$\mathrm{x}_{pixel} = \mathrm{u}/\mathrm{w} \ , \tag{6.2}$$

$$\mathrm{y}_{pixel} = \mathrm{v}/\mathrm{w} \ , \tag{6.3}$$

where X, Y and Z are the 3D coordinates of the chessboard corners from some reference frame, $\left[\mathbf{R} \mid \mathbf{t}\right]$ is the geometric transformation matrix, not homogenized, from the sensor link to that reference frame, $\mathbf{K}$ represents the camera intrinsic matrix and, finally, x_{pixel} and y_{pixel} are the chessboard corners pixel coordinates on the camera image.

[1] `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`

Equation 6.1 gives the relation between 3D world coordinates from a common frame and camera image pixel indexes. This means that Equation 6.1 could be applied to each camera, separately. Considering the 3D coordinates measured by the chessboard frame, Z value will, for all points, be zero, since all the chessboard corners are in the XoY plan: $Z^{\text{chess}} = 0$.

In that way, Equation 6.1 simplifies to:

$$\begin{bmatrix} \text{u} \\ \text{v} \\ 1 \end{bmatrix}^{\text{pix}} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r11 & r21 & tx \\ r12 & r22 & ty \\ r13 & r23 & tz \end{bmatrix} \cdot \begin{bmatrix} \text{X} \\ \text{Y} \\ 1 \end{bmatrix}^{\text{chess corners}} , \tag{6.4}$$

where the geometric transformation matrix ${}^{\text{camera}}\mathbf{T'}_{\text{chess}}$ is now just a portion of the ${}^{\text{camera}}\mathbf{T}_{\text{chess}}$ matrix.

As was already mentioned in subsection 5.2.1, by knowing the number of corners that characterize the chessboard used in this project, and its real square size, all X and Y coordinates of all corners relatively to the chess frame are directly calculated.

On the other hand, the JSON file that resulted from the optimization procedure contains information about the optimized transformations between all the links that compose ATLASCAR2 intelligent vehicle and between the reference frame (*base_link*) and the chessboard frame, for each collection (see section 5.5).

With this information, the transformation from each camera to the chessboard for each collection is quickly computed:

$${}^{\text{camera}}\mathbf{T}_{\text{chess}} = {}^{\text{camera}}\mathbf{T}_{\text{base\_link}} \cdot {}^{\text{base\_link}}\mathbf{T}_{\text{chess}} . \tag{6.5}$$

It is important not to forget that only a part of each ${}^{\text{camera}}\mathbf{T}_{\text{chess}}$ is used in Equation 6.4. The same calibration procedure outputted JSON file contains the optimized intrinsic parameters of each camera, and the *ground_truth* pixel indexes for both images for each collection. These *ground_truth* values are, as was already explained, the detected chessboard corners in each camera image by the OpenCV functions mentioned in section 5.2. All this information allows to use this proposed methodology to compute the errors related to the calibration procedure. Equation 6.4 could be re-written as:

$$\begin{bmatrix} \text{X} \\ \text{Y} \\ 1 \end{bmatrix}^{\text{chess}} = \begin{bmatrix} r11 & r21 & tx \\ r12 & r22 & ty \\ r13 & r23 & tz \end{bmatrix}^{-1} \cdot \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \text{u} \\ \text{v} \\ 1 \end{bmatrix}^{\text{pix}} \tag{6.6}$$

$$\mathbf{P}^{\text{chess}} = \left({}^{\text{camera}}\mathbf{T'}_{\text{chess}}\right)^{-1} \cdot \mathbf{K}^{-1} \cdot \mathbf{p}^{\text{pix}} ,$$

Knowing that it can be applied to any of the cameras, and that the only matrix in common for both cameras equations is the 3D chess corners coordinates, it is possible to relate the pixel values from both images by substituting the 3D chess corners matrix:

$$\mathbf{p}^{\text{camera2}} = \mathbf{K}_2 \cdot {}^{\text{camera2}}\mathbf{T'}_{\text{chess}} \cdot \left({}^{\text{camera1}}\mathbf{T'}_{\text{chess}}\right)^{-1} \cdot (\mathbf{K}_1)^{-1} \cdot \mathbf{p}^{\text{camera1}} , \tag{6.7}$$

where $\mathbf{K}_1$ and $\mathbf{K}_2$ represent the intrinsic matrices of *top_left_camera* and *top_right_camera*, respectively; ${}^{\text{camera2}}\mathbf{T'}_{\text{chess}}$ is the specific portion, mentioned above, of the geometric transformation matrix from the *top_right_camera* to the chessboard frame; ${}^{\text{camera1}}\mathbf{T'}_{\text{chess}}$ has

the same meaning but applied to the *top_ left_ camera*, and $\mathbf{p}^{\text{camera1}}$ and $\mathbf{p}^{\text{camera2}}$ are the pixel values of the chessboard corners in the *top_ left_ camera* detected image and in the *top_ right_ camera* detected image, respectively. In other words, *camera1* always refers to the *top_ left_ camera* and *camera2* refers to the *top_ right_ camera*.

Equation 6.7 finally provides the relation between the pixel values of the chessboard corners for both captured images for each collection. By using all the information supplied by the JSON file that the optimizer returned, at the end of the calibration procedure, the expected indexes on the *top_ right_ camera* are directly computed.

