Universidade de Aveiro Departamento de Engenharia Mecânica
2016

David Tiago Vieira da Silva

Calibração Multissensorial e Fusão de Dados Utilizando LIDAR e Visão

Multisensor Calibration and Data Fusion Using LIDAR and Vision

Universidade de Aveiro Departamento de Engenharia Mecânica
2016

**David Tiago
Vieira da Silva**

Calibração Multissensorial e Fusão de Dados Utilizando LIDAR e Visão

Multisensor Calibration and Data Fusion Using LIDAR and Vision

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestrado em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro e de Paulo Miguel de Jesus Dias, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president	**Prof. Doutor Jorge Augusto Fernandes Ferreira** Professor Auxiliar da Universidade de Aveiro
vogais / committee	**Doutor Cristiano Premebida** Investigador da Universidade de Coimbra - Faculdade de Ciência e Tecnologia (arguente)
	Prof. Doutor Vítor Manuel Ferreira dos Santos Professor Associado da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Deixo os meus sinceros agradecimentos aos Professores Vítor Santos e Paulo Dias pela motivação e orientação que me deram ao longo deste trabalho, assim como pela sua dedicação e entusiamo contagiante pelo projeto. Agradeço também ao Marcelo Pereira pela sua disponibilidade e ajuda na transição deste projeto.

Quero agradecer à minha família pelo apoio incondicional nesta jornada. Nada disto seria possível sem eles.

Agradeço também aos meus amigos, em especial à pequena família que é o LAR, pela entreajuda e pelos excelentes momentos que passamos juntos.

palavras-chave calibração extrínseca, fusão sensorial, nuvem de pontos, visão computacional, interface do utilizador.

resumo

Este trabalho expande um pacote de calibração extrínseca a sensores baseados em visão. Uma bola é usada como alvo de calibração para estimar a posição e orientação de vários sensores relativamente a um referencial comum. Primeiramente, é desenvolvido um algoritmo para deteção da bola na imagem, em seguida, são implementados e testados dois métodos para estimar a posição e orientação da câmara relativamente ao referencial de referência. Adicionalmente, uma interface gráfica é desenvolvida para o pacote de calibração. A interface permite a calibração de várias configurações de sensores, simplifica o processo de calibração e facilita a expansão do pacote a novos sensores. De forma a ilustrar o processo de expansão, o sensor Microsoft Kinect 3D-depth é integrado no pacote de calibração e respetiva interface. Por último, utilizando a posição e orientação estimadas, os dados dos sensores são fundidos em um único referencial. Fusão sensorial é realizada no ATLASCAR 1, um veículo autónomo, e em testes laboratoriais.

keywords extrinsic calibration, sensor data fusion, point cloud, computer vision, user interface.

abstract The work presented in this thesis expands an existing extrinsic calibration package to vision-based sensors. A ball is used as a calibration target to estimate the pose of multiple sensors relative to a reference frame. First, an algorithm is developed to detect the ball in an image then, two methods are implemented and tested to estimate the camera pose relative to a reference frame. Additionally, a graphical user interface is developed for the extrinsic calibration package. The interface allows multiple sensor configurations to be calibrated, greatly simplifies the calibration procedure and is easily expandable to new sensors. To illustrate the expansion process, Microsoft Kinect 3D-depth sensor is integrated in the calibration package and interface. Finally, from the estimated poses, sensor data fusion is achieved with ATLASCAR 1, an autonomous vehicle, and in indoors tests.

Contents

List of Figures

List of Tables

Acronyms

AD Autonomous Driving. 1, 18

ADAS Advanced Driver Assistance Systems. 1, 18

API Application Programming Interface. 23

GUI Graphical User Interface. 4, 21, 22, 24, 26, 37, 41, 45–48, 51, 53, 54, 60, 66, 71, 72, 77, 78

HSV Hue, Saturation, and Value. 32, 71, 78

IAV Internal Angle Variance. 9

ICP Iterative Closest Point. 7, 11

IR Infrared. 16, 17

KPIECE Kinematic Planning by Interior-Exterior Cell Exploration. 72

LAR Laboratory for Automation and Robotics. 1, 3, 4, 7

LARtk Laboratory for Automation and Robotics Toolkit. 24

LIDAR Light Detection And Ranging. 1, 4, 16, 24, 25, 35, 51

MLE Maximum Likelihood Estimation. 6

MOC Meta-Object Compiler. 21

OpenCV Open Source Computer Vision Library. 7, 20, 21, 24, 26, 27, 31

PCD Point Cloud Data. 20

PCL Point Cloud Library. 11, 12, 19, 20, 24, 47

PnP Perspective-n-Point. 5

ROS Robot Operating System. 22–24, 26, 41, 77

RPC Remote Procedure Call. 22

RRT Rapidly-exploring Random Tree. 72

SAC SAmple Consensus. 20

SDK	Software Development Kit. 23, 77
SNN	Spatial Nearest Neighbor. 9
SPARS	SPArse Roadmap Spanner. 72
SVD	Singular Value Decomposition. 12
TOF	Time-Of-Flight. 16
UI	User Interface. 21
VTK	Visualization ToolKit. 8

Chapter 1

Introduction

Continuous advancements in robotics require robots to be increasingly aware of their surroundings and behave accordingly. In the fields of Autonomous Driving (AD) and Advanced Driver Assistance Systems (ADAS), perception of the vehicle's surroundings is essential, since without it the vehicle cannot assess other cars' motion, pedestrian motion or even its own motion.

The lack of a universal sensor solution for robot perception leads AD and ADAS to frequently rely on multiple Light Detection And Ranging (LIDAR) and vision-based sensors. Two very distinct approaches are utilized to extract information from multiple sensors of different types: the first approach is to process data and draw conclusions separately for each sensor. The second approach is to merge all sensor data in a single reference frame, processing and drawing conclusions from it.

Merging the data from sensors in a single reference frame requires knowledge of the relative position and orientation (geometric transformation) between each sensor and the reference frame. The process of determining those geometric transformations is called an extrinsic calibration. Many methods have been proposed to solve the extrinsic calibration problem with sensors of different types, most of them are manual or semi-automatic. The problem with most methods is that they require manual measurements, manual correspondences between data or are restricted to two types of data sensors.

This dissertation is an extension of the calibration method presented by Pereira [1], an automatic calibration method which uses a ball as a calibration target; the ultimate goal is to achieve sensor data fusion.

1.1 The ATLAS Project

This work is part of the ATLAS Project, created in 2003 and developed by the Group of Automation and Robotics at the Laboratory for Automation and Robotics (LAR), located in the Department of Mechanical Engineering of the University of Aveiro, Portugal. Its mission is to create and develop advanced sensing and active systems designed for implementation in AD and ADAS.

The first robots were designed to compete in autonomous driving competitions at the Portuguese National Robotics Festival. Along the years several prototypes, some of them shown in Figure 1.1, were created and developed with great success.

With the experience acquired in autonomous driving in controlled environments the pro-

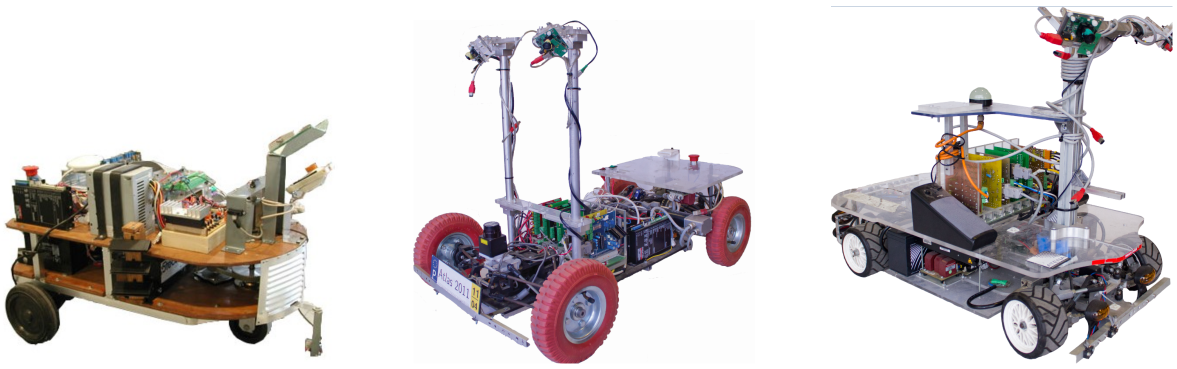

(a) Atlas1 (b) Atlas2000 (c) AtlasMV

Figure 1.1: ATLAS autonomous robots.

ject took on a new and more complex challenge – real world autonomous driving with a real scale prototype, ATLASCAR 1, shown in Figure 1.2.

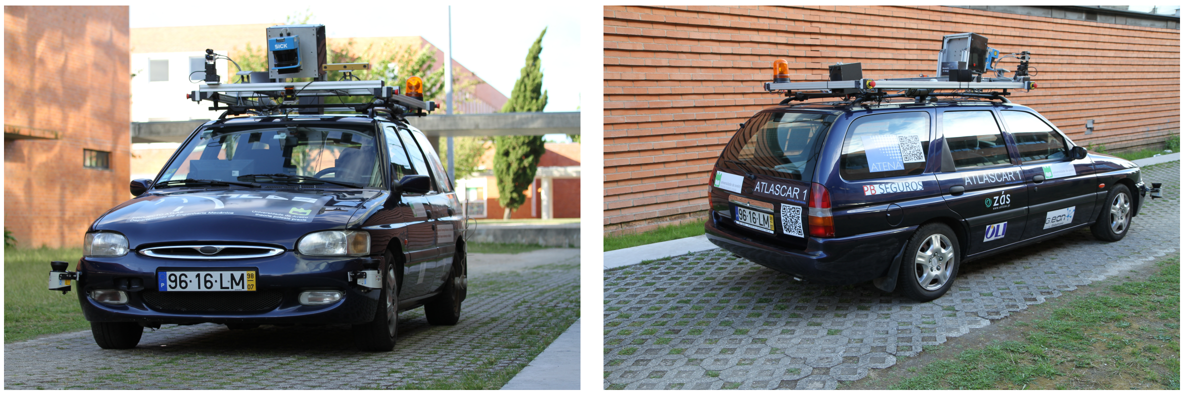


Figure 1.2: ATLASCAR 1 autonomous vehicle.

ATLASCAR 1 is a Ford Escort Station Wagon from 1998 equipped with several sensors, actuators and computational units (computers and micro controllers). Sensors are used to perceive the vehicle state as well as its surroundings. Data from the sensors is processed and analyzed by the computer which generates commands according to the data. Actuators receive commands from the computer and control the vehicle. All external equipment are powered by an auxiliary alternator, its DC output is then directed to a buffer battery, and converted by an inverter to a 220V AC power line that is stabilized by an UPS. Two voltage regulators are also set at 24V DC and 12V DC to power the sensors and other devices.

However, after years of modifications and experiments, ATLASCAR 1 is approaching its end-of-life and is going to be subsumed by ATLASCAR 2 (see Figure 1.3) a Mitsubishi i-MiEV (2015). Being an electric vehicle, ATLASCAR 2 can potentially be much easier to control, greatly reducing the number of custom-made actuators used on ATLASCAR 1. It is also planned to use its battery pack as the power supply for all sensors, computational units and other external equipments.

Figure 1.3: ATLASCAR 2.

1.2 Project Context

In autonomous driving, knowing the vehicle's surroundings is key for safe driving. The inexistence of a universal solution leads vehicles, like ATLASCAR 1, to use multiple sensors of different types. However, each sensor has its own perspective of the world around it, making decisions based on each perspective separately doesn't always yield good results.

Merging all sensor data in one frame allows for the best possible perception of the vehicle's surroundings, hence also allowing for better decision-making. To merge data in one frame, knowledge of the geometric transformations between each sensor frame and the reference frame is required, this is an extrinsic calibration problem. Solving this problem accurately is also very important, especially for orientations, since even a one degree error translates to meters at large distances.

Pereira [1] presented an automatic calibration method to solve this problem, with good results for the SICK LMS151, SICK LD-MRS400001 and a SwissRanger SR4000, but lacking in its implementation of vision-based sensors. So, in this work two more sensors will be implemented – Point Grey FL3-GE-28S4-C cameras and Microsoft Kinect. Both sensors are frequently used in experiments at LAR: the Point Grey cameras are used on ATLASCAR 1 and Kinect is widely used in indoors experiments with robots.

The complete lack of extrinsic calibration applications for multiple sensors and multiple configurations also calls for the generalization of the current calibration methodology to work with any number of the supported sensors. With the future migration to ATLASCAR 2 the ability to easily calibrate multiple sensors in different configurations is going to be particularly useful in order to obtain the best possible coverage of the vehicle's surroundings.

1.3 Objectives

As described in Section 1.2, there is specific need at LAR to expand the calibration methodology to more sensors and, due to the complete lack of extrinsic calibration applications for multiple sensors and multiple sensor configurations, the main objectives of this work are:

- integrate vision based sensors on the calibration framework;
- develop an easy-to-use GUI for the calibration package, easily expandable to more sensors and usable with any number of the supported sensors;
- perform sensor data fusion in a single frame.

1.4 Document Structure

This thesis is composed of six chapters, including the present chapter. Chapter 2 provides a description of the main hardware and software used in this work. Chapter 3 presents the proposed solution for camera-LIDAR extrinsic calibration to be integrated in the calibration package. The methods and algorithms used to achieve the proposed solution are also detailed in Chapter 3. Chapter 4 addresses the Graphical User Interface (GUI) development for the calibration package. Chapter 5 describes the tests made to evaluate the proposed solution for camera-LIDAR extrinsic calibration, along with the overall performance of the calibration package, additionally sensor data fusion is carried out. Chapter 6 presents the conclusions and future work. Finally, Apendix A contains instructions to install and use the calibration package.

1.5 Related Work

Extrinsic calibration is a basic requirement in multisensor platforms where data needs to be represented in a common reference frame for the purpose of data fusion and subsequent analysis. Several solutions have been proposed along the years. This section will first present related work external to LAR, followed by related work developed at LAR.

1.5.1 Related Work External to LAR

In 1995, Wasielewski and Strauss proposed a calibration method for a perception system consisting of a camera and a 2D laser range finder in [2]. The calibration method is based on an edge-like calibration pattern with two zones – a white-like zone and a black-like zone – shown in Figure 1.4. The method matches the image of the line separating the zones to the point; which results from intersection between the line and the laser plane. The calibration pattern is placed at a distance varying from 1 to 5 meters, however the maximum distance is a function of the laser angular resolution; placing the pattern too far leads to not enough points to detect the edge. This calibration method requires the sensors to be in a planar configuration and, in a multi-camera configuration, the calibration pattern has to be positioned precisely so that all cameras have vision over both zones.

In [3], Zhang and Pless describe theoretical and experimental results for extrinsic calibration of a sensor platform consisting of a camera and a 2D laser range finder, as shown in Figure 1.5. The calibration method is based on observing a planar checkerboard pattern visible to both sensors. The authors assume that the calibration plane (the plane surface defined by the checkerboard) is the Z=0 plane in the world coordinates system. In the camera coordinate system, the calibration plane can be parametrized by a vector. The laser points must lie on the calibration plane estimated from the camera, which gives two geometric constraints on the rigid transformation between the camera and the laser.

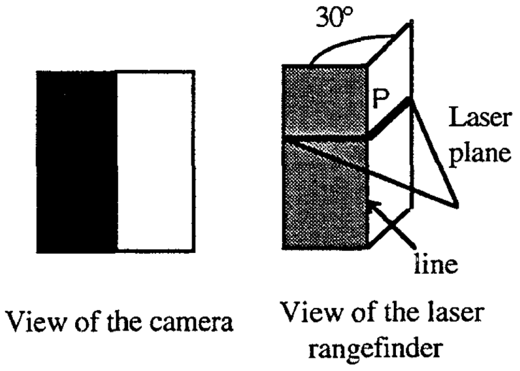


Figure 1.4: Calibration pattern used in [2].

The authors then solve the extrinsic calibration problem with a linear solution and subsequently, with a non-linear minimization of a cost function based on the reprojection error and laser to calibration plane Euclidean distance. The non-linear minimization problem is solved with the Levenberg-Marquardt method [4], [5]. However, the calibration target cannot be oriented too obliquely to the cameras; it also has to be close enough to the cameras so that the checkerboard grid corners can be accurately detected; and it has to be positioned and oriented differently in a sufficient number of views.

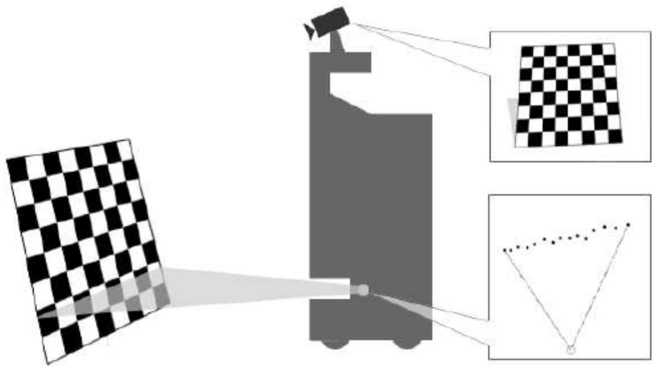


Figure 1.5: General setup used in [3].