The error of the calibration procedure will be the difference between these expected indexes and the *ground_ truth* indexes. In this particular case, the pixels from the *top_ left_ camera* were used and, thus, the error was calculated with the indexes of the *top_ right_ camera* images:

$$\begin{bmatrix} x\_error \\ y\_error \end{bmatrix}^{top\_right\_camera} = \begin{bmatrix} u \\ v \end{bmatrix}^{\text{expected}} - \begin{bmatrix} u \\ v \end{bmatrix}^{ground\_truth} . \tag{6.8}$$

The computation of these errors can be performed with the transformations, and intrinsic values, at the first iteration of the optimizer (before being calibrated) and after optimization procedure has achieved the best sensor configuration. In this way, we could verify if the calibration procedure actually improved sensor poses. It is essential to remember that this calibration was done for all sensors simultaneously, including 2D LiDARs, and, therefore, it did not only focus on the position and orientation of the pair of cameras.

Figure 6.1 shows the errors related to the projection of the chessboard corners from the *top_ left_ camera* to the *top_ right_ camera* before the optimization of the position and orientation parameters of the cameras. Figure 6.2 shows the exact same information but with the camera poses already estimated (after optimization procedure). The errors were saved for all corners of each different collection.

The optimization running time is less than a minute.

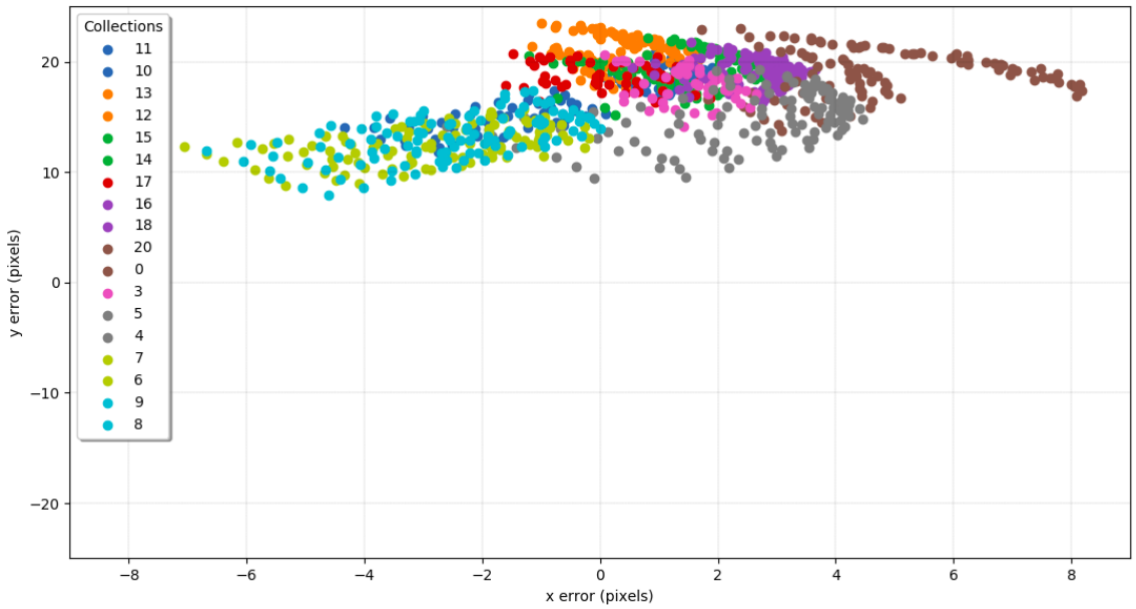


Figure 6.1: Pixel coordinates errors between projected (expected) chessboard corners and the *ground_truth* indexes, from *top_left_camera* to *top_right_camera*, for each collection, before the optimization procedure.

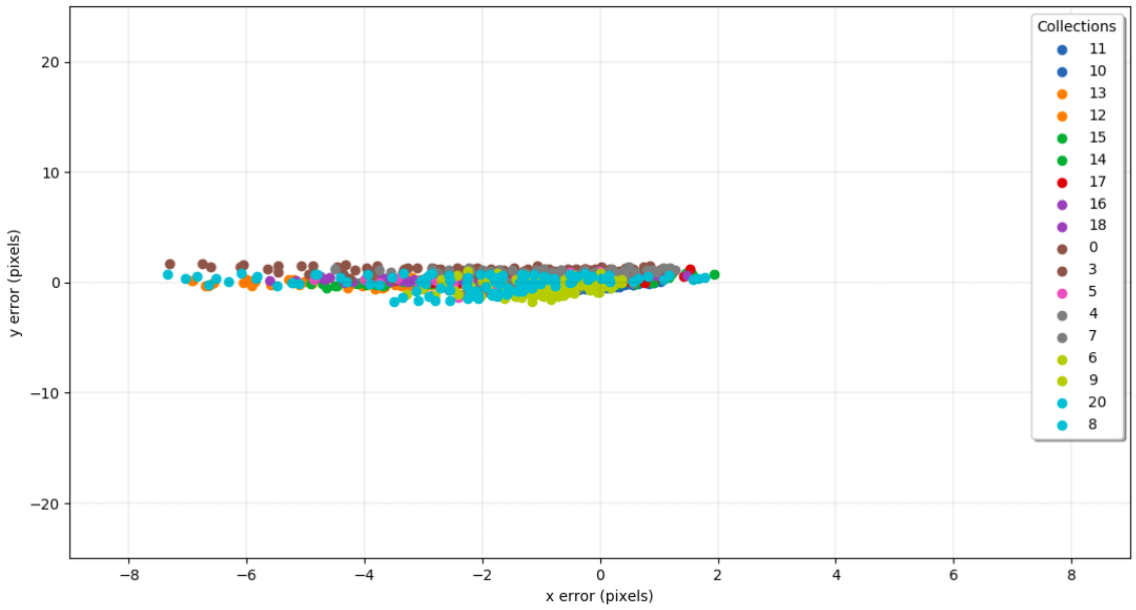


Figure 6.2: Pixel coordinates errors between projected (expected) chessboard corners and the *ground_truth* indexes, from *top_left_camera* to *top_right_camera*, for each collection, after the optimization procedure.

These results can be better evaluated through the calculated mean error and standard deviation values. Table 6.1 provides this numerical information that gives a more precise

feedback about the effect of the calibration procedure.

Values	Average Error (pixels)		Standard Deviation (pixels)	
	Initial	**Final**	**Initial**	**Final**
X vector	2.25	1.64	2.68	1.69
Y vector	17.09	0.53	3.32	0.62
TOTAL	17.34	1.83	8.71	1.51

Table 6.1: Average errors and standard deviations, in pixels, for the distances by x axis, distances by y axis, and Euclidean distances, before and after the optimization.

As we can see, both average error and standard deviation have decreased with the optimization. That means that the position and orientation parameters (and the intrinsic values) of each ATLASCAR2 camera, related to the other camera, are more close to reality than it was with the manually chosen first guess (see section 4.2). This test was done for 21 collections but three of them were not considered since at least one of the cameras could not find the chessboard in its image in those same collections. Thus, times 54 corners, i.e., 972 points, were studied.

6.1.2 Comparison Between the Proposed Approach and Pairwise Calibration Procedures

After ensuring that the proposed calibration procedure actually enhanced the cameras poses, it is now interesting to know if the accuracy of this estimation is better or worse than the pairwise calibrations that could have been applied for the purpose of calibrating only the ATLASCAR2 cameras.

For this comparison purpose, the OpenCV *stereoCalibrate* function [2] and the OpenCV *calibrateCamera* function [3] were chosen to also estimate the transformation between the *top_ left_ camera* and the *top_ right_ camera*.

In order to make this comparison fair, the three distinct calibration procedures should have the exact same source of information. In this case, this same source is the JSON file that outcomes from the first main stage of the proposed approach (see section 4.4). It is important to remember that this JSON file, referenced as *Original JSON*, contains the first guess of the position and orientation parameters of all sensors, as well as the pixel coordinates of the detected chessboard corners, for the cameras case.

To allow for a direct comparison of the results of all different calibrations, the OpenCV procedures were implemented in such a way that the returned estimated parameters, and remaining data, were organized similarly to the proposed approach. This means that each distinct approach will output a final JSON file with the estimated sensors position and orientation parameters. This is the best way to ensure that all this comparison work is homogeneous, fair and reliable.

[2] `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=stereocalib#cv2.stereoCalibrate`

[3] `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#calibratecamera`

Taking all this into account, a specific tool was created for visualizing the results of the different calibrations. *Results Visualization* imports the JSON files outputted by each one of the several approaches that are going to be compared. Since the information within the JSON files is not exactly the same for all procedures, a way of evaluating the results with the same tool that fits for all data provided needs to be found. Further in this sub-section it will be detailed how the *Results Visualization* computes the results.

Figure 6.3 shows a flowchart that describes this work structure, built specifically to compare the presented calibration procedure with some standard pairwise approaches.

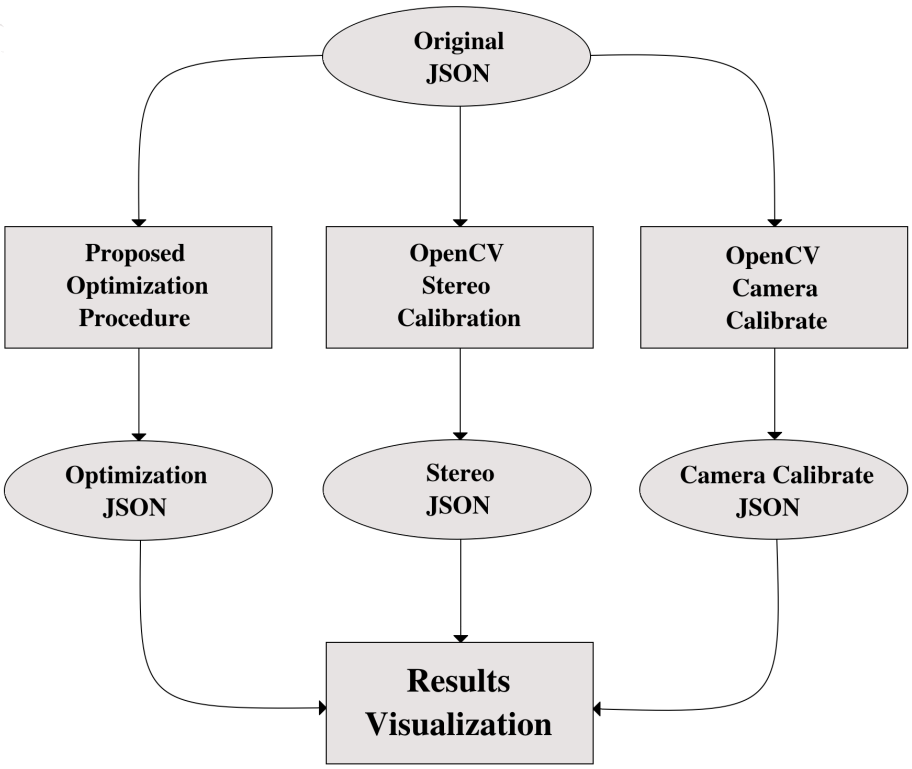


Figure 6.3: Flowchart representing the results comparison structure. Each ellipse represents a JSON file and each rectangle identifies a programmed application.