There are also proposed solutions for extrinsic calibration between a camera and a 3D laser range finder. Scaramuzza et al., [6] presents an extrinsic calibration method for a camera and a 3D laser range finder. As opposed to the previously discussed solutions, this method does not use any calibration pattern or object, instead the authors use point correspondences that the user has to hand select from a single laser-camera acquisition. Another difference is that this calibration technique only needs a single acquisition of both the laser scanner and the camera, furthermore, the acquired data can be of any natural scene. To compute the extrinsic calibration the authors solve a Perspective-n-Point (PnP) problem by using the algorithm described in [7]. The output of the PnP algorithm are depth factors of the camera points in the camera reference frame. To obtain the rigid transformation between the two point sets, the authors used the motion estimation algorithm proposed in [8]. Since the PnP algorithm is quite sensitive to the position of the manually selected points and also to noisy information, the authors refine the solution with a non-linear minimization of a cost function based on the reprojection error.

Test results indicate that the user should select 5 or more laser-camera correspondences uniformly chosen from the entire scene. Although the results have low reprojection errors in their tests, hand-picked correspondences mean that a different user would obtain different results. Moreover, picking uniformly correspondences from the entire scene requires large overlaps in the field of view of both sensors, which is not always possible.

Ruan and Huber [9] introduce a method for extrinsic calibration of multiple 3D-depth sensors observing a common workspace. The authors used a hand-held spherical target which is positioned around the common workspace. The sphere is automatically segmented from the 3D-depth sensors point clouds. The ball segmentation is done by first constructing a per-pixel background model from a number of frames with an empty scene. During the calibration process the background model is subtracted to the point cloud, leaving only points that are not part of the empty scene. Further processing is done with a 2.5D erosion and dilation developed by the authors, isolating the ball from the person and stick holding the sphere. The remaining point clusters are passed to a cost function; a lower cost means the cluster has a higher potential to be a sphere. The sphere center is then estimated by using the method described in [10]. After calculating the sphere center for each camera, the geometric transformation between them is computed by minimizing the 3D Euclidean error and then refined by a Maximum Likelihood Estimation (MLE) algorithm developed by the authors.

A comparison between the proposed method and Zhang's [3] checkerboard method is made using two Asus Xtion Pro. The method proposed by [9] performed better by approximately 50%, with errors below 2 cm in all test metrics. The authors then tested the method with two different 3D-depth sensors – a SwissRanger SR4000 and a Asus Xtion Pro – and a Tyzx stereo camera. Obtaining again errors below 2 cm in all metrics, the fused point cloud is shown in Figure 1.6.

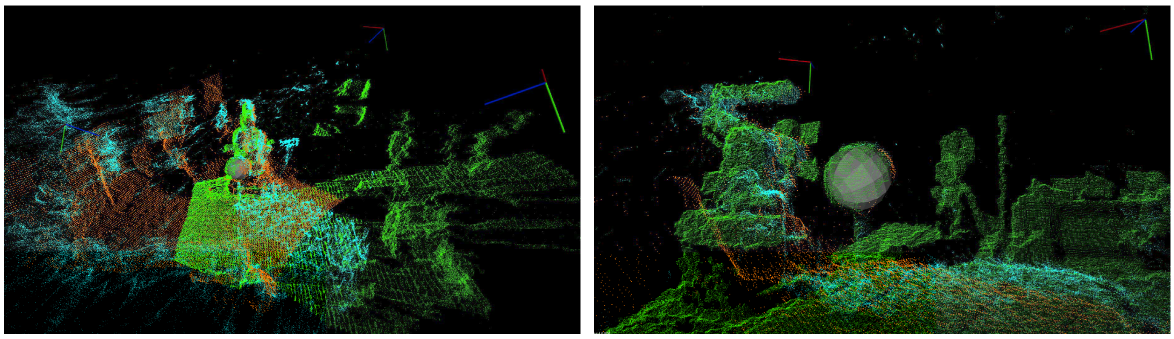

(a) 3D view of the fused point clouds. (b) Close-up view of the calibration target.

Figure 1.6: Fused point clouds results from [9]. SwissRanger SR4000 in orange, Asus Xtion Pro in green and Tyzx stereo camera in cyan.

The calibration method presented by Ruan and Huber in [9] is in principle very similar to the one developed in this work, the major differences are:

- the algorithms and methods employed to detect the ball and compute its center;

- the calibration package presented in this thesis is not restricted to 3D-depth sensors, the package supports 3D-depth sensors, cameras, 3D laser scanners and 2D laser scanners.

1.5.2 Camera-LIDAR calibration at LAR

Within LAR, the first proposed solution to the extrinsic calibration problem was presented by Almeida et al., in [11]. The approach focused on 3D-2D laser scanner extrinsic calibration using a known calibration object. The object, shown in Figure 1.7, consists of two cones connected by an intermediate plane to keep the cones at a fixed distance and to support visual patterns, like a checkerboard, so the method could be expanded to cameras in future works. The conic geometry removes the height ambiguity and the second cone provides information to determine the sensors orientation around their vertical axis.

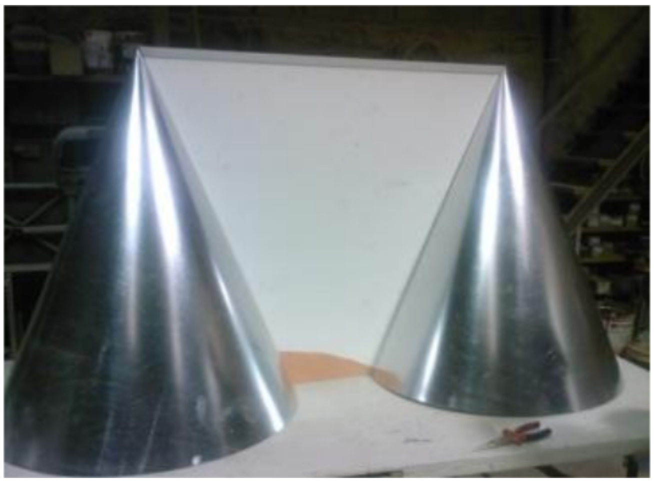
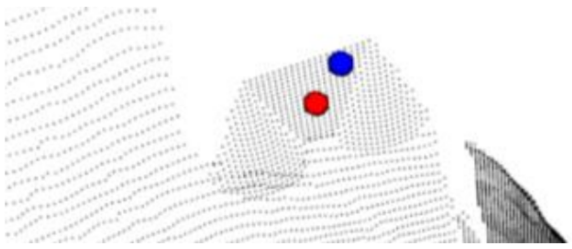


Figure 1.7: Calibration object used in [11].

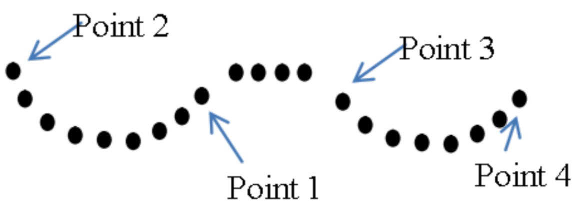
The 3D laser scanner generates a 3D reference point cloud containing the calibration object. To detect the object, the user has to manually select two points, as shown in Figure 1.8a; one at its center, and the other in the intermediate plane between the two cones. The authors then extract the object from the point cloud by filtering the data with a bounding box based on the known object dimensions. Then a virtual calibration object is fit to the extracted data using the Iterative Closest Point (ICP) [12]. The result is a transformation matrix that moves the calibration object from the origin of the 3D point cloud reference system to the calculated position and orientation.

Regarding the 2D data, the calibration object generates two ellipse sections connected by a line, corresponding to the two cones and the intermediate plane between them, respectively. The user has to select two points per ellipse section, as shown in Figure 1.8b. Then, an ellipse fitting algorithm, from Open Source Computer Vision Library (OpenCV), is used to find the ellipse that best fits the extracted points from the 2D point cloud. The algorithm returns the ellipse center, major and minor axis lengths, and the angle between the major axis and the Y axis. An additional step, using OpenCV, is required to compute the angle between the calibration object and the ellipse major axis.

Fitting the ellipse to the cone is a single solution problem for height and roll. However, there is an indetermination around the cone's symmetry axis. The authors solve this geometric



(a) User-selected points for 3D ellipse detection. Red – point at the center of the calibration object; Blue – point in the intermediate plane.



(b) User-selected points for 2D ellipse detection.

Figure 1.8: Calibration object detection in [11].

indetermination by using the previously calculated angle between the calibration object and the ellipse major axis, unequivocally defining the ellipse fitting to the cone.

The 2D laser scanner to 3D laser scanner transformation is estimated by taking four points corresponding to the intersections of major and minor ellipse axis. Hence, a total of eight points are obtained from both ellipses fittings to the 2D laser data. Fitting the ellipses to the calibration objects yield eight more points. Finally, rigid body transform functionalities offered by Visualization ToolKit (VTK) [13] are used to compute the transformation matrix that best transform the eight 2D laser scanner points into the eight 3D laser scanner points. Although the authors did obtain better results with the developed calibration method, results also revealed the rotation along X was not correctly calculated, as can be seen in Figure 1.9. The authors indicate that the error is due to the limited number of points available in the 2D point cloud to fit the ellipse.

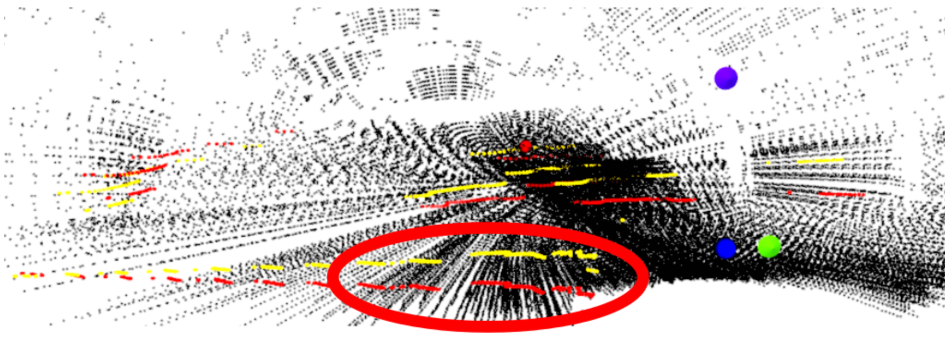
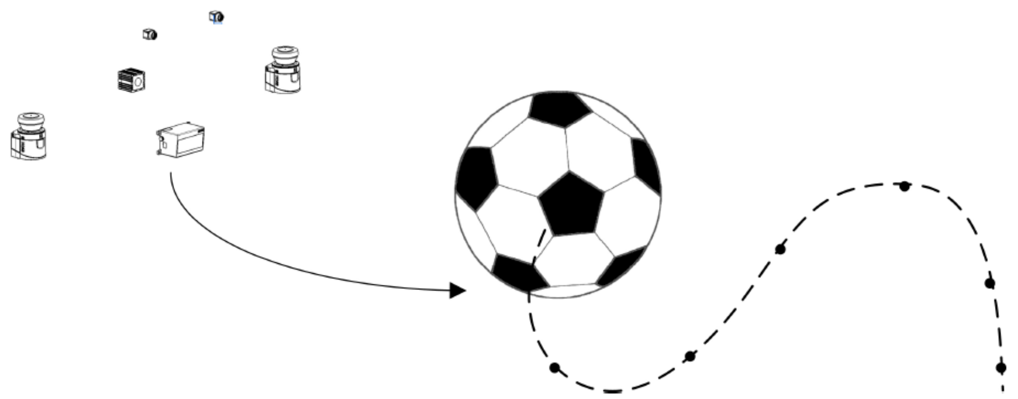


Figure 1.9: Calibration results obtained in [11].

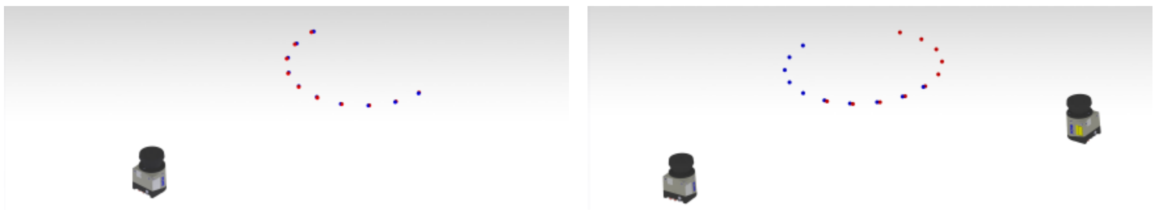
As discussed in the introduction to Chapter 1, the work developed in this thesis is an extension of Pereira [1] calibration method. Due to its importance, it will be described here in more detail.

Pereira developed an automatic extrinsic calibration method. A ball in motion (Figure 1.10a) is used as a calibration target, allowing the uncalibrated sensors (Figure 1.10b) to detect its center along successive positions, generating a point cloud of ball centers detected by each sensor. Aligning the point clouds with respect to a reference frame, allows the author to determine the geometric transformation between each sensor and the reference frame (Figure 1.10c).

Several sensors were used and integrated into the calibration software: two SICK LMS151, one SICK LD-MRS400001, one SwissRanger and two Point Grey cameras in a stereo configuration. One of the SICK LMS151 is chosen as the reference frame, or reference sensor, as the author verified that it is the most accurate.



(a)



(b) (c)

Figure 1.10: (a) Representation of the calibration approach. (b) Uncalibrated (overlapped) sensors. (c) Calibrated sensors. [1]

In order to compute the ball center it is necessary to first detect the ball with all sensors. Since each sensor outputs a different type of data, different ball detection and ball center computation methods are employed for each sensor.

Ball Center Detection on SICK LMS151 Data

The SICK LMS151 is a 2D laser scanner that outputs a 2D point cloud. For this type of data the author applies the following techniques:

1. Data segmentation – Segmentation allows data to be grouped in clusters that have a great probability of belonging to the same object. The segmentation algorithm used is the Spatial Nearest Neighbor (SNN), based on Coimbra's work [14].

2. Circle detection – Circle detection is based on a method previously developed in [15] to detect lines, arcs/circles and legs from laser data. The circle detection technique, named Internal Angle Variance (IAV), makes use of trigonometric proprieties of arcs: every point in an arc has congruent angles with respect to the arc extremes. A circle is positively detected if its mean aperture angle is between 105° and 140° and the standard deviation is below 5°.

3. Computation of the circle properties – The circle center and radius are computed by finding the circle that best fit the points positively detected as being part of a circle.

4. Computation of the ball center – With knowledge of the circle properties, the center of the ball is computed using the simple relation $d = \sqrt{R^2 - R'^2}$, where R is the known ball radius, R' is the detected circle radius and d is the distance between the center of the circle and the center of the ball. At this point, an ambiguity about the ball hemisphere arises, d has two possible solutions; one in the lower hemisphere and the other on the upper hemisphere, consequently the ball center coordinates are $(x_c, y_c, \pm d)$, where x_c and y_c are the circle center coordinates. To solve the ambiguity the user as to supply *a priori* information about the relative position of the sensor in relation to the ball hemisphere. The end-result is shown in Figure 1.11.

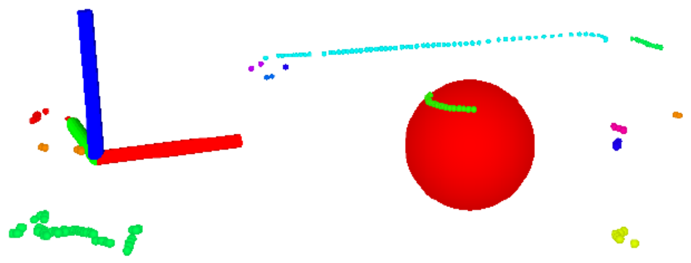


Figure 1.11: Ball detection on SICK LMS151 data [1].

Ball Center Detection on SICK LD-MRS400001 Data

The SICK LD-MRS400001 is a 3D laser scanner with four scan layers. Pereira expected to resolve the hemisphere ambiguity by verifying the following cases:

- if the sensor layers are below the hemisphere, then the circle radius increases from the bottom to the top sensor layers;
- if the hemisphere is within the bottom and top layers, then the circle radius increases before decreasing from the bottom to the top sensor layers;
- if the sensor layers are above the hemisphere, then the circle radius decreases from the bottom to the top sensor layers.

However, the SICK LD-MRS400001 intrinsic error did not allow the cases above to be verified. Consequently, *a priori* information about the relative position of the sensor in relation to the ball hemisphere is also needed for this sensor.

Regarding the ball center detection, each scan layer is treated as a separate 2D scan to which the technique described in the previous section (Section 1.5.2) is applied. After processing all four layers and calculating the center of the ball in at least two of the four scan layers, the mean ball center coordinates is computed. The result of this procedure is shown in Figure 1.12.