To get the results of all these calibrations, the strategy used in subsection 6.1.1 seems the best methodology to be considered. However, there is a problem with this strategy for this case. The idea presented in subsection 6.1.1 has the objective of getting to the Equation 6.7. This equation, as is directly understood, requires the information about the geometric transformations between each one of both cameras and the chessboard coordinate frame for each collection. In other words, the transforms ${}^{\text{top\_left\_camera}}\mathbf{T}_{\text{chessboard}}$ and ${}^{\text{top\_right\_camera}}\mathbf{T}_{\text{chessboard}}$, for each collection, are requested for computing the projection (of the pixels related to the chess corners) errors.

The *calibrateCamera* OpenCV function and the proposed approach in this project return the estimation of these transformations which enables the direct usage of Equation 6.7. The problem is the *stereoCalibration* tool of OpenCV: this function only returns the position and orientation parameters of one of the cameras in relation to the other cameras, i.e., it only provides information about the geometric transformation between both the ATLASCAR2 cameras.

Because of this, a new way of computing the errors that could be applied for all different approaches, in order to maintain the philosophy of fair comparison is required.

The planned strategy is to find the relation between one of the cameras coordinate system and the chessboard coordinate frame of each collection, by using the same tool mentioned in section 5.2: *solvePnP* OpenCV function. This function will be used for each one of the distinct calibration procedures. Then, the other camera-chessboard relation will be directly the ones that were estimated and are stored in the *Optimization JSON* and the *Camera Calibrate JSON*, for the cases of the proposed optimization and the *calibrateCamera* function, respectively, and, for *stereoCalibration* case, this other camera to chessboard transformation will be computed as follows:

$$^{\text{camera2}}\mathbf{T}_{\text{chess}} = \overbrace{\left({}^{camera1}\mathbf{T}_{camera2}\right)^{-1}}^{\text{stereo calibration}} \cdot \overbrace{{}^{camera1}\mathbf{T}_{chess}}^{\text{solvePnP}} \tag{6.9}$$

Once again, let's consider the *top_ left_ camera* as the *camera1* and the *top_ right_ camera* as *camera2*.

Thus, *Results Visualization* will use *solvePnP* function, for each different calibration procedure, in order to find the ${}^{camera1}\mathbf{T}_{chess}$. Then, it will get ${}^{camera2}\mathbf{T}_{chess}$ directly from the JSON data for the proposed optimization procedure and for the *calibrateCamera* OpenCV calibration tool. For the case of *setereoCalibration*, the ${}^{camera1}\mathbf{T}_{camera2}$ that is estimated in the related JSON data is used to compute the ${}^{camera2}\mathbf{T}_{chess}$.

Notice that each ${}^{camera1}\mathbf{T}_{chess}$, estimated by *solvePnP*, is not exactly the same for all procedures, since it uses the intrinsic parameters and the distortion parameters which values are already estimated in each distinct JSON file of each calibration approach.

After applying this strategy to have all required information, in a fair way, it is now possible to use Equation 6.7 to project the pixels form the *camera1* (*top_ left_ camera*) to the *camera2* (*top_ right_ camera*), for all studied calibration procedures. Once again, the error computed will measure pixel distance between the projected chess corners (expected pixel coordinates) and the detected corners in the image (*ground_ truth* pixels):

$$\begin{bmatrix} x\_error \\ y\_error \end{bmatrix}^{top\_right\_camera} = \begin{bmatrix} u \\ v \end{bmatrix}^{\text{expected}} - \begin{bmatrix} u \\ v \end{bmatrix}^{ground\_truth} . \tag{6.10}$$

Figure 6.4 show the pixel errors of the three distinct calibration approaches, for each collection.

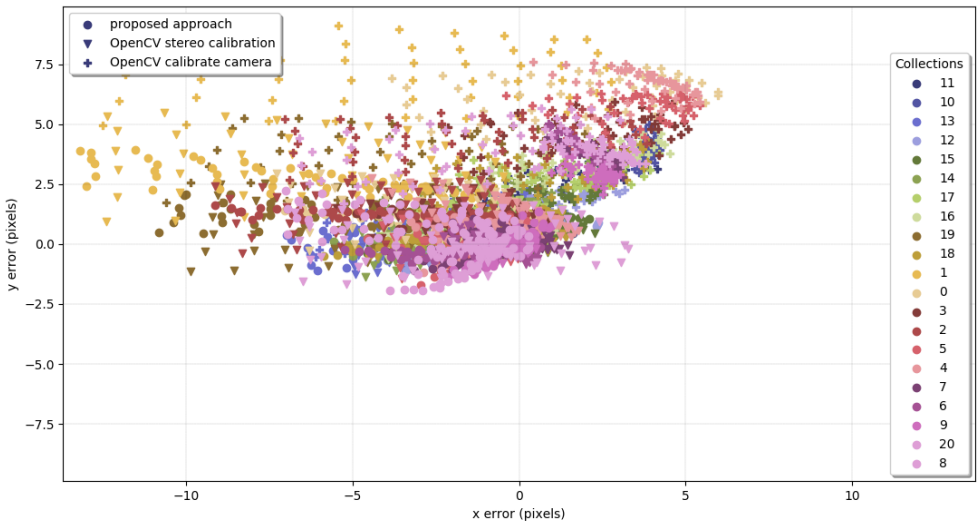


Figure 6.4: Pixel coordinate errors between projected (expected) chessboard corners and the *ground_ truth* indexes, from the *top_ left_ camera* to the *top_ right_ camera*, for each collection, for all different calibration approaches: the circles represent the proposed calibration results; the triangles represent the stereo calibration results and the crosses symbolize the errors of the calibrate camera estimations.