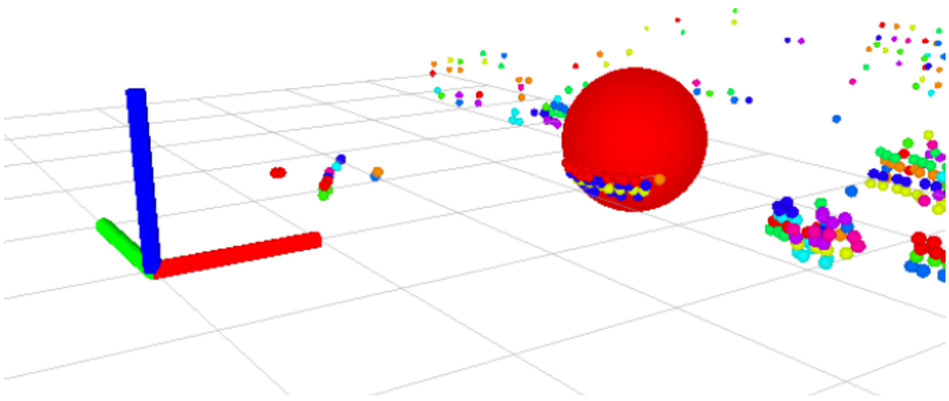
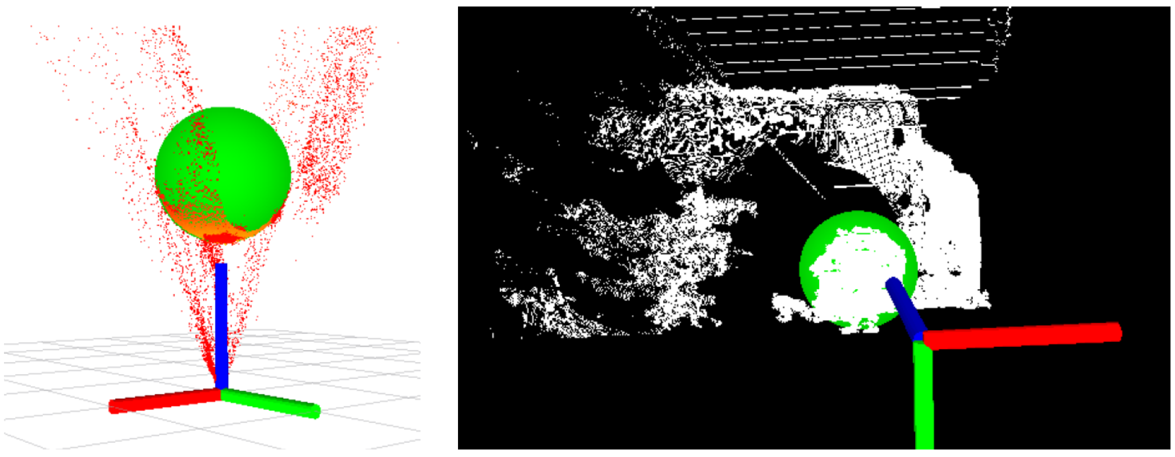


Figure 1.12: Ball detection on SICK LD-MRS400001 data [1].

Ball Center Detection on SwissRanger SR4000 and Stereo Data

Both SwissRanger SR4000 and the Point Grey cameras in a stereo configuration provide 3D point clouds of the scene, in this case ball detection is achieved by using a Point Cloud Library (PCL) algorithm which finds a set of points, from the 3D point cloud, that best fit a mathematical model of a sphere. The algorithm returns the sphere radius and its center. Figure 1.13 shows the result of applying the algorithm on point clouds obtained from the SwissRanger SR4000 and the Point Grey cameras in a stereo configuration.



(a) (b)

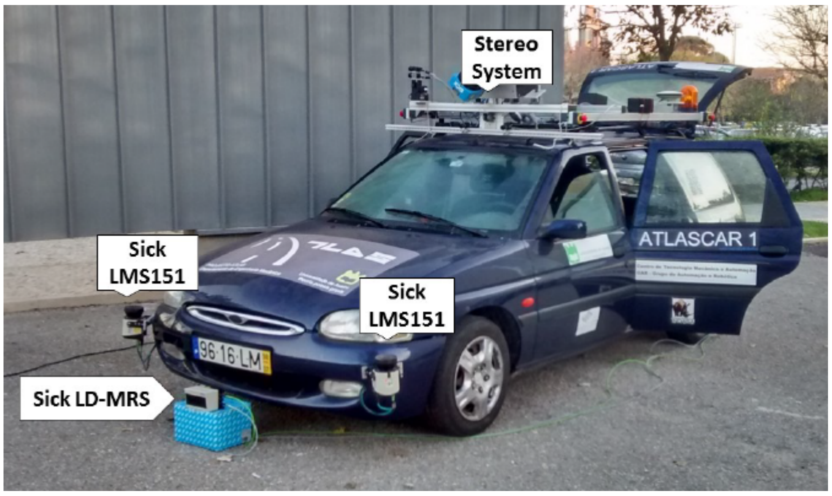
Figure 1.13: Ball detection on: (a) SwissRanger SR4000; (b) Point Grey cameras in a stereo configuration. [1]

3D Rigid Body Transformation Estimation

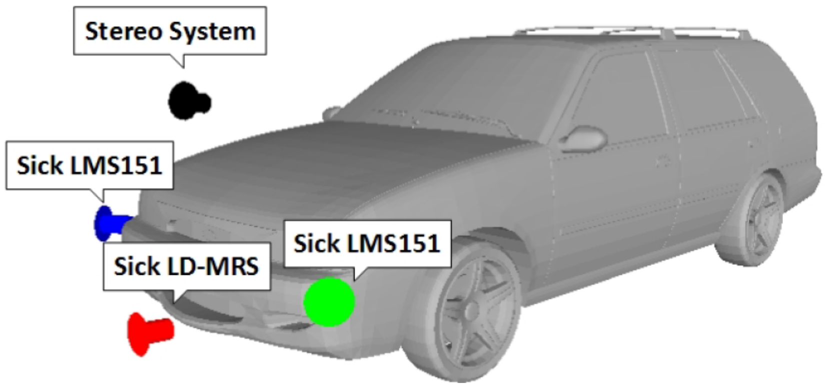
At this stage, the author is able to estimate the transformation between the sensors in a global coordinate frame by matching the ball centers point clouds obtained for each sensor. The transformation between a pair of point clouds is computed from a variant of the ICP algorithm available on PCL, which has two error metrics: point-to-point and point-to-plane. The author chose to use the point-to-point metric. The algorithm then finds the transformation that minimizes the point-to-point error metric by using a closed-form solution based on the Singular Value Decomposition (SVD) method proposed by [16], [17]. The PCL algorithm requires two point clouds as inputs: the target point cloud and the source point cloud. The algorithm finds the transformation applied to the source point cloud that minimizes the chosen error metric.

Extrinsic Calibration Results

Finally, the author tested the proposed extrinsic calibration method with ATLASCAR 1, using two SICK LMS151, one SICK LD-MRS400001 and two Point Grey cameras in a stereo configuration. The sensor setup is shown in Figure 1.14a; Figure 1.14b shows the resulting sensor poses after the calibration procedure. Between the lasers, the resulting poses are very close to their real position and orientation. However, the stereo configuration calibration presents large errors, and is not suitable for real usage.



(a) Sensors setup on ATLASCAR 1.



(b) Sensors pose on a 3D model of a Ford Escort SW (1998). The arrows point in the X direction of each sensor frame.

Figure 1.14: Comparison between the sensors setup and the sensors pose obtained from the calibration method developed in [1].

Chapter 2

Experimental Infrastructure

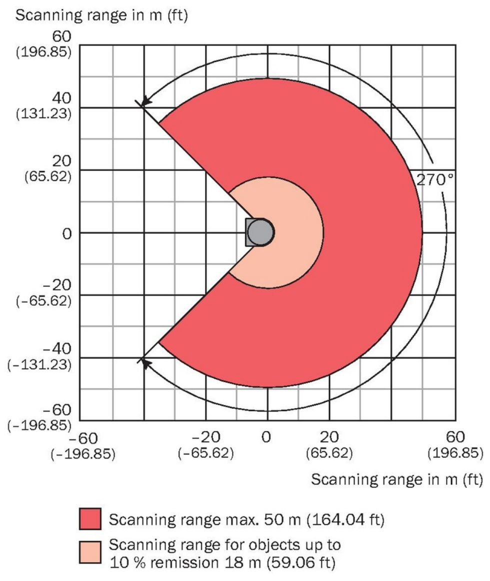
This chapter discusses the hardware and software during the development of this thesis. Section 2.1 presents and describes the sensors integrated in the calibration package. In Section 2.2, a more detailed description of ATLASCAR 1 as a test platform is given. Finally, the main software frameworks and libraries are discussed in Section 2.3.

2.1 Sensors

2.1.1 SICK LMS151

The SICK LMS151 (Figure 2.1a) is a 2D laser scanner suitable for outdoors applications. It is widely used in robotics and autonomous driving applications for its large aperture, angular resolution, high scanning frequency, operating range (Figure 2.1b) and its ability to measure distances through glass, fog and dust (multi-echo technology). Table 2.1 details SICK's LMS151 most relevant specifications.

(a)



(b)

Figure 2.1: SICK SICK LMS151 2D laser scanner (a), and its operating range (b). [18]

Table 2.1: SICK LMS151 specifications [19].

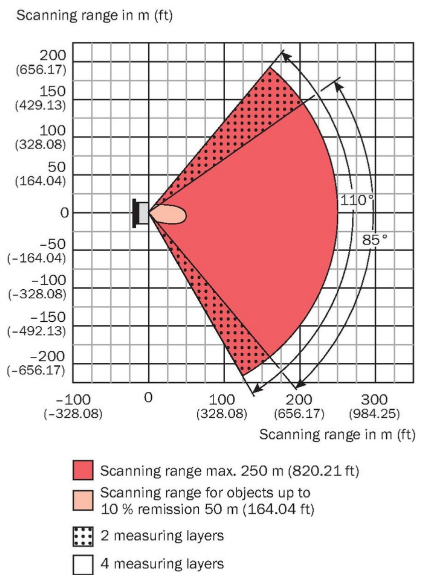
Field of application	Outdoors
Laser class	Class 1, IEC 60825-1 (2007-3)
Aperture angle	270°
Scanning frequency	25 Hz — 50 Hz
Angular resolution	0.25°— 0.5°
Operating range	0.5 ... 50 m
Max. range with 10 % reflectivity	18 m
Systematic error	± 30 mm
Statistical error (1σ)	12 mm
Interface	Ethernet
Weight	1.1 kg

In this work, the SICK LMS151 is used at 50 Hz with a 0.5° angular resolution. Its raw output are 540 points per scan, in polar coordinates (r, θ).

2.1.2 SICK LD-MRS400001

The SICK LD-MRS400001 (Figure 2.2a) is a 3D laser scanner, also suitable for outdoors applications. Four scan planes with a vertical aperture of 0.8° between them, provide 3D capabilities. Like the SICK LMS151, it also makes use of SICK's multi-echo technology on all four scan planes. Its long range, up to 250 meters (Figure 2.2b), and 3D capabilities make it ideal to be placed at the front of autonomous vehicles in order to detect road obstacles at long distances. Table 2.2 lists SICK's LD-MRS400001 most relevant specifications.

(a)



(b)

Figure 2.2: SICK LD-MRS400001 3D laser scanner (a), and its operating range (b) [20].

The sensor can be configured to use four or two scan planes; with two scan planes the aperture angle is increased from 85° to 110°. Independently of the number of scan planes,

Table 2.2: SICK LD-MRS400001 specifications [21].

Field of application	Outdoors
Laser class	Class 1, IEC 60825-1 (2007-3)
Scan planes	4 with a total aperture of 3.2°
Aperture angle	4 scan planes: 85°— 2 scan planes: 110°
Scanning frequency	12.5 Hz — 25 Hz — 50 Hz
Angular resolution	0.125°— 0.25°— 0.5°
Operating range	0.5 ... 250 m
Max. range with 10 % reflectivity	50 m
Systematic error	± 300 mm
Statistical error (1σ)	100 mm
Interface	Ethernet
Weight	1 kg

the LD-MRS400001 has three possible configurations for scanning frequency and angular resolution: 1) 12.5 Hz with 0.125°; 2) 25 Hz with 0.25°; and 3) 50 Hz with 0.5°.

Since the SICK LMS151 has a large aperture angle (270°) the LD-MRS400001 is mainly used for its 3D capabilities. As such, the sensor is configured to use four scan planes with an aperture angle of 85°. Scan frequency and angular resolution are set to 50 Hz and 0.5°, respectively.

2.1.3 Point Grey FL3-GE-28S4-C

The Point Grey FL3-GE-28S4-C (Figure 2.3) is a 2.8 MP color camera with a Sony ICX687 CCD. The defining characteristics of the camera are its high-resolution, small size, low weight and GigE Vision interface. Table 2.3 details the camera's most relevant specifications.

The camera is also highly configurable to suit specific application needs. In this case, even though the camera can output a 1928 × 1448 image at 15 FPS, working at those specifications would saturate the network bandwidth and increase image processing times. A balance between image quality, bandwidth and image processing time is found at a resolution of 960 × 724 pixels, at 15 FPS.

Figure 2.3: Point Grey FL3-GE-28S4-C camera.

Table 2.3: Point Grey FL3-GE-28S4-C specifications [22].

Max. Resolution	1928 × 1448
Max. Framerate	15 FPS @ 1928 × 1448
Megapixels	2.8 MP
Chroma	Color
Sensor	Sony ICX687 1/1.8” CCD
Interface	GigE Vision v1.2
Weight (without optics)	38 g
Dimensions (W×H×D	29 mm × 29 mm × 30 mm

2.1.4 SwissRanger SR4000

The SwissRanger SR4000, shown in Figure 2.4, is a Time-Of-Flight (TOF) camera built by Mesa Imaging. A TOF camera measures the time taken for a light signal – in this case, Infrared (IR) – to travel from its source to an IR reflective object and back to the sensor [23].

Figure 2.4: SwissRanger SR4000 TOF camera.

The illumination source is an array of 24 LEDs, and an optical filter at the front of the sensor only allows light of wavelengths close to the wavelength of the source LEDs to pass into the camera. Despite the optical filter, the sensor is designed for indoor use as direct sunlight interferes with the device, causing wrong measurements and areas of random noise.

Being limited to indoors use and with a maximum radial range of five meters, the SwissRanger SR4000 is not suitable for autonomous driving applications. However, its compact size, low weight and high frame rate make it an attractive solution for robotic manipulators and humanoids. A more detailed description about the SR4000 is found in Table 2.4.

Similarly to laser scanners, the SR4000, and TOF cameras in general, provide a point cloud where each point corresponds to a laser obstacle. In fact, TOF cameras are a class of LIDAR. The major difference between TOF cameras and laser scanners is that cameras capture the entire scene with each laser pulse, while laser scanners capture the scene point-by-point with a laser beam.

Table 2.4: SwissRanger SR4000 specifications [24].

Field of application	Indoors
Operating range	0.1 ... 5 meters
Frame Rate	50 FPS
Pixel array size (h$\times$v)	176$\times$144
Field of view (h$\times$v)	43.6°$\times$ 34.6°
Angular resolution	0.24°
Systematic error	$\pm$ 10 mm
Statistical error (1σ)	4 mm
Interface	USB
Weight (without optics)	470 g
Dimensions (W$\times$H$\times$D	65 x 65 x 68 mm

2.1.5 Microsoft Kinect

Microsoft Kinect (Figure 2.5) is a sensor developed by Microsoft for the Xbox 360 console. However, it is also widely used in robotics since it packs multiple built-in sensors with very good performance for its cost. Kinect contains the following sensors:

1. an RGB camera;
2. a 3D-depth structured light sensor consisting of an IR laser projector and an IR camera to measure the distance between the camera and the objects;
3. an array of four microphones to record sound and find its source location;
4. a 3-axis accelerometer to determine the Kinect's current orientation.

Figure 2.5: Microsoft Kinect.

This work will focus on the 3D-depth sensor. As indicated above, the 3D-depth sensor includes a IR projector and an IR camera; the projector casts an IR speckle dot pattern while the camera captures the reflected speckles. The camera-to-object distance is obtained through an off-line calibration procedure [25].

Table 2.5 lists the main specifications of the Kinect 3D-depth sensor. A comparison between the Kinect and the SR4000 is inevitable as both provide the same kind of information, a 3D point cloud of their surroundings. Smisek *et al.* [26] made a direct comparison between Kinect 3D-depth sensor, a SLR stereo rig (consisting of two Nikon D60) and a SwissRanger SR4000, concluding that Kinect is more accurate than the SR4000 and slightly less accurate than the SLR stereo rig, for a fraction of the cost.

Table 2.5: Microsoft Kinect 3D-depth sensor [27].

Field of application	Indoors
Operating range	0.8 ... 3.5 meters
Frame Rate	30 FPS
Pixel array size (h×v)	640×480
Field of view (h×v)	57°× 43°
Interface	USB

2.2 Sensors setup on ATLASCAR 1

As presented in Section 1.1, ATLASCAR 1 is a prototype for AD and ADAS development, equipped with several sensors to perceive the surrounding environment as well as its internal systems. Figure 2.6 details the sensors currently equipped on the vehicle that perceive the surrounding environment. The LD-MRS400001 is the newest addition, and with the future migration to ATLASCAR 2, the sensor doesn't have a support to be attached to ATLASCAR 1. Nonetheless, Figure 2.6 shows its planned position.