As can be directly conclude from the figure, the three calibration methodology errors are very close to each other, meaning that the proposed approach is, in fact, as good as pairwise calibration procedures. Besides this, the biggest error of each one of the compared methodologies occurs for the same collection (in this case, for collection 1), giving the reliance that the values of these same errors are only due to the specific collection and have nothing to do with the results evaluation implemented strategy or even with the calibration procedure.

These kinds of values of deviation are possibly related to the fact of the chessboard not being static at the captured moment. User attention and patience, in abiding to the procedure explained in section 4.4, could actually ensure a better dataset and, thus, a more accurate calibration; however, the one that is being here evaluated is already very good since it has around 18 collections with small error values for all distinct compared calibration procedures.

Figure 6.4 only show the big picture of the comparison results. Table 6.2 gives the average error and the standard deviation for each direction of the images pixel coordinates for each one of the compared approaches. In this way, the comparison between the proposed procedure and the pairwise calibrations becomes more concrete.

Calibration	Average Error (pixels)		Standard Deviation (pixels)	
	X axis	**Y axis**	**X axis**	**Y axis**
Proposed Approach	2.208	0.733	2.420	0.863
OpenCV stereoCalibrate	1.878	0.842	2.344	1.023
OpenCV calibrateCamera	2.609	3.771	3.053	1.739

Table 6.2: Average errors and standard deviations, in pixels, for the distances in x axis and y axis, for the proposed approach, the OpenCV stereo calibration and the OpenCV camera calibration.

These results reveal that the re-pojection errors of the chessboard corners with the estimated pose of both ATLASCAR2 cameras by the OpenCV *steroCalibrate* pairwise calibration and the proposed approach in this project are very similar. On the other hand, it seems that the estimation of the camera poses by the OpenCV *calibrateCamera* tool is less accurate than the other two approaches, since the error of this specific function is a little greater than the first two procedures mentioned in Table 6.2.

Here is the first big conclusion of the proposed approach to calibrate all sensors on board the ATLASCAR2: the accuracy of calibrating the four sensors simultaneously is the same, for the cameras case, if they were calibrated by a standard pairwise calibration procedure. As mentioned in section 1.1, the majority of complex robot systems, that are usually equipped with more than two sensors, are calibrated trough sequential pairwise approaches, which have some inherent problems. One, and the most important of them, is not considering all the possible arrangements of sensor combinations which also provide data that is certainly important for the overall system configuration. Other problem due to sequential pairwise calibrations is being sensitive to cumulative error.

In fact, these two problems are solved since, without compromising the accuracy of the calibration, all position and orientation sensor parameters were estimated at the same time. The proposed approach in this project actually does not ensure better results of the sensor estimated poses for simple robot platforms with just two sensors, when compared to the traditional pairwise calibration tools. Instead, it works for robots with more sensors, with multiple modalities, and takes in consideration all sensors simultaneously, ensuring that all detected data combinations are contemplated without losing the usual calibration accuracy.

After analysing the camera poses estimation, it is now time to study the results for the 2D LiDARs.

6.2 Other Sensor Combinations: Lasers Case

Besides the RGB cameras, the accuracy of the ATLASCAR2 lasers poses estimation should also be evaluated. The objective is computing the data association errors of the relations laser-to-laser and camera-to-laser, in order to be compared with some standard pairwise calibration approaches, as was developed for the cameras case.

At this point of the project, there is no results evaluation tool for the sensors combination that involve one of the 2D LiDARs yet. Contrary to what had already been implemented for camera-to-camera data combination, the lasers measurements are still not being judged and compared to pairwise known calibrations procedures.

The reason for this missing function, that will complete the presented work, is related with the final configuration of the ATLASCAR2 2D LiDARs.

At the end of the optimization procedure, it is possible to realize that the position and orientation of the *left_laser* and the *right_laser* are very far for the real world scene of ATLASCAR2. In fact, when all sensors pose parameters are considered, the cameras end up with a very similar configuration to the actual real relation between each other poses, but that is not happening for LiDARs. It is possible to notice this bad result through the graphical feedback provided by the *Optimizer.setVisualizationFunction* (mentioned in section 3.2) or even through the information about the geometric transformations between the ATLASCAR2 links contained in the JSON file outputted by the optimizer. The graphical support is the same as the one shown in Figure 5.3, which has the *right_laser* and the *left_laser* misplaced.

Despite the bad final laser configuration, the cost sub-function created for this modality was well programmed and implemented and the optimizer is actually lowering the returned residuals. Figure 6.5 shows the four (two distances for each one of the two cluster limit points) returned residuals of the *left_laser* cost sub-function at the beginning and at the end of the optimization procedure.