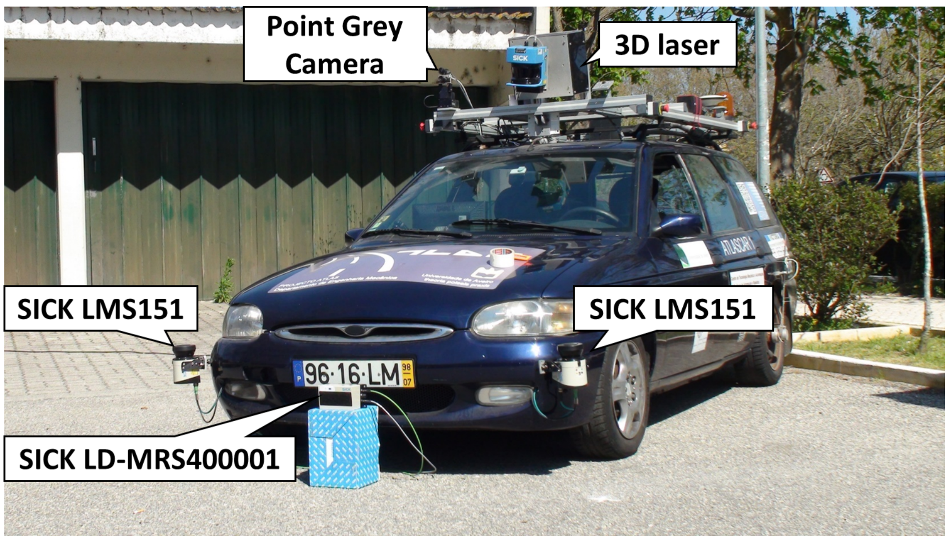


Figure 2.6: Currently equipped sensors on ATLASCAR 1.

Tests on ATLASCAR 1 will focus on the two SICKs LMS151, on the SICK LD-MRS400001 and on the Point Grey FL3-GE-28S4-C camera. Both Kinect and SwissRanger SR4000 are indoor sensors, consequently they are not used in outdoors tests with ATLASCAR 1.

Regarding the 3D laser in Figure 2.6, it is a 2D SICK LMS200 planar laser scanner, proposed originally in [28] and further modified to its actual configuration in [29]–[31]. The sensor is not used in this work, as its frame rate is far too low to correctly detect a moving ball.

2.3 Software

2.3.1 PCL

Technological advancements in low-cost 3D-sensing hardware such as the Kinect, led to the massification of real time 3D point clouds. Following the hardware technological advancement, Rusu and Cousins [32], presented in 2011 the PCL.

PCL is a large scale open-source framework for 2D and 3D point cloud processing written in C++. The framework contains numerous state-of-the-art algorithms that operate on point cloud data, including: filtering, feature estimation, surface reconstruction, model fitting, segmentation, registration and others. Each set of algorithms are defined through base classes that attempt to integrate all common functionality used throughout the entire pipeline. Generally, the PCL processing pipeline consists of the following: [32]

1. create the processing object (a filter, feature estimator, segmentation and others);
2. use *setInputCloud* to pass the input point cloud to the processing module;
3. set needed parameters;
4. call compute, or filter, segment and others to get the result.

PCL is essential to the calibration package developed here, since the SICK LMS151, SICK LD-MRS400001, SwissRanger SR4000 and Microsoft Kinect all output a point cloud that has to be processed in order to detect the ball.

PCL is also cross-platform, and has been ported to Linux, MacOS, Windows, Android and iOS. To simplify development, it has been split into smaller libraries or modules (Figure 2.7), that can be compiled independently from each other [32].

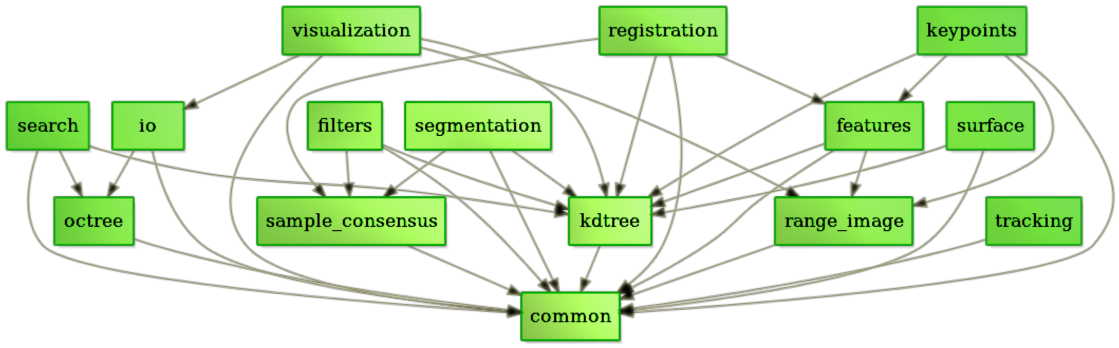


Figure 2.7: PCL modules dependency graph [33].

The most relevant modules for this work are: *common*, *io*, *filters* and *sample_consensus*.

Module *common*

The *pcl_common* library is the base for all other PCL modules. It contains structures, like the PointCloud class, and methods used by the majority of the other modules. It also contains a variety of functions to compute norms, means, covariance, angular conversions, transformations, and many others.

Module *io*

The *pcl_io* library contains classes and functions for reading and writing Point Cloud Data (PCD) files. Point clouds resulting from the calibration procedure are saved and read as PCD files.

Module *filters*

The *pcl_filters* library includes noise reduction/removal and outlier removal. Noise reduction/removal is implemented through downsampling filters, passthrough filters, color filters, and more. Filters can greatly reduce processing time in large point clouds with a small impact on the end result.

Module *sample_consensus*

The *pcl_sample_consensus* library holds SAmple Consensus (SAC) methods like RANSAC and parametric models for planes, cylinders and, more importantly for this work, spheres. This library allows objects with specific geometric features to be detected in the point cloud.

2.3.2 OpenCV

OpenCV is a computer vision and machine learning software library, presented by Bradski [34] in 2000. It was built to provide a common infrastructure for computer vision applications. Although it is written in C++, it can be used with C++, C, Python and MATLAB and is supported by Linux, Windows, MacOS, Android and iOS.

Like PCL, OpenCV includes numerous state-of-the-art algorithms. These algorithms can be used to detect faces, classify human actions in videos, track camera movements, track objects, scenery and object recognition and much more. OpenCV also has a modular structure allowing modules to be compiled independently from each other. The structure includes the following modules [35]: *core*, *imgproc*, *video*, *calib3d*, *features2d*, *objdetect*, *highui*, *gpu* and other small helper modules.

OpenCV is used for processing Point Grey FL3-GE-28S4-C images by using the *core*, *imgproc*, *calib3d* and *highui* modules.

Module *core*

The *core* module defines basic data structures, like the multi-dimensional array *Mat*, methods and functions used by all other modules.

Module *imgproc*

The *imgproc* module is an image processing module including linear and non-linear image filtering, geometrical image transformations, color space conversions, histograms, drawing functions, feature detection and more.

Module *calib3d*

The *calib3d* module contains single and stereo calibration algorithms, object pose estimation, stereo correspondences algorithms and 3D reconstruction elements. This module is used for its calibration and pose estimation algorithms.

Module *highui*

The *highui* module is an easy-to-use and develop interface for video capturing, image and video codecs with simple User Interface (UI) capabilities. This allows the calibration package to control image processing parameters trough a simple OpenCV-based UI.

2.3.3 Qt

Qt is the chosen framework to develop the calibration package GUI. It's a leading cross-platform application development framework written in C++. Supported platforms include Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS and others [36].

Qt is currently developed by Qt Project, which consists of multiple companies and individuals developing under open-source governance. It is also available under multiple licenses, as a commercial license and as GPL and LGPL licenses [36].

As a GUI framework, Qt is very well documented, has a modern look and is visual oriented. Since its early releases, Qt also relies on three key concepts to differentiate itself from other GUI toolkits: Qt platform abstraction; signals and slots; and, the Meta-Object Compiler (MOC) preprocessor.

Qt Platform Abstraction

Qt itself is separated into two layers. One is its core functionality, implemented in standard C++, which is essentially platform-independent. The second layer is a set of plugins called the "Qt platform abstraction" that contains all platform-specific code related to creating windows, drawing, using fonts, and more. Therefore, porting Qt to a new platform (provided it uses a supported C++ compiler) is essentially implementing the Qt platform abstraction set of plugins. As a result, Qt is easily portable to new platforms and is currently supported in many desktop, embedded and mobile platforms [37].

Signals and Slots

Signals and slots are used for communication between Qt objects, replacing callbacks. A signal is emitted when a particular event occurs. Qt's widgets have predefined signals, but more can always be added. A slot is a function that is called in response to a particular signal [38].

MOC preprocessor

The MOC preprocessor is run on Qt applications source files, looking for the Q_OBJECT macro. It uses them to generate C++ code containing the meta-object code for classes used in the application. The information is used by Qt to provide programming features not supported directly in C++, like signals and slots, introspection and asynchronous function calls [39].

2.3.4 ROS

Robot Operating System (ROS) is a collection of software frameworks for robot software development, providing operating system-like functionality. ROS provides hardware abstraction, low-level device control, implementation of commonly used functionalities, communication between processes and package management. ROS is written in C++, Python and LISP, is open source and supports Ubuntu Linux while other systems – Fedora Linux, OS X, Windows and Android – are supported by the community.

The fundamental concepts of ROS implementation, according to [40], are: packages, nodes, topics and services. The following sections describe each of the fundamental ROS concepts, as well as, two key packages for this work, Rviz and roslaunch.

Packages

A package is a collection of programs and/or libraries that perform a function. The package structure usually contains: a *src* subdirectory where the source code is stored; an *include* subdirectory where source headers are stored; a *manifest.xml* file which is a declaration of the package's dependencies and information about the package maintainer; and a *CMakeList.txt* file with instructions for standard *CMake* compilation.

Nodes

Nodes are processes that perform computation. A package is typically comprised of many nodes that can exchange information via messages. Nodes can publish or receive messages by creating topics.

Topics

Topics are simply strings that identify a message within the ROS structure. Nodes that create and send messages to the topic are called publisher nodes. Nodes receiving the messages are called subscriber nodes. To deal with the asynchronous nature of ROS, publishers and subscribers have message queues which store messages until the defined queue limit is reached.

Services

For synchronous transactions and Remote Procedure Call (RPC) request/reply interactions, ROS provides services. Services are defined by a string and a pair of strictly typed messages, one of request and one for reply, analogous to web services. Unlike topics, services can only be advertised by one node.

Rviz

Rviz is a 3D visualizer for robots integrated into ROS. Rviz provides and easy-to-use but powerful GUI to display numerous built-in message types by subscribing to topics.

It also includes a library, *librviz*, allowing most of its functionalities to be accessed by other applications. The library allows a Rviz-like visualizer to be included in the calibration package GUI developed in this work.

Roslaunch Package

The roslaunch ROS package contains the roslaunch tools, which read XML configuration files (with a *.launch* extension) that specify the nodes to launch, parameters to set and other options. These characteristics greatly simplify launching and configuring multiple nodes.

The *XML* configuration files includes several tags, described extensively in [41], of which the following are the most relevant for this work:

- <launch> – root element of any roslaunch file. It acts as a container for the other elements;
- <arg> – defines an argument. Allows the launch file to be configurable through values passed via the command-line;
- <group> – applies settings to a group of nodes. The "ns" attribute sets the group namespace;
- <node> – specifies a ROS node to be launched. Contains several attributes, of which the following are the most relevant for this work:
  - name – node name;
  - pkg – node package name;
  - type – node executable;
  - required – if set to "*true*" the entire roslaunch is killed if the node dies. If set to "*false*", roslaunch is not terminated if the node dies.
- <param> – defines a parameter within the scope it is inserted. Its relevant attributes, for this work, are:
  - name – parameter name;
  - type – parameter data type. Can be set to *str*, *int*, *double* or *bool*;
  - value – sets the value of the parameter.

2.3.5 Calibration Package Dependencies

The calibration package developed in this work has various package dependencies:

- csiro-asl-ros-pkg – provides a ROS driver node for the SICK LD-MRS400001 laser scanner;
- RCPRG_laser_drivers – provides a ROS driver node for the SICK LMS151 laser scanner;
- libmesasr-dev – operating system driver for the SwissRanger SR4000;
- swissranger_camera-master – provides a ROS driver node for the SwissRanger SR4000 device;
- FlyCapture 2.x Software Development Kit (SDK) – provides a complete software Application Programming Interface (API) library for Point Grey cameras;

- colormap – the colormap ROS package provides similar functionality as MATLAB colormap. Included on Laboratory for Automation and Robotics Toolkit (LARtk) [42];
- lidar_segmentation – provides 2D LIDAR segmentation algorithms. Included on LARtk [42].

2.4 Summary

This chapter presented and described the sensors and software used in this work. ROS is the base architecture that connects the hardware to the software. ROS drivers publish data acquired from the sensors while PCL and OpenCV process the data according to their type efficiently.

Qt is the chosen framework to develop the calibration package GUI, which will integrate some of the Rviz functionalities by using the *librviz* library. It is also noteworthy that *librviz* is based on Qt version 4.8.6 consequently the GUI is also going to be developed with Qt 4.8.6.

Chapter 3

Camera-LIDAR Calibration

This chapter presents the camera-LIDAR extrinsic calibration methodology. The main goal is to obtain an extrinsic calibration package for cameras, 2D and 3D sensors; allowing calibration of the sensors used on ATLASCAR 1 – two SICK LMS151, one SICK LD-MRS400001 and a Point Grey FL3-GE-28S4-C camera. Pereira's work [1], described in Section 1.5.2, is used for the extrinsic calibration between the LIDAR sensors. The stereo calibration method developed by Pereira is not used in this work, as it did not present reliable results. Additionally, the Point Grey FL3-GE-28S4-C cameras on ATLASCAR 1 are normally not used in a stereo configuration, therefore, this work focuses on a methodology for monocular camera-LIDAR calibration.

Pereira also explored camera-LIDAR calibration, but failed to reliably detect the ball available at the time (Figure 3.1a), advising the use of a uniform color ball in future works. Following Pereira's recommendation a new ball with uniform color has been acquired, shown in Figure 3.1b.

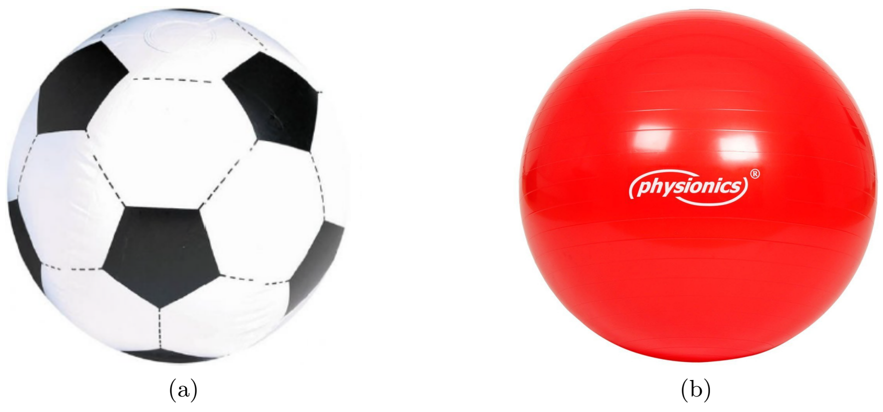


Figure 3.1: (a) Calibration ball (Ø=107 cm) used by Pereira [1]. (b) Calibration ball used in this thesis (Ø=95 cm).

The ball as a calibration target presents several interesting characteristics:

- for 2D data, any planar section of the ball is a circle, a well-known geometry that is easy to detect;

- for 3D sensors a partial sphere is acquired, independently of the sensor point-of-view, which can then be fit to a mathematical model of a sphere;
- in rectified camera images, the ball will be perceived as a circle with a uniform color, facilitating its detection.

3.1 Intrinsic Camera Calibration

Before presenting further details about LIDAR-camera extrinsic calibration, this section will discuss intrinsic camera calibration. Intrinsic camera calibration is the process of determining the parameters of a lens and image sensor of a camera.

The intrinsic parameters, according to the pinhole camera model, are the focal length f_x and f_y in pixels, the optical centers c_x and c_y also in pixels and the skew coefficient s. The matrix that contains these parameters is called the camera matrix (Eq. (3.1)).

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

There are many available tools for intrinsic camera calibration, including in ROS. The *camera_calibration* package for ROS allows the calibration of monocular or stereo cameras using a checkerboard as a calibration target with a simple GUI (Figure 3.2). The package uses OpenCV camera calibration as described in [43], which assumes a skew coefficient equal to zero, thus the camera matrix becomes Eq. (3.2).

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

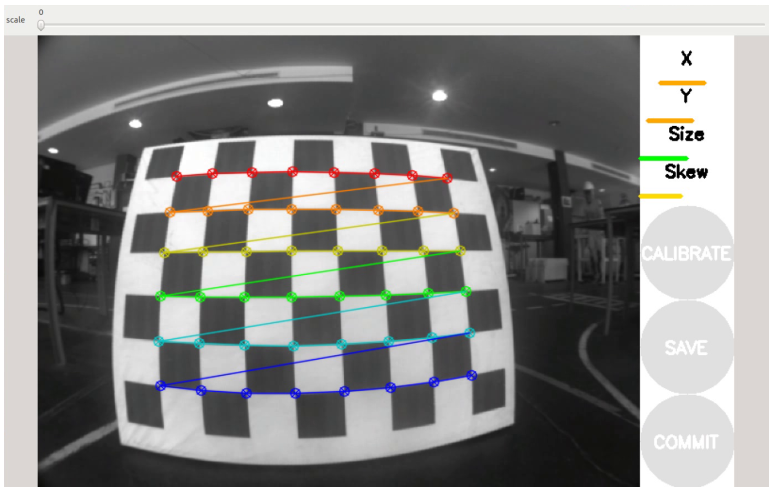


Figure 3.2: The ROS *camera_calibration* package GUI.