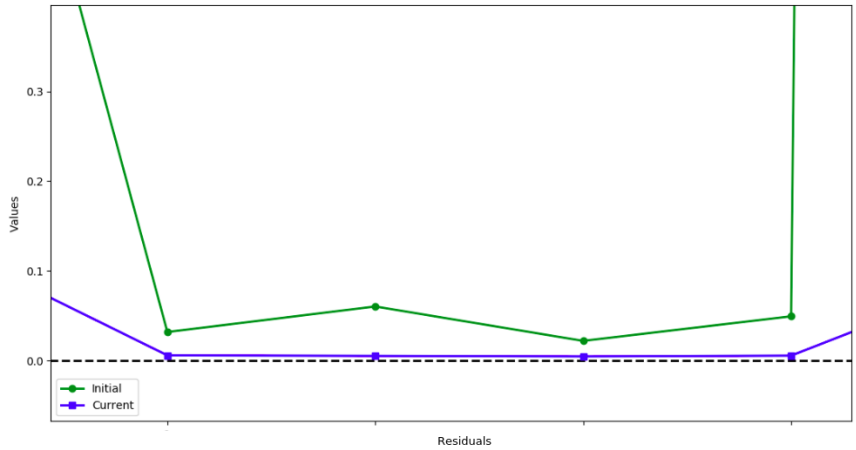


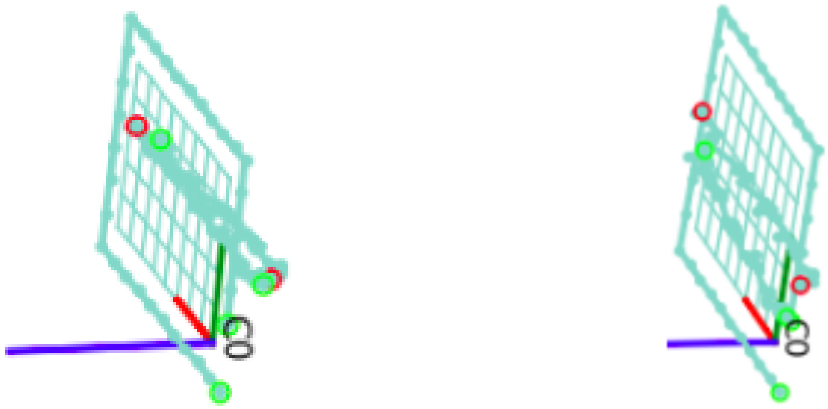
Figure 6.5: *Left_laser* residuals at the beginning (green line) and at the end (blue line) of the optimization procedure, for a specific collection.

As is directly visualized from this portion of the residuals graphic, the distances between the two points that limit the selected cluster, between all the laser detected ranges, and the border of the chessboard were minimized and are very close to zero at the end of the optimization.

In addition to the residuals graphic information, it is also provided a real time visual feedback of the detected clusters and of the chessboard position and orientation. This means that,

since the beginning of the optimization until it finishes, *Optimizer.setVisualizationFunction* shows the chessboard and the lasers measurements poses in the same 3D world coordinate, for each collection. Thanks to this, it is possible to ensure that the labeled cluster from all the point cloud, for each 2D LiDAR, is getting closer to the chessboard.

Figure 6.6 and Figure 6.7 show the same scene from different view angles, before the optimization starts, and after the sensors and the chessboard poses are estimated.



(a) (b)

Figure 6.6: First view of the chessboard for collection 0 and the labeled clusters detected by the *left_laser* (green circles around the two limit points) and by the *right_laser* (red circles around the two limit points): a) at the beginning of the optimization; b) at the end of the optimization procedure.

(a) (b)

Figure 6.7: Second view of the chessboard for collection 0 and the labeled clusters detected by the *left_laser* (green circles around the two limit points) and by the *right_laser* (red circles around the two limit points): a) at the beginning of the optimization; b) at the end of the optimization procedure.

Both images demonstrate, precisely, that the clusters are actually getting closer to the chessboard plan, which makes sense since the distances returned by the laser cost sub-function are being minimized, as was also concluded from the residuals graphics.

Knowing that the optimization procedure is working just fine and also that all developed cost-function is being respected, the wrong final LiDARs configuration turns out to be a delicate concern that should be analysed carefully, in order to get to the best possible approach. To solve this issue, the first step is to understand the reasons responsible for this misplaced laser results.

By looking into the final position of the labeled clusters from the *left_ laser* and the *right_ laser* within the chessboard pose (see Figure 6.8), it is deduced that there are infinite cluster disposals that also satisfy the minimum of the laser related cost sub-function.

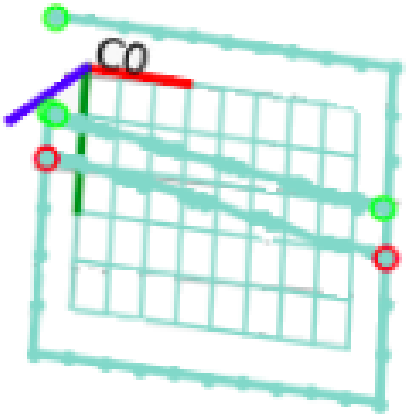


Figure 6.8: Chessboard for collection 0 and the labeled clusters detected by the *left_ laser* (green circles) and by the *right_ laser* (red circles) at the end of the optimization procedure.

In order to lower the returned residuals by the cost function, it is only required that the limit cluster points (red and green circles in Figure 6.8) get as close as possible to the chessboard plan boundary. The size of the cluster, i.e., the number of points that compose it, is fixed for each collection; however, even with a fixed cluster size, there are infinite possibilities for having the two specific points of the detected point cloud coincident to some chessboard plan delimited points. In other words, the information that is used for computing the cost function is not sufficient to ensure a correct final laser pose, even if the returned value of this cost sub-function is zero.

The work that is being developed at this moment focuses, precisely, on this issue. The next step is saving a dataset with specific chessboard positions and orientations, because a possible way to solve this problem is to ensure that the combination of all the collections will only allow one final laser configuration that could achieve the lowest value of the cost sub-function.