The OpenCV algorithm is based on the camera model proposed by J. Bouguet [44] which computes the camera matrix from Eq. (3.2) and is also able to estimate five lens distortions coefficients: three radial distortion coefficients (k_1, k_2 and k_3) and two tangential distortion coefficients (p_1 and p_2). These coefficients are used to correct images acquired from the Point Grey camera, as shown in Figure 3.3.

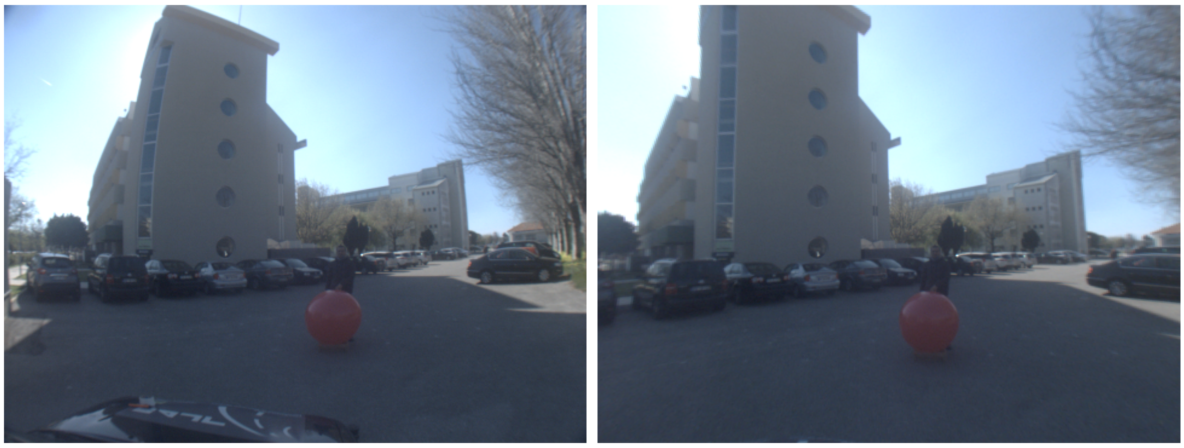

(a) Acquired image. (b) Undistorted image.

Figure 3.3: Camera lens distortion correction.

3.2 Ball Detection in an Image

3.2.1 Hough Circle Transform

Since the ball appears as a circle on a rectified camera image, the first approach was to use the Hough Transform Circles implemented in OpenCV. The Hough transform is a method for finding lines, circles and other simple geometric lines in an image.

The boundary of a circle can be described by Eq. (3.3), where (x_c, y_c) is the center of the circle, r is the radius circle and (x, y) is any 2D point.

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \tag{3.3}$$

From Eq. (3.3), it is clear that three parameters define a circle (x_c, y_c, r), thus a 3D accumulator is needed, which would be computationally demanding. Therefore, the implementation of Hough circle transform in OpenCV relies on the Hough gradient method [45].

According to [43], the Hough gradient method works by first passing a greyscale image through a Canny edge detector (*Canny*() in OpenCV). Next, for every nonzero point in the edge image, the local gradient is computed using a first order Sobel derivative (*Sobel*() in OpenCV). Every point along the gradient line and inside a minimum and maximum distance is incremented in the accumulator. The potential centers are then selected from the points in the accumulator that are both above a given threshold and with more support than all of their immediate neighbors. Finally, for each center all the nonzero pixels are considered and sorted, in ascending order, according to their distance from the circle center. A single radius is selected that is best supported by nonzero pixels and if it is a sufficient distance from any previously selected center.

Along with a greyscale image, *HoughCircles* requires the user to specify the following parameters:

- *minDist* – minimum distance between the centers of the detected circles. Too small and multiple neighbor circles may be falsely detected. Too large and some circles may be missed;
- *param1* – higher Canny edge detector threshold;
- *param2* – the accumulator threshold for circle centers. Small values may lead to detecting more false circles. Large values may return not return a circle;
- *minRadius* – minimum circle radius in pixels;
- *maxRadius* – maximum circle radius in pixels.

Initially, a greyscale image, without any pre-processing, was passed to the *HoughCircles* function. Results proved to be highly unreliable, even after fine-tuning the parameters described above. An example image is shown in Figure 3.4 – Figure 3.4a shows the input greyscale image, Figure 3.4b shows the Canny edges map and Figure 3.4c shows the circles detected in the input image with the parameters found in Table 3.1a. Observing the detected edges it is possible to see that the ball edge is not completely detected by the Canny edge detector, experimentation with lower Canny thresholds led to significant increase in noise; higher values resulted in the edge of the ball not being detected at all.

In order to improve results, a color threshold is applied to the image acquired from the camera. Color thresholding creates a binary image, where pixels within the specified color range are white, and pixels outside that range are black. Next, a morphological closing followed by an opening is applied to the image. A Gaussian blur filter is also applied to the image to remove noise. The goal of these pre-processing operations is to remove every object excluding the ball from the image, so that the Canny edge detector can find its edge easily. This approach resulted in a significant improvement in circle detection, as shown in Figure 3.5, obtained with the parameters described in Table 3.1b.

Table 3.1: Hough circle transform parameters.

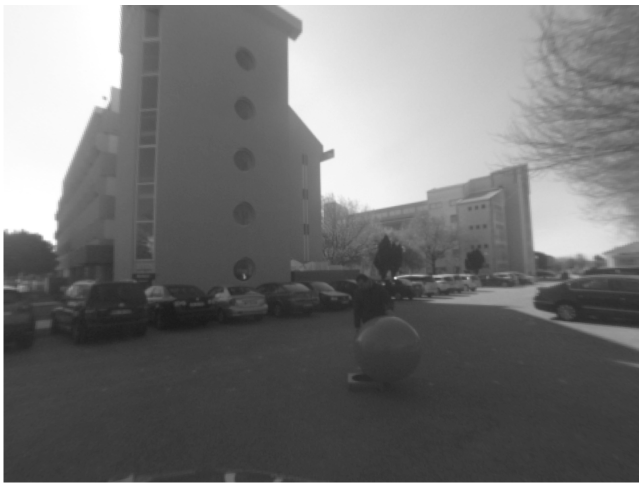
(a) Parameters used for ball detection in Figure 3.4.

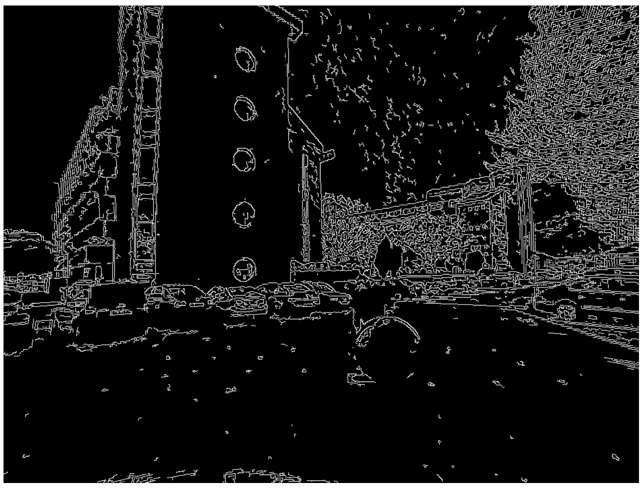
Min. Distance (pixels)	240
Canny Thresh.	9
Accumulator Thresh.	16
Radius (pixels)	$[49, 56]$

(b) Parameters used for ball detection in Figure 3.5.

Hue	$[142, 179]$
Saturation	$[45, 255]$
Value	$[0, 255]$
Min. Distance (pixels)	240
Canny Thresh.	9
Accumulator Thresh.	9
Radius (pixels)	$[49, 56]$

The limitation of this approach is the necessity to constantly tune the parameters in Table 3.1b in order to reliably detect the circle. Even if the ideal range of HSV color, and values of minimum distance, Canny threshold and accumulator threshold could be found, the circle radius range always requires adjustments as the ball moves closer and further away from the camera. Consequently, the Hough circle transform ball detection method is not suitable for an automatic calibration package.


(a) Input greyscale image.


(b) Canny edges map.

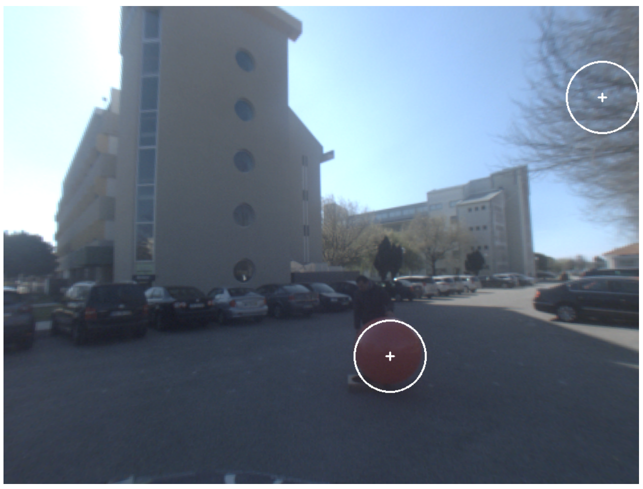
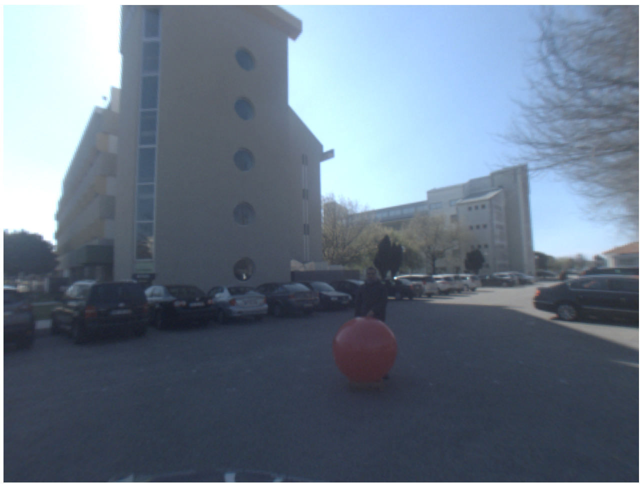
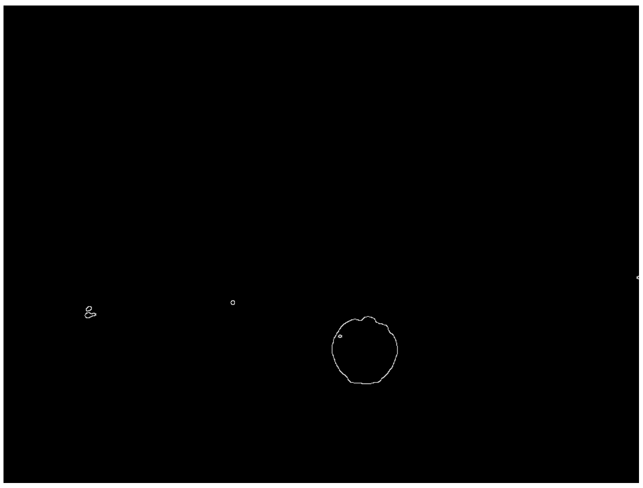

(c) Circles detected in the input image.

Figure 3.4: Hough circle transform method results without pre-processing.


(a) Input undistorted image.


(b) Canny edges map.

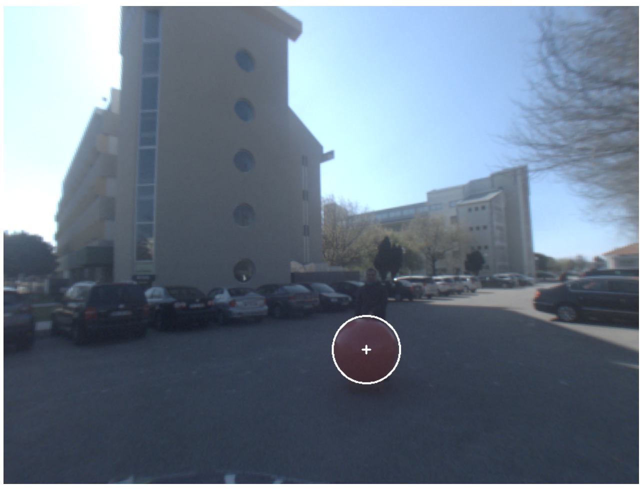

(c) Circles detected in the input image.

Figure 3.5: Hough circle transform method results with pre-processing.

3.2.2 Approximated Contour

Therefore, a second algorithm was implemented which can be broken down in three main steps: 1) Finding contours in the image; 2) Contour approximation by a polygonal curve; and, 3) Ball contour detection and circle properties.

Finding contours in the image

Contours are simply a set of points that represent a closed curve in an image. Thus, contours are used to obtain the contour of objects, in this case a ball, represented by a circle in the image. OpenCV already includes an implementation of a contour finding function, *findContours*(), which takes a binary image as its input. To obtain this binary image the same pre-processing operations are applied to an image acquired by the camera – color thresholding, morphological closing, morphological opening and noise removal.

findContours() returns every contour in the binary image. Ideally the image would only contain one object, the ball; thus, only the desired contour would be returned. However such conditions are rare, especially outdoors.

Contour approximation by a polygonal curve

To increase the robustness of this algorithm each contour is approximated by a polygonal curve. OpenCV also has a function for this purpose, based on the Ramer-Douglas-Peucker algorithm [46], [47].

The implementation works by first picking two extremal points and connecting them with a line. Then the original curve is searched to find the point farthest from the line just drawn, and that point is added to the approximated curve; this step is repeated until all points in the original curve are bellow a distance threshold relative to the approximated contour [43]. The distance threshold has been empirically set to 2% of the initial curve length.

Ball contour detection and circle properties

It is now possible to find the approximated contour that represents the ball. Approximated contours with 6 or more vertices are considered potential circles. The area limited by the original contour is calculated ($A_{contour}$), and an up-right bounding rectangle is computed around the approximated contour in order to calculate its width ($w_{b.r}$) and height ($h_{b.r}$). A circle is positively detected when: i) the bounding rectangle has similar height and width – Eq. (3.5); and ii) the area limited by the contour is similar to the area of a circle with radius given by Eq. (3.4) – Eq. (3.6). Thresholds on Eq. (3.5) and (3.6) were obtained through empirical tests.

$$R_{pix} = \frac{w_{b.r} + h_{b.r}}{4} \tag{3.4}$$

$$\left|1 - \frac{w_{b.r}}{h_{b.r}}\right| \leqslant 0.2 \tag{3.5}$$

$$\left|1 - \frac{A_{contour}}{\pi R_{pix}^2}\right| \leqslant 0.2 \tag{3.6}$$

After successfully detecting the calibration ball, its radius is determined by Eq. (3.4) and its center $(x_{c.pix}, y_{c.pix})$ can be computed from the bounding box.

Figure 3.6 illustrates every step of the algorithm, from the acquired image to the final image where the calibration ball is successfully detected using the HSV color range in Table 3.2.

Table 3.2: Parameters used for ball detection in Figure 3.6.

Hue	$[142, 179]$
Saturation	$[45, 255]$
Value	$[0, 255]$

In conclusion, the algorithm proved to be more reliable than the Hough circle transform methods presented in the previous section, furthermore, it only requires a Hue, Saturation, and Value (HSV) color range for thresholding. In conditions of uniform lighting, the color range is set at the start of the calibration procedure and does not require further adjustments.

3.3 Ball Center in the Camera's Frame

This section describes how the ball center coordinates in the image coordinate system (image points) are transformed to the camera coordinate system (world points).

According to the pinhole camera model, Eq. (3.7) transforms world points into image points. However, the goal is to transform image points into world points, this is an ill-posed problem as there are three unknowns – X_c, Y_c and Z_c – for two equations, since the third equation reduces itself to $Z_c = Z_c$.

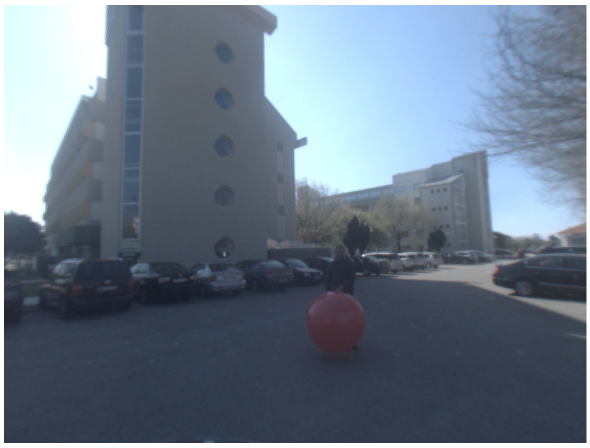
$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = Z_c \begin{bmatrix} x_{c.pix} \\ y_{c.pix} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{3.7}$$

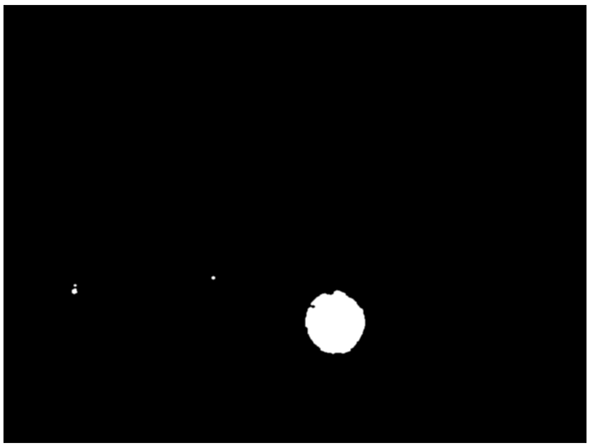
Taking advantage of the known ball diameter and the up-right bounding rectangle around the ball's approximated contour, it is possible to calculate Z_c. Figure 3.7 illustrates an up-right bounding rectangle around an approximated ball contour. Taking the two opposite corners of the rectangle, $P_1 = (x_{pix,1}, y_{pix,1})$ and $P_2 = (x_{pix,2}, y_{pix,2})$, and writing Eq. (3.7) for both points, results in Eq. (3.8) and (3.9), for P_1 and P_2, respectively.