Besides this, a new chessboard, that could be easily detected by cameras and lasers at the same time, is also being projected. This special chessboard must have the white squares in a different plan than the black squares. These distinct planes will be directly detected by the

2D LiDARs and not compromise the chessboard cameras detection. Of course this idea is still just a concept that will be tested after the creation of this special calibration pattern.

Summing up, the optimizer and the cost function are working as was expected and were well implemented but the final position and orientation of the 2D LiDARs is not the correct one since there are several pose possibilities (optimization parameters values) that could converge into the cost function minimum. Because of this, there is still no available tool to evaluate the final pose of the lasers. At the moment some methods are being studied to solve this problem.

Chapter 7

Conclusions and Future Work

The main purpose of the presented work is to perform the extrinsic calibration of multi-modal sensor platforms, as in the case of ATLASCAR2. This means that the objective is to estimate the position and orientation parameters that describe the configuration of the two RGB cameras and the two 2D LiDARs of ATLASCAR2.

Considering this ambition, the developed project was successful since it enhanced the poses of *top_ left_ camera* and *top_ right_ camera* with enough accuracy, and is almost achieving the same good results for *left_ laser* and *right_ laser* poses.

Apart from the accomplished main goal, the proposed approach for calibrating the sensors on board ATLASCAR2 is characterized for bringing new advantages for the calibration process, which turn this project into a very useful tool for other robot developers that work with robotic systems.

In fact, for complex robotic platforms, like the case of Intelligent Vehicles, the common approach to get all sensor poses estimated is to perform sequential pairwise calibrations, that will only take in consideration two sensors, and the related data combination, at a time. This kind of procedure has a lot of inherent problems that have already been explained at the beginning of this document. The proposed approach has the advantage of calibrating all sensors simultaneously. This means that all sensor data combinations are considered when it comes to optimizing the pose of each sensor, not letting any sensor-to-sensor arrangement out of attention. This methodology is more reliable than sequential pairwise calibrations, since it does not compromise the final configuration of the complete sensors system.

The fact of considering all sensor poses simultaneously could result in a lower accuracy of the position and orientation parameter estimation. However, as shown in section 6.1, the camera poses had the same accuracy as if they were calibrated only between each other through the OpenCV *stereoCalibrate* tool. This is the second most important achieved conclusion about the proposed approach: the confidence of the sensors complete group final position, maintaining the accuracy level of the calibration procedure, makes the developed methodology the first choice for estimating the sensor poses of an Intelligent Vehicle.

The first most important conclusion is that the approach designed for calibrating the sensors on board ATLASCAR2 is actually general to any other complex robot system. This flexibility comes from the fact that the number of sensors and their modalities are not restricted. The general implemented philosophy characterizes this work as very profitable for many robot

research communities, by allowing its usage for applications other than the ATLASCAR2. The possible usage by others robot developers of the proposed calibration is also ensured by the compliance with the ROS framework.

There is another notable detail of the calibration performance: the ATLASCAR2 transformation graph was not altered during the process of calibrating its sensors. The particular attention for not altering the *tf* tree is essential not to compromise some robot functionalities such as, for example, robot visualization or collision detection. To accomplish this, the problem was formulated as an optimization procedure of a set of partial transformations which account for specific links in the transformation chains of the sensors. By parsing the robot description, it was possible to recognize the transformations due to be optimized within the robot structure.

The last advantage of the presented calibration procedure is using interactive tools for positioning the sensors and labelling the sensor detected data, which, for being intuitive and user-friendly, eases the calibration proceedings.

All these advantages led to the development of a paper that was published and presented in ROBOT2019: Fourth Iberian Robotics Conference. The publication of [19] derived from the work described in this document.

After enumerating all these advantages of the proposed approach, identifying the next steps that should be taken for the improvement of this project becomes essential.

The results achieved by calibrating ATLASCAR2 sensors prove to be very positive for the cameras, but the procedure is not finished for lasers yet. Despite the fact of the lowered residuals and, thus, the respected laser related cost sub-function, the final laser poses is not close to what is expected. For this reason, creating a new approach to ensure that there is only one parameters configuration that could return the lowest possible residual values is required. This could be accomplished by saving a dataset with established chessboard different positions and orientations or even by using a new and specific calibration pattern that could easily be recognized by all sensors. One way or the other, this is the most urgent step of the future work.

An obvious idea to add value to this work is extending the procedure to additional sensor modalities, e.g., 3D LiDARs, RGB-D cameras, *Radio Detection And Ranging* (RaDAR), etc. After solving the 2D LiDARs issue, future work will focus on the extension to other sensor modalities, which will also contribute for the generality of the approach.

Finally, the most wanted goal is turning all this project into a usable ROS package, in order to provide a tool that could be used for calibrating robot systems with several multi-modal sensors. The ultimate ambition is precisely contributing to the calibration world by releasing this package to the robotic research communities.