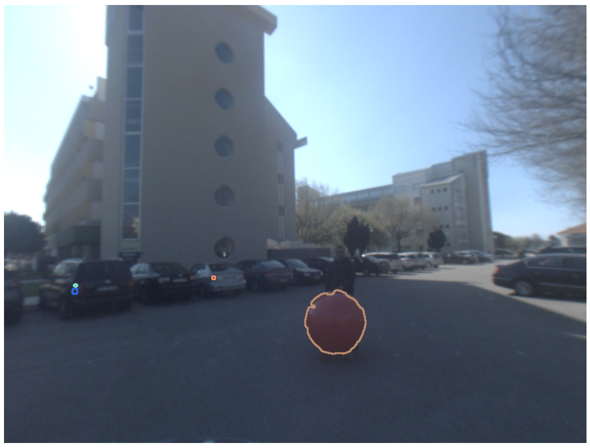
$$Z_{c,1} \begin{bmatrix} x_{pix,1} \\ y_{pix,1} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{c,1} \\ Y_{c,1} \\ Z_{c,1} \end{bmatrix} \iff \begin{cases} Z_{c,1} x_{pix,1} = f_x X_{c,1} + c_x Z_{c,1} \\ Z_{c,1} y_{pix,1} = f_y Y_{c,1} + c_y Z_{c,1} \\ Z_{c,1} = Z_{c,1} \end{cases} \tag{3.8}$$

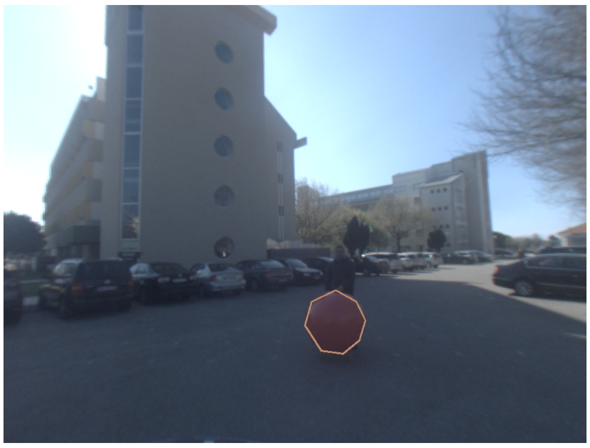
$$Z_{c,2} \begin{bmatrix} x_{pix,2} \\ y_{pix,2} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{c,2} \\ Y_{c,2} \\ Z_{c,2} \end{bmatrix} \iff \begin{cases} Z_{c,2} x_{pix,2} = f_x X_{c,2} + c_x Z_{c,2} \\ Z_{c,2} y_{pix,2} = f_y Y_{c,2} + c_y Z_{c,2} \\ Z_{c,2} = Z_{c,2} \end{cases} \tag{3.9}$$

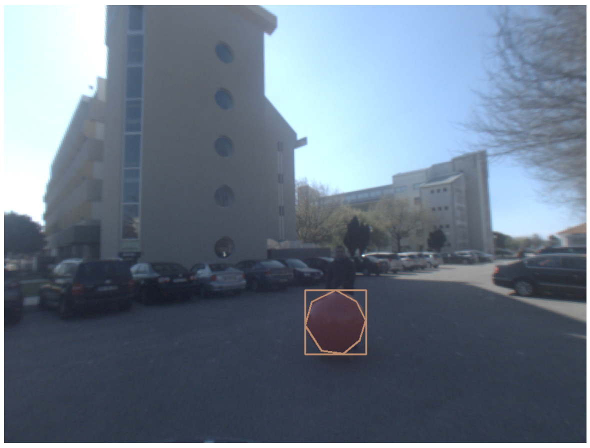
Since the bounding rectangle lies in the $\mathrm{X_c Y_c}$-plane, $Z_{c,1}$ and $Z_{c,2}$ are equal to Z_c. Subtracting Eq. (3.8) and (3.9) yields


(a) Undistorted image acquired from Point Grey camera.


(b) Binary image after color thresholding (see Table 3.2).


(c) Contours found by *findContours*().


(d) Approximated contour from the circle that represents the calibration ball.


(e) Up-right bounding rectangle computed around the approximated contour.

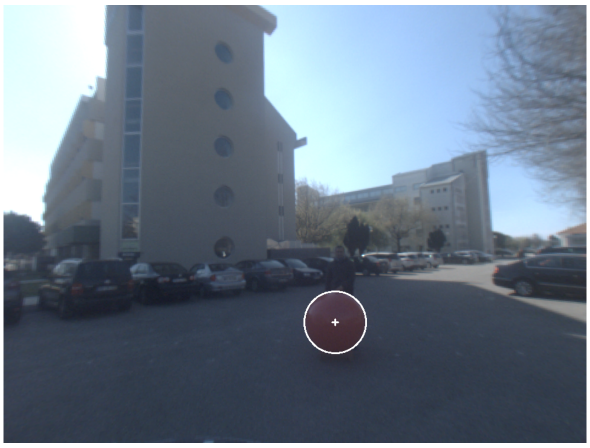

(f) Detected circle that represents the calibration ball.

Figure 3.6: Illustration of the approximated contour algorithm.

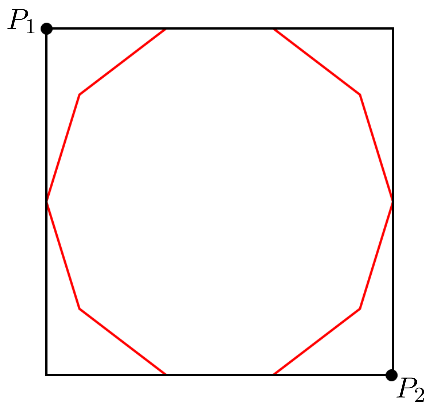


Figure 3.7: An up-right bounding rectangle around an approximated ball contour.

$$\begin{cases} Z_c\left(x_{pix,2} - x_{pix,1}\right) = f_x\left(X_{c,2} - X_{c,1}\right) \\ Z_c\left(y_{pix,2} - y_{pix,1}\right) = f_y\left(Y_{c,2} - Y_{c,1}\right) \ . \end{cases} \tag{3.10}$$

$x_{pix,2} - x_{pix,1}$ and $y_{pix,2} - y_{pix,1}$ are simply the bounding rectangle's width ($w_{b.r}$) and height ($h_{b.r}$), respectively. Similarly, $X_{c,2} - X_{c,1}$ and $Y_{c,2} - Y_{c,1}$ are the bounding rectangle's width and height in the camera's frame which are both equal to the ball diameter, D. Thus, Eq. (3.10) can be rewritten as

$$\begin{cases} Z_c w_{b.r} = f_x D \\ Z_c h_{b.r} = f_y D \ . \end{cases} \tag{3.11}$$

Solving Eq. (3.4) for $w_{b.r}$ gives

$$w_{b.r} = 2D_{pix} - h_{b.r} \ , \tag{3.12}$$

substituting Eq. (3.12) into the first equation from Eq. (3.11) yields

$$Z_c\left(2D_{pix} - h_{b.r}\right) = f_x D \ . \tag{3.13}$$

Finally, reordering the second equation from Eq. (3.4) and substituting $h_{b.r}$ into Eq. (3.13) results in

$$Z_c = \frac{f_x + f_y}{2}\frac{D}{D_{pix}} \ , \tag{3.14}$$

which relates the unknown Z_c coordinate to the known camera focal lengths, f_x and f_y (obtained from the camera calibration procedure described in Section 3.1), the known ball diameter in the camera's frame, D, and the ball diameter in the image plane, D_{pix}, calculated in Section 3.2.2. Thus, Eq. (3.14) allows Z_c to be computed.

Knowing Z_c and the camera matrix, Eq. (3.7) can now be solved. Therefore, the remaining ball center coordinates in the camera's frame, X_c and Y_c, can be determined.

3.4 Camera-LIDAR Calibration Methods

Two different calibration methods are used for camera-LIDAR calibration: the 3D rigid body transformation and the extrinsic camera calibration method.

3.4.1 3D Rigid Body Transformation

The 3D rigid body transformation method implemented by Pereira in [1] is described in Section 1.5.2. The method requires a target point cloud and a source point cloud; usually the target point cloud is the ball centers point cloud obtained by one of the SICK LMS151 sensors, as it is the most accurate, and the source point cloud is the ball centers point cloud in the camera coordinate system (X_c, Y_c, Z_c). The method returns a geometric transformation applied to the source point cloud that minimizes the point-to-point error metric between the transformed source point cloud and the target point cloud.

3.4.2 Extrinsic Camera Calibration

Extrinsic camera calibration can be used as an alternative to the 3D rigid body transformation for cameras. According to the pinhole model, a scene is formed by projecting 3D world points to the image plane. For a generic 3D world point (X_w, Y_w, Z_w), Eq. (3.15) gives its 2D projection in the image plane. The goal of an extrinsic calibration is to find the camera pose, matrix T.

$$\begin{aligned}\begin{bmatrix} u \\ v \\ w \end{bmatrix} = Z_c \begin{bmatrix} x_{c.pix} \\ y_{c.pix} \\ 1 \end{bmatrix} &= \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{1,2} & r_{2,2} & r_{2,3} & t_2 \\ r_{1,3} & r_{2,3} & r_{3,3} & t_3 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \\ &= KT \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}\end{aligned} \tag{3.15}$$

Similarly to the 3D rigid body transformation method, a 3D target point cloud is used; however, instead of using the ball centers point cloud in the camera coordinate system (X_c, Y_c, Z_c) as the source cloud, the extrinsic camera calibration uses the ball centers coordinates in the image coordinate system $(x_{c.pix}, y_{c.pix})$ to evaluate the extrinsic parameters of the camera.

Pose estimation of a calibrated camera with a set of 3D points and their corresponding 2D image projections is a Perspective-n-Point problem. The problem is solved resorting to an iterative method based on Levenberg-Marquardt optimization algorithm [4], [5], finding a pose that minimizes reprojection error. The implementation used is the one available in the *solvePnP* function of OpenCV. Although, given the frequent existence of outliers in the source and target point clouds, more stable results were obtained using the available RANSAC version of the same algorithm (*solvePnPRansac*).

Chapter 4

Graphical Interface for the Calibration Package

The second main objective of this work is the development of a GUI for the calibration package. The goal is to develop an easy-to-use GUI, easily expandable to more sensors and capable of calibrating any number of supported sensors.

This chapter will describe the GUI main elements, the main changes to the calibration package architecture, how nodes are launched and stopped through the GUI, and how the calibration results can be visualized and stored. Finally, the calibration package and GUI are expanded to support the Microsoft Kinect 3D-depth sensor, demonstrating the process of adding sensors to the package.

Qt version 4.8.6, presented in Section 2.3.3, is the chosen framework to develop the GUI. The reason for using Qt version 4.8.6 specifically, instead of more recent versions, is compatibility with the *librviz* library which is based of version 4.8.6.

Additionally, Appendix A contains instructions to install and use the GUI developed in this chapter.

4.1 Main GUI Elements

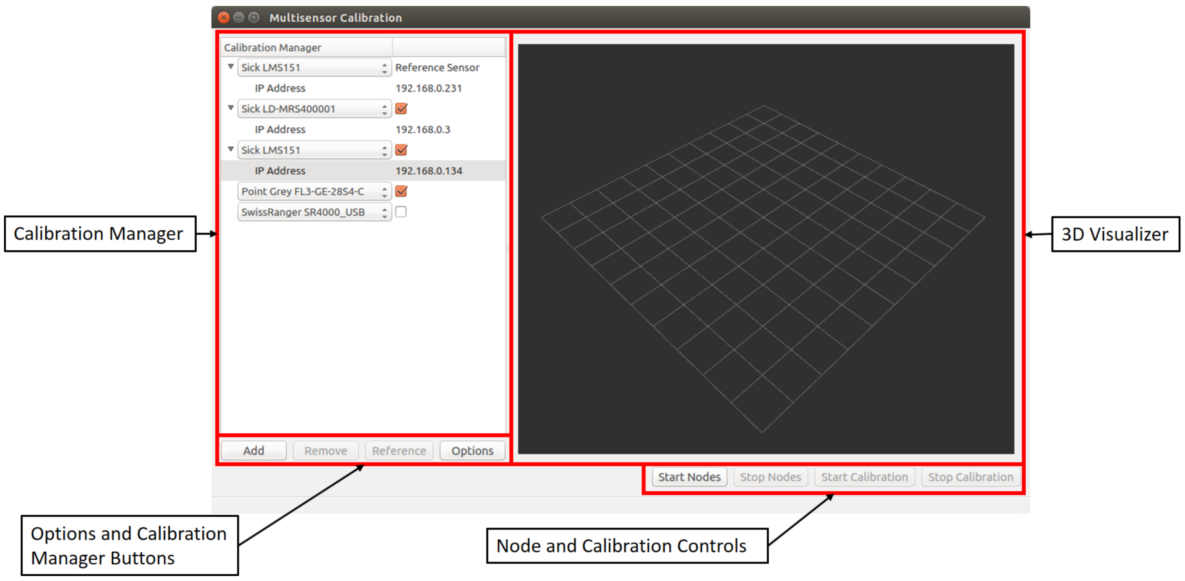
The current section presents the GUI main windows, their main areas and details the functionality behind their contents. There are two main windows, which will be detailed in the following sections, the "Multisensory Calibration" window will be presented first, followed by the "Options" window.

4.1.1 Multisensory Calibration Window

The GUI main window is the "Multisensory Calibration" window, presented in Figure 4.1. It is constituted by four main areas: the "Calibration Manager", the "Options and calibration manager buttons", the "Visualization" and the "Node and calibration controls".

Calibration Manager

The "Calibration Manager" area is a *QTreeWidget* which is an item-based interface, similar to a list. Its purpose is to list the sensors added by the user and their sensor-specific options, like the sensor's IP address. The first item/sensor on the list is considered to be the

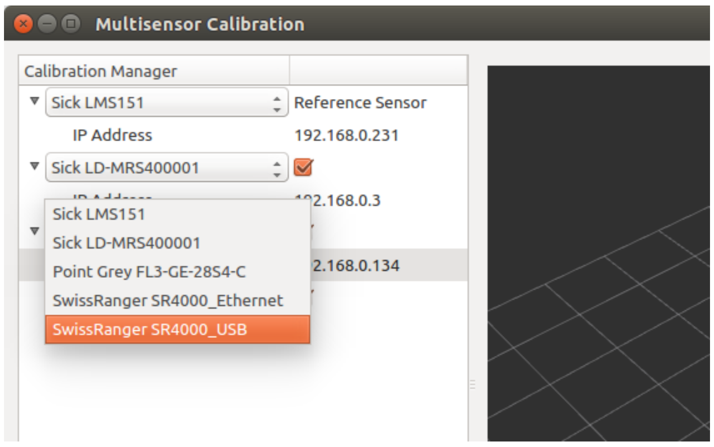


Calibration Manager
Sick LMS151 Reference Sensor
IP Address 192.168.0.231
Sick LD-MRS400001 ✓
IP Address 192.168.0.3
Sick LMS151 ✓
IP Address 192.168.0.134
Point Grey FL3-GE-28S4-C ✓
SwissRanger SR4000_USB

Figure 4.1: The calibration package GUI "Multisensory Calibration" window.

reference sensor, as the text in the second column indicates; the calibration will compute the geometric transformation of the other sensors in the list relative to this sensor. In the second column there is also a check box (*QCheckBox*): when checked the sensor will be calibrated, when unchecked the sensor will not be calibrated. Note that the reference sensor does not have a check box since it is always needed for calibration.

Sensor selection is done using a *QComboBox* that contains every supported sensor by the calibration package (see Figure 4.2). Sensor-specific options are also changed accordingly.



Calibration Manager
Sick LMS151 Reference Sensor
IP Address 192.168.0.231
Sick LD-MRS400001 ✓
192.168.0.3
2.168.0.134

Figure 4.2: Sensor selection using a *QComboBox*.

Options and Calibration Manager Buttons

This area contains four buttons:

- Add – adds a sensor/item to the end of the "Calibration Manager" *QTreeWidget*;

- Remove – removes the selected sensor/item from the “Calibration Manager” *QTreeWidget*;
- Reference – makes the selected sensor the reference sensor, moving it to the top of the “Calibration Manager” *QTreeWidget*;
- Options – launches the “Options” window presented on the following section.

As can be observed in Figure 4.1, when the user selects a sensor-specific option, the “Remove” and “Reference” buttons are disabled since that is not a valid sensor selection to remove, or to make the reference sensor.

Visualization

The “Visualization” area is a 3D visualizer based on Rviz, created using the *librviz* library, see Section 2.3.4 for more details. The visualizer automatically subscribes to a topic published by the calibration node, allowing the user to monitor the calibration process within the application.

Node and Calibration Controls

This area allows the user to launch and stop nodes as well as start and stop the calibration procedure. It contains four buttons:

- Start Nodes – launches the required nodes for each sensor currently in the “Calibration Manager” *QTreeWidget*, provided their respective checkbox is checked;
- Stop Nodes – terminates previously launched nodes;
- Start Calibration – starts the calibration node to calibrate the previously launched sensors;
- Stop Calibration – stops the calibration procedure and node.

To help guide the user through the calibration process, and to avoid undefined behavior, the buttons are enabled or disabled depending on the current state of the application, as detailed in Table 4.1.

It is important to clarify what happens when the “Stop Nodes” button is pressed by the user, according to Table 4.1, every button in this area is disabled, however, that is only the case while the application waits for the nodes to terminate. Therefore, a label containing appropriate text notifies the user that the application is waiting for the nodes to terminate, as shown in Figure 4.3. After every launched node terminates, the buttons in this area revert to their “No sensors launched” state.

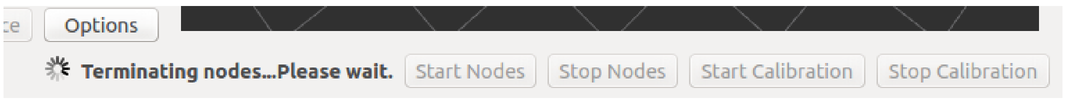

Terminating nodes...Please wait.

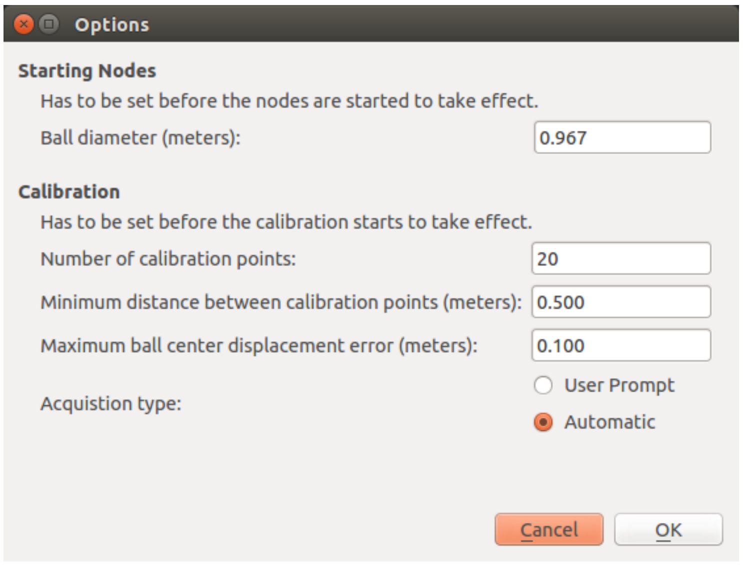
Figure 4.3: Message shown when the launched nodes are being terminated.

Table 4.1: Button state after user actions. 1 – Enabled; 0 – Disabled.

	Button state after action			
Action	**Start Nodes**	**Stop Nodes**	**Start Calibration**	**Stop Calibration**
No sensors launched	1	0	0	0
Start Nodes	0	1	1	0
Stop Nodes	0	0	0	0
Start Calibration	0	1	0	1
Stop Calibration	0	1	1	0

4.1.2 Options Window

The "Options" window, shown in Figure 4.4, contains options that are specific to the calibration process. As Figure 4.4 indicates, there are two sets of options: "Starting Nodes" and "Calibration".


Options

Starting Nodes
Has to be set before the nodes are started to take effect.
Ball diameter (meters): 0.967

Calibration
Has to be set before the calibration starts to take effect.
Number of calibration points: 20
Minimum distance between calibration points (meters): 0.500
Maximum ball center displacement error (meters): 0.100
Acquistion type: User Prompt [✓ Automatic]

Figure 4.4: The calibration package GUI "Options" window.

Starting Nodes Options

The "Starting Nodes" options consists of every option whose information is required when the sensors nodes are launched. To take effect, these options have to be set before the sensor nodes are launched.

Currently, the only option that fits this group, is the calibration ball diameter, in meters. By default, it's set at 0.967 meters, which is the diameter of the calibration ball used in this work, as will be detailed in Section 5.1.

Calibration Options

The second set of options are the "Calibration" options, whose information is required when the calibration process is started. Therefore, these options have to be set before starting the calibration process. This set of options consists of:

- Number of calibration points – the number of ball centers that every sensor has to detect to estimate the transformation between the reference sensor and the other sensors. Set, by default, to 20 points;
- Minimum distance between calibration points – minimum euclidean distance, in meters, between a newly acquired ball center and the previously accepted ball centers. If the newly acquired point is below the set distance, it will be discarded. This parameter ensures that the calibration points are distributed in space. Set, by default, to 0.5 meters;
- Maximum ball center displacement error – for each sensor the ball center displacement is calculated between a newly acquired position and its previously accepted position; in perfect conditions the displacement would be the same for every sensor. However, given ball detection errors, the intrinsic error of the sensors and the asynchronous nature of the messages received by the sensors, that is not verified. Therefore, this option sets a maximum difference, or error, between those displacements. If the difference is above the set threshold, the new ball center is invalid and discarded. Set to 0.1 meters, by default;
- Acquisition type – allows the user to choose between "Automatic" or "User Prompt" modes. In "User Prompt" mode, the user has the ability to choose when a calibration point is acquired through a message box (*QMessageBox*), shown in Figure 4.5. In automatic mode, the points are constantly being acquired without any user input. By default, this option is set to "Automatic" mode.

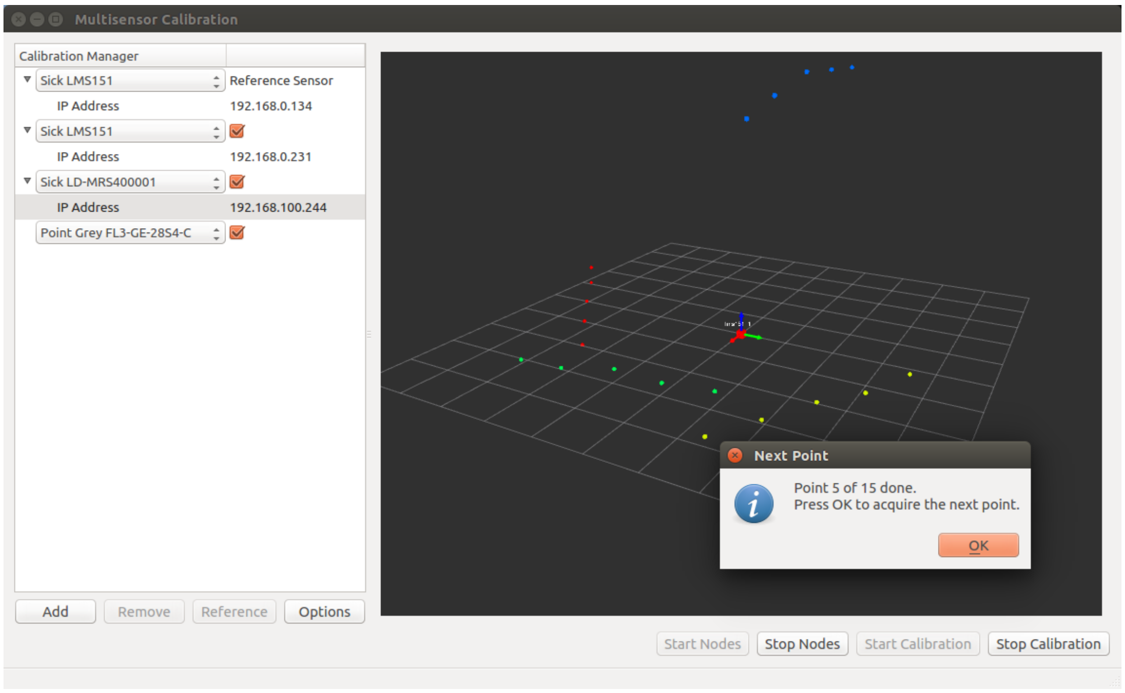
4.2 ROS Architecture

Before developing the GUI, the calibration package was built with a focus on ATLASCAR 1 sensors, detailed in Section 2.2. Calibrating a different set of sensors required changing the package source code, and even modifying the package *CMakeLists.txt* to add more executables if the user required more sensors. Thus, one of main goals for this application is allowing the user to calibrate any number of sensors, provided that the sensors are supported by the calibration package. Achieving this goal required many changes to the package architecture, though its principles are same.

To demonstrate the architecture changes made to the package, *rqt_graph* is used to generate the ROS computation graph[1] of the calibration package, before (Figure 4.6a) and after (Figure 4.6b) the GUI development. The computation graphs were generated during the calibration of the sensors on ATLASCAR 1 – two SICKs LMS151, one SICK LD-MRS400001 and one Point Grey FL3-GE-28S4-C camera.

The main difference between Figures 4.6a and 4.6b is the naming convention. Essentially, the naming convention allows the *calibration_gui* node to automatically subscribe to the necessary topics by only knowing their node name, which is automatically generated by the GUI. The generated node name is simply a pre-defined string that identifies the sensor, and an integer number that is used to differentiate between multiple of the same sensor.

[1]More details about *rqt_graph* and ROS computation graphs can be found at [48] and [49], respectively


Calibration Manager

Sick LMS151	Reference Sensor
IP Address	192.168.0.134
Sick LMS151	✓
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.100.244
Point Grey FL3-GE-28S4-C	✓

Next Point

Point 5 of 15 done.
Press OK to acquire the next point.

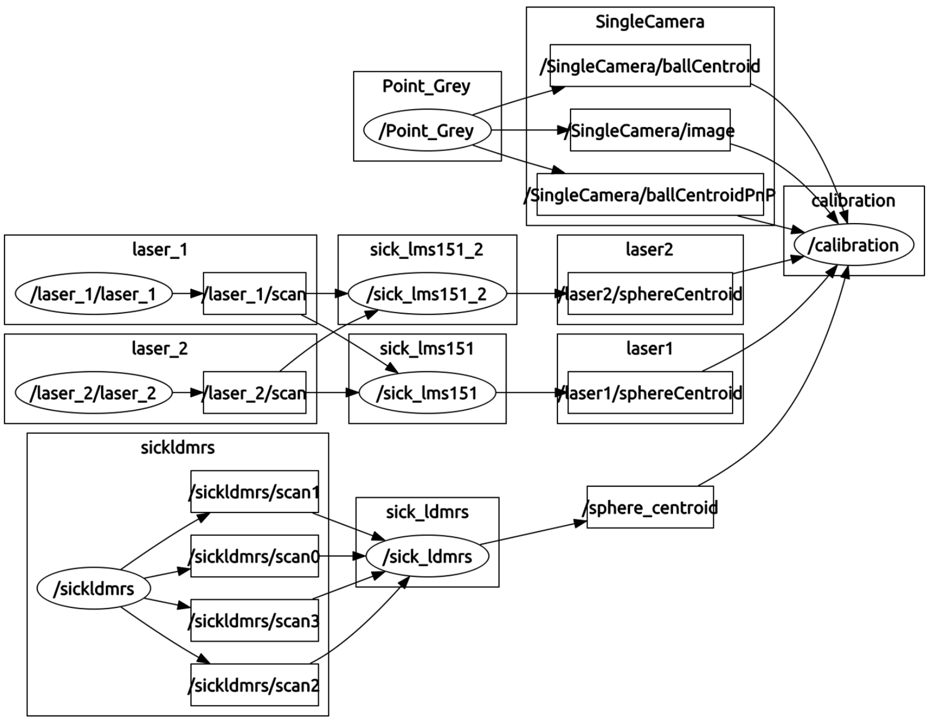
Figure 4.5: Message Box displayed in "User Prompt" mode.

The second main difference is the clear structure that all sensors follow, as described in Figure 4.7 and applied in Figure 4.6b.

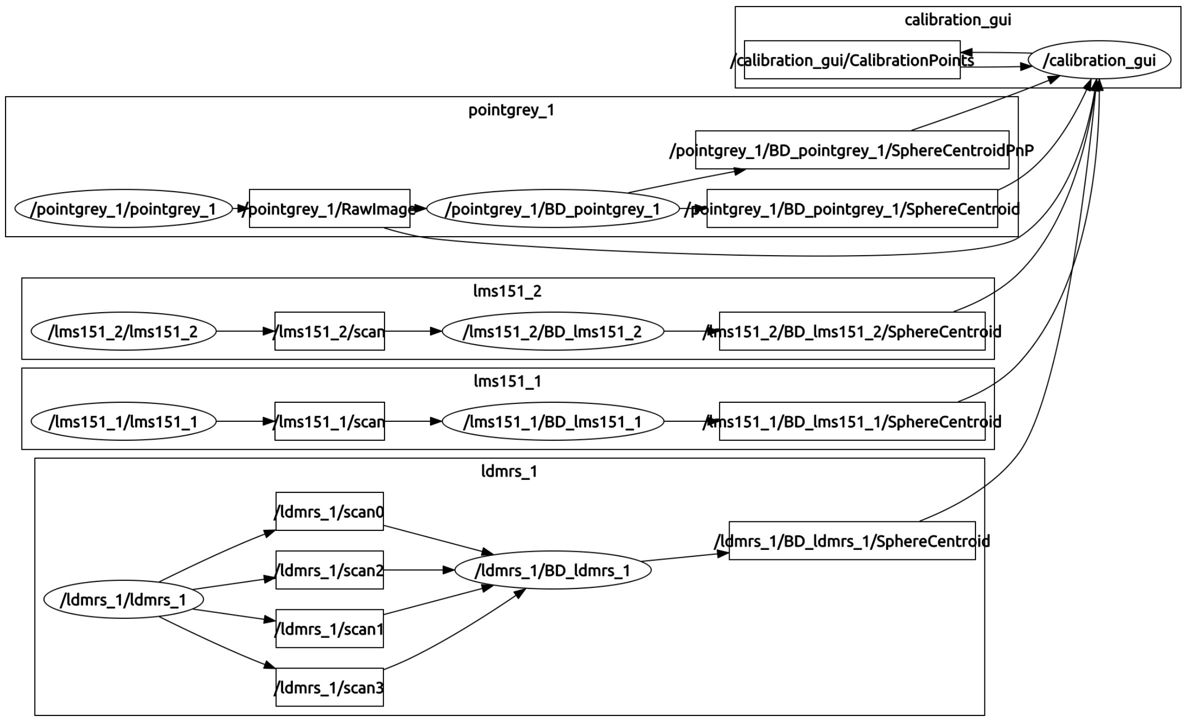
Most of the naming and structure conventions implemented are merely guidelines that should be followed when a new sensor is added to the calibration package. There are, however, a few rules regarding topic names:

- for cameras, the following topic names are required:
  - the acquired/raw image has to be published to a topic named "/%s_%d/RawImage";
  - the detected ball center in the camera's frame has to be published to a topic named "/%s_%d/BD_%s_%d/SphereCentroid";
  - the detected ball center in the image's frame has to be published to a topic named "/%s_%d/BD_%s_%d/SphereCentroidPnP";
- for other sensors, the detected ball center in the sensor's frame has to be published to a topic named "/%s_%d/BD_%s_%d/SphereCentroid";

Where %s represents the pre-defined string, which identifies the sensor, and %d is an integer number, which is used to differentiate between multiple of the same sensor. As an example, for a SICK LMS151, %s is "lms151" and %d is "1", thus detected ball centers have to published to the "/lms151_1/BD_lms151_1/SphereCentroid" – see Figure 4.6b for more examples with different sensors.



(a)



(b)

Figure 4.6: Calibration package computation graph before (a), and after (b) the GUI development. Ellipses represent nodes, and rectangles represent topics.

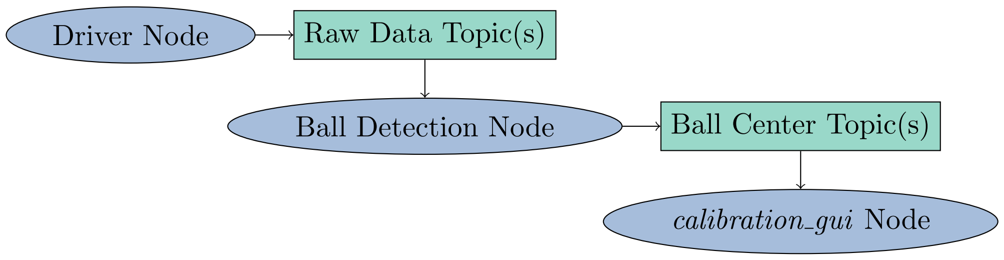


Figure 4.7: Generic ROS architecture implementation for each sensor. Ellipses represent nodes, and rectangles represent topics.

4.3 Launching and Terminating Nodes

This section details how the required nodes from each sensor in the "Calibration Manager" list are launched, through the application. Then, it describes how the launched nodes are monitored and terminated when requested by the user.