Bibliography

[1] Agarwal, S., Snavely, N., Seitz, S.M., Szeliski, R.: Bundle adjustment in the large. In: Proceedings of the 11th European Conference on Computer Vision: Part II. pp. 29–42. ECCV'10, Springer-Verlag, Berlin, Heidelberg (2010)

[2] Almeida, M., Dias, P., Oliveira, M., Santos, V.: 3d-2d laser range finder calibration using a conic based geometry shape. In: Image Analysis and Recognition. pp. 312–319 (2012)

[3] Basso, F., Menegatti, E., Pretto, A.: Robust intrinsic and extrinsic calibration of RGB-D cameras. IEEE Trans. on Robotics 34(5), 1315–1332 (2018)

[4] Chen, G., Cui, G., Jin, Z., Wu, F., Chen, X.: Accurate intrinsic and extrinsic calibration of RGB-D cameras with gp-based depth correction. IEEE Sensors Journal 19(7), 2685–2694 (2019)

[5] Chen, Z., Yang, X., Zhang, C., Jiang, S.: Extrinsic calibration of a laser range finder and a camera based on the automatic detection of line feature. In: 2016 9th Int. Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI). pp. 448–453 (2016)

[6] Dinh, V.Q., Nguyen, T.P., Jeon, J.W.: Rectification using different types of cameras attached to a vehicle. IEEE Trans. on Image Processing 28(2), 815–826 (2019)

[7] Elatta, A., Gen, L., Zhi, F., Daoyuan, Y., Fei, L.: An overview of robot calibration. Information Technology Journal 3, 74–78 (2004)

[8] Foote, T.: tf: The transform library. In: 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA). pp. 1–6 (2013)

[9] Gao, D., Duan, J., Yang, X., Zheng, B.: A method of spatial calibration for camera and radar. In: 2010 8th World Congress on Intelligent Control and Automation. pp. 6211–6215 (2010)

[10] Gomez, R., Briales, J., Fernández-Moral, E., González-Jiménez, J.: Extrinsic calibration of a 2d laser-rangefinder and a camera based on scene corners. In: Proceedings - IEEE International Conference on Robotics and Automation. vol. 2015, pp. 3611–3616 (2015)

[11] Guindel, C., Beltrán, J., Martín, D., García, F.: Automatic extrinsic calibration for LiDAR-stereo vehicle sensor setups. 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC) pp. 1–6 (2017)

[12] Hornegger, J., Tomasi, C.: Representation issues in the ml estimation of camera motion. In: Proceedings of the IEEE International Conference on Computer Vision. vol. 1, pp. 640 – 647 vol.1 (1999)

[13] Häselich, M., Bing, R., Paulus, D.: Calibration of multiple cameras to a 3d laser range finder. In: 2012 IEEE Int. Conf. on Emerging Signal Processing Applications. pp. 25–28 (2012)

[14] Khan, A., Aragon-Camarasa, G., Sun, L., Siebert, J.P.: On the calibration of active binocular and RGBd vision systems for dual-arm robots. In: 2016 IEEE Int. Conf. on Robotics and Biomimetics (ROBIO). pp. 1960–1965 (2016)

[15] Liao, Y., Li, G., Ju, Z., Liu, H., Jiang, D.: Joint kinect and multiple external cameras simultaneous calibration. In: 2017 2nd Int. Conf. on Advanced Robotics and Mechatronics (ICARM). pp. 305–310 (2017)

[16] Ling, Y., Shen, S.: High-precision online markerless stereo extrinsic calibration. In: 2016 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). pp. 1771–1778 (2016)

[17] Madeira, T.: Enhancement of RGB-D image alignment using fiducial markers. Master Thesis, University of Aveiro (2019)

[18] Mueller, G.R., Wuensche, H.: Continuous stereo camera calibration in urban scenarios. In: 2017 IEEE 20th Int. Conf. on Intelligent Transportation Systems (ITSC). pp. 1–6 (2017)

[19] Oliveira, M., Castro, A., Madeira, T., Dias, P., Santos, V.: A general approach to the extrinsic calibration of intelligent vehicles using ros. In: Robot 2019: Fourth Iberian Robotics Conference (2019)

[20] Pereira, M., Silva, D., Santos, V., Dias, P.: Self calibration of multiple LiDARs and cameras on autonomous vehicles. Robotics and Autonomous Systems 83, 326 – 337 (2016)

[21] Pham, D., Yang, Y.: Optimization of multi-modal discrete functions using genetic algorithms. Proceedings of the Institution of Mechanical Engineers 207, 53–59 (1993)

[22] Pradeep, V., Konolige, K., Berger, E.: Calibrating a Multi-arm Multi-sensor Robot: A Bundle Adjustment Approach, pp. 211–225. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[23] Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)

[24] Rehder, J., Siegwart, R., Furgale, P.: A general approach to spatiotemporal calibration in multisensor systems. IEEE Trans. on Robotics 32(2), 383–398 (2016)

[25] Santos, V., Almeida, J., Ávila, E., Gameiro, D., Oliveira, M., Pascoal, R., Sabino, R., Stein, P.: Atlascar - technologies for a computer assisted driving system, on board a common automobile. In: 13th Int. IEEE Conf. on Intelligent Transpor tation Systems. pp. 1421–1427 (2010)

[26] Vasconcelos, F., Barreto, J.P., Nunes, U.: A minimal solution for the extrinsic calibration of a camera and a laser-rangefinder. IEEE Trans. on Pattern Analysis and Machine Intelligence 34(11), 2097–2107 (2012)

[27] Wu, L., Zhu, B.: Binocular stereovision camera calibration. In: 2015 IEEE Int. Conf. on Mechatronics and Automation (ICMA). pp. 2638–2642 (2015)

[28] Zhang, C., Zhang, Z.: Calibration between depth and color sensors for commodity depth cameras. In: 2011 IEEE Int. Conf. on Multimedia and Expo. pp. 1–6 (2011)

[29] Zhang, Q., Pless, R.: Extrinsic calibration of a camera and laser range finder (improves camera calibration). In: 2004 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). vol. 3, pp. 2301–2306 vol.3 (2004)