4.3.1 Launching Nodes

Launching a node is the same as launching any other process, external to the application through the command line. For this exact purpose, Qt implements the *QProcess* class, which allows the application to start and communicate with external processes.

To further simplify the task, roslaunch is used to launch the nodes. Therefore, a *QProcess* starts a sensor-specific roslaunch process that, in turn, starts the necessary sensor-specific nodes. Allowing the application to take advantage of the roslaunch package tools while keeping the number of *QProcesses* to a minimum.

Generic Structure of Roslaunch Files

Even though each sensor supported by the calibration package has a specific roslaunch configuration file, their structure is essentially the same and is exemplified Listing 4.1 – for details about the tags and attributes used in the example see Section 2.3.4.

When the user presses the "Start Nodes" button, the application will evaluate: the sensors and their sensor-specific options, in the "Calibration Manager" area; and, the starting nodes options in the "Options" window. Then, a *QProcess* is created which runs the appropriate command according to the sensor, sensor-specific options, and starting nodes options. As an example, if the user selects two SICK LMS151 sensors, with IP Addresses "192.168.0.134" and "192.168.0.231", and a ball diameter of 0.967 meters; two *QProcess* will be created and run the following commands:

1. roslaunch calibration_gui Sick LMS151.launch ball_diameter:=0.967 host:=192.168.0.134 node_name:=lms151_1;

2. roslaunch calibration_gui Sick LMS151.launch ball_diameter:=0.967 host:=192.168.0.231 node_name:=lms151_2.

Where "calibration_gui" is the calibration package name, "Sick LMS151.launch" is the launch file for the SICK LMS151 sensor, and "ball_diameter", "host" and "node_name" are roslaunch arguments.

Listing 4.1: Generic structure for roslaunch files to launch sensor-specific nodes.

```
<?xml version="1.0"?>
<launch>
  <!--Sensor-specific arguments below/-->
  <arg name="host"/>
  <arg name="node_name"/>
  <arg name="ball_diameter"/>

  <group ns="$(arg node_name)">
    <!--Launch sensor driver node/-->
    <node name="$(arg node_name)" pkg="sensor_driver_package" type="
       ↪ driver_executable" required="true" output="screen">
      <!--Driver node parameters are set below/-->
      <param name="host" value="$(arg host)"/>
    </node>

    <!--Launch sensor-specific ball detection node/-->
    <node name="BD_$(arg node_name)" pkg="ball_detection_package" type
       ↪ ="ball_detection_executable" required="true" output="screen">
      <!--Ball detection node parameters are set below/-->
      <param name="ballDiameter" type="double" value="$(arg
         ↪ ball_diameter)"/>
    </node>
  </group>
</launch>
```

4.3.2 Terminating Nodes

The *QProcess* class implements the *terminate*() slot to terminate the process. On Linux, the operating system used, *terminate*() sends the SIGTERM signal to the process, giving it the chance to exit cleanly.

Pressing the “Stop Nodes” button simply calls the *terminate*() slot in every *QProcess*, the application then enters the state shown in Figure 4.3 until every *QProcess* emits the *finished*() signal, which means the process is dead.

4.4 Saving and Visualizing Calibration Results

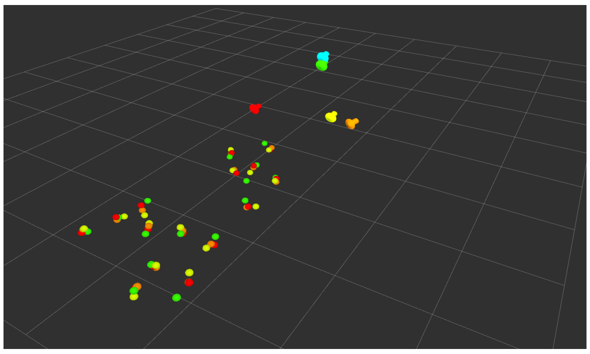
Regarding the results of the calibration process, the application relies on the built-in 3D Visualizer to display them, in real time, and saves all the relevant information in appropriate files.

4.4.1 Built-in 3D Visualizer

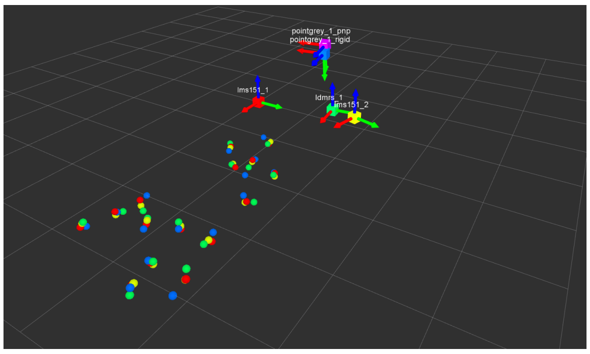
As introduced in Section 4.1.1, the built-in visualizer is based on the Rviz visualizer. It allows the user to monitor the calibration process and see its results.

With the development of the GUI the visualization aspect of the calibration package has also been improved. More specifically, the sensors pose shown after the calibration process consisted of an arrow pointing in the same direction as the cameras’ optical axis (Z-axis) or, for laser scanners, along their zero degree ray line (X-axis). The pose shown did not provide enough information about the sensor's coordinate system relative to the reference sensor. Thus, the X-Y-Z axes are now displayed for each sensor pose, as well as, a square at the axes center, so the pose can be color-matched to the displayed point clouds.

Figure 4.8 provides a comparison between the results visualization before and after the GUI development, for the same sensor setup – two SICKs LMS151, one SICK LD-MRS400001 and one Point Grey FL3-GE-28S4-C camera.



(a)



(b) X-Y-Z axes are represented by red-green-blue arrows, respectively.

Figure 4.8: Calibration package results visualization before (a), and after (b) the GUI development.

4.4.2 Saving Calibration Results

Results from the calibration procedure are also saved in files automatically. A folder in the calibration package directory is created with the following name: "day-month-year_hour-minutes-seconds". The time-dependent format guarantees that every calibration is saved and does not overwrite previous tests.

The estimated geometric transformations, in homogeneous format, between the reference sensor and each non-reference sensor are saved in *.txt* files; one for each estimated geometric transformation. The ball centers point clouds used to estimate those transformations are saved in separate *.pcd* files. If the calibration process includes cameras, the images where a valid ball center is acquired are saved, as well as, an image containing the projection of the reference sensor point cloud to their respective image plane.

4.5 Expanding the Calibration Package to Microsoft Kinect

The current section describes the process of expanding the calibration package to the Microsoft Kinect 3D-depth sensor. The goal is to offer a generic approach to the process by using a practical example.

The first step to add a sensor to the calibration package, provided its drivers are already available, is developing the ball detection and ball center computation node for the sensor. Next, a roslaunch file should be built according to the guidelines presented in Sections 4.2

and 4.3.1. Finally, the sensor has to be added to the list of supported sensors within the GUI. The following sections apply this process to the Microsoft Kinect 3D-depth sensor.

4.5.1 Ball Detection and Ball Center Computation

The data generated by Kinect's 3D-depth sensor is very similar to the data generated by the SwissRanger SR4000. Hence, the PCL algorithm used with the SR4000, discussed in Section 1.5.2, can be applied here, to detect the ball and compute its center. The only difference in methodology is the application of a 3D voxel grid filter to downsample the point cloud generated by Kinect. Since each point cloud generated by its sensor consists of 307200 points, a downsampling filter is necessary to increase performance – for comparison, the SR4000 generates 25344 points per point cloud. Figure 4.9 shows a point cloud, with RGB registration[2], from Kinect and the detected ball.

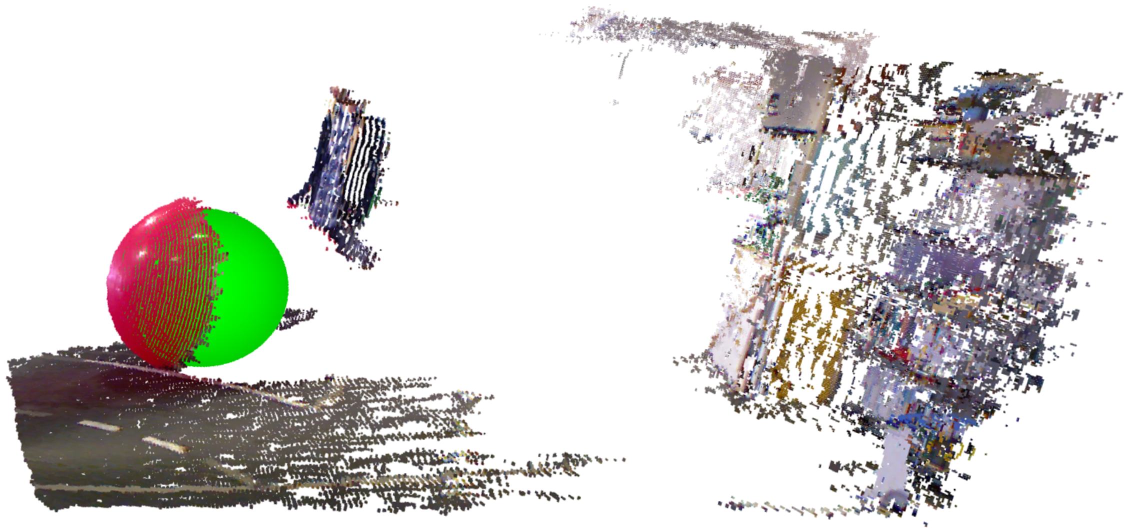


Figure 4.9: Ball detection (green sphere) in the point cloud acquired by the Microsoft Kinect 3D-depth sensor with RGB registration.

4.5.2 Roslaunch File

The roslaunch configuration file developed for the Kinect 3D-depth sensor is presented in Listing 4.2. Comparing the roslaunch file developed for the Kinect 3D-depth sensor with the generic roslaunch file structure presented in Section 4.3.1, it is possible to note that the only major difference between the two is the replacement of the driver node by the inclusion of the OpenNI roslaunch file. OpenNI is not only a driver node for Kinect but also loads multiple nodelets to convert raw data streams to depth images, disparity images and point clouds. To use OpenNI's factory-calibrated depth to RGB registration, the "depth_registration" argument is set to true.

[2]RGB registration is done by OpenNI and is only used for visualization purposes; RGB information is not used to detect the ball.

Listing 4.2: Roslaunch file to launch the required Microsoft Kinect 3D-depth sensor nodes.

```
<?xml version="1.0"?>
<launch>
    <arg name="node_name"/>
    <arg name="ball_diameter"/>

    <group ns="$(arg node_name)">
        <include file="$(find openni_launch)/launch/openni.launch">
            <arg name="depth_registration" value="true"/>
        </include>

        <node name="BD_$(arg node_name)" pkg="calibration_gui" type="
           ↪ kinect" required="true" output="screen">
            <param name="ballDiameter" type="double" value="$(arg
               ↪ ball_diameter)"/>
        </node>
    </group>
</launch>
```

4.5.3 Adding Microsoft Kinect 3D-depth sensor to the GUI

Adding support for a new sensor to the GUI is a two-step process, presented here for the Microsoft Kinect 3D-depth sensor. The first step is adding the sensor to the list of supported sensors, which is done by adding the following code to the *SupportedSensors* class[3] constructor:

```
this->addSupportedSensors("Microsoft Kinect 3D Sensor", "kinect_3d",
    ↪ false);
```

where *addSupportedSensors* is a private method of the *SupportedSensors* class. "Microsoft Kinect 3D Sensor" is the roslaunch file name, and the entry name on the sensor selection combo box, as shown in Figure 4.10. "kinect_3d" is the string portion of the nodes names that identify the sensor, as described in Section 4.2. Finally, "false" indicates whether the sensor being added is a camera or not; since the Kinect 3D-depth sensor is not a camera, the argument is "false".

The second, and final step, is adding sensor-specific options to the calibration manager *QTreeWidget*. These options are added through the *addTreeChilds* method from the *SupportedSensors* class. However, since Kinect uses an USB interface there are no sensor-specific options.

Kinect 3D-depth sensor nodes are launched as described in Section 4.3.1. As an example, to launch a single Microsoft Kinect 3D-depth sensor, with a ball diameter of 0.967 meters, a *QProcess* is created and runs the following command: "roslaunch calibration_gui Microsoft Kinect 3D Sensor.launch ball_diameter:=0.967 node_name:=kinect_3d_1". Where "Microsoft Kinect 3D Sensor.launch" is the launch file presented in the previous section, and "ball_diameter" and "node_name" are roslaunch arguments.

[3]The class is found in the *gui_supportedsensors.cpp* source file.

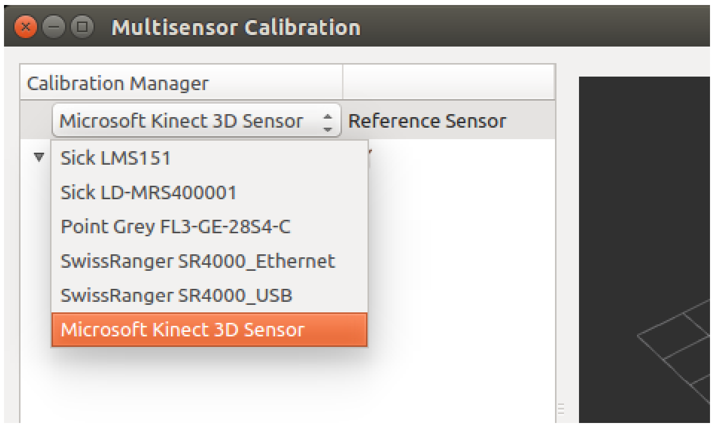


Figure 4.10: Supported sensors list with the new entry for the Microsoft Kinect 3D-depth sensor.

Chapter 5

Results

This chapter presents the results from extrinsic calibration tests to evaluate the performance of the calibration package, especially for both the camera–LIDAR extrinsic calibration methods described in Chapter 3. In order to obtain accurate extrinsic calibrations, the calibration ball diameter is first measured using the SwissRanger SR4000, Microsoft Kinect 3D-depth sensor and manually. Finally, sensor data fusion is carried out using the estimated transformations by the calibration process. Additionally, the GUI developed in Chapter 4 is utilized to perform the tests presented in the following sections.

5.1 Measuring the Ball Diameter

Every ball detection method implemented in this package relies on knowledge about the ball diameter to compute its center coordinates in the sensors local frame. Therefore, an exact measurement of the diameter is essential to obtain accurate results in the following tests.

Since the calibration ball used in this work (Figure 3.1b) is inflatable, it is not easy to guarantee that its diameter is exactly 95 centimeters as announced by the vendor. Thus, after inflating the ball, three methods were used to measure its diameter:

1. using the SwissRanger SR4000 ball detection algorithm;
2. similarly to the previous method, Microsoft Kinect ball detection algorithm is also used;
3. the circumference around the ball's equator is manually measured using a flexible tape. Then, the measured circumference is divided by π to obtain the diameter.

For the SR4000 and Kinect, 750 ball diameter measurements were taken with the ball static. The setup is shown in Figure 5.1 and the results in Figures 5.2a and 5.2b, for the SR4000 and Kinect, respectively.

The mean ball diameter computed by the SR4000 is 0.964 m, with a standard deviation of 0.03 m. Regarding Kinect, the mean diameter obtained is 0.965 m, with a standard deviation of 0.02 m. Although, both measurements agree on the ball diameter, the sensors use the same algorithm to compute the ball diameter; to guarantee that the results are reliable, the circumference around the ball's equator was manually measured using a flexible tape. The circumference measured 3.05 m, equivalent to a diameter of 0.971 m. Finally, the ball diameter is taken as the average of the three measurements is computed, yielding a diameter of 0.967 m.

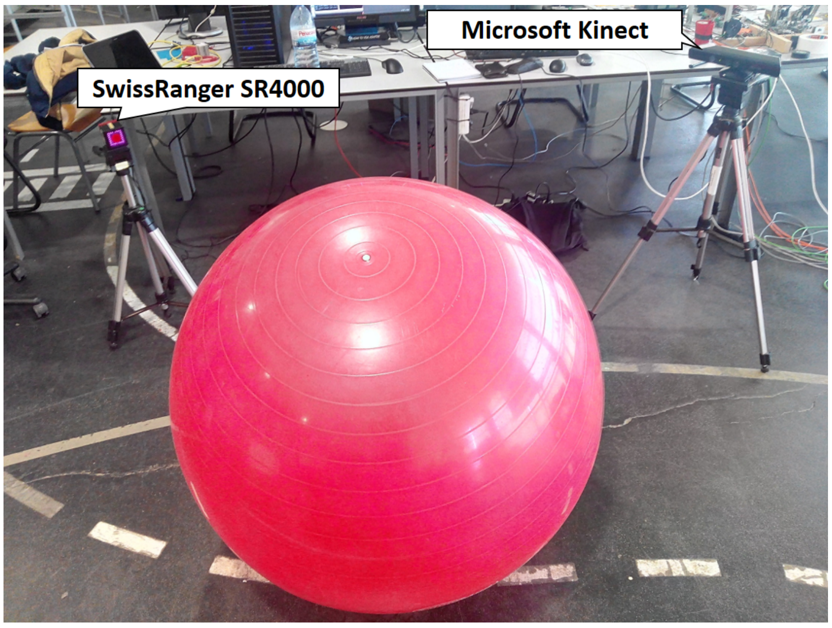


Figure 5.1: Setup used to measure the ball diameter with the SwissRanger SR4000 and Microsoft Kinect.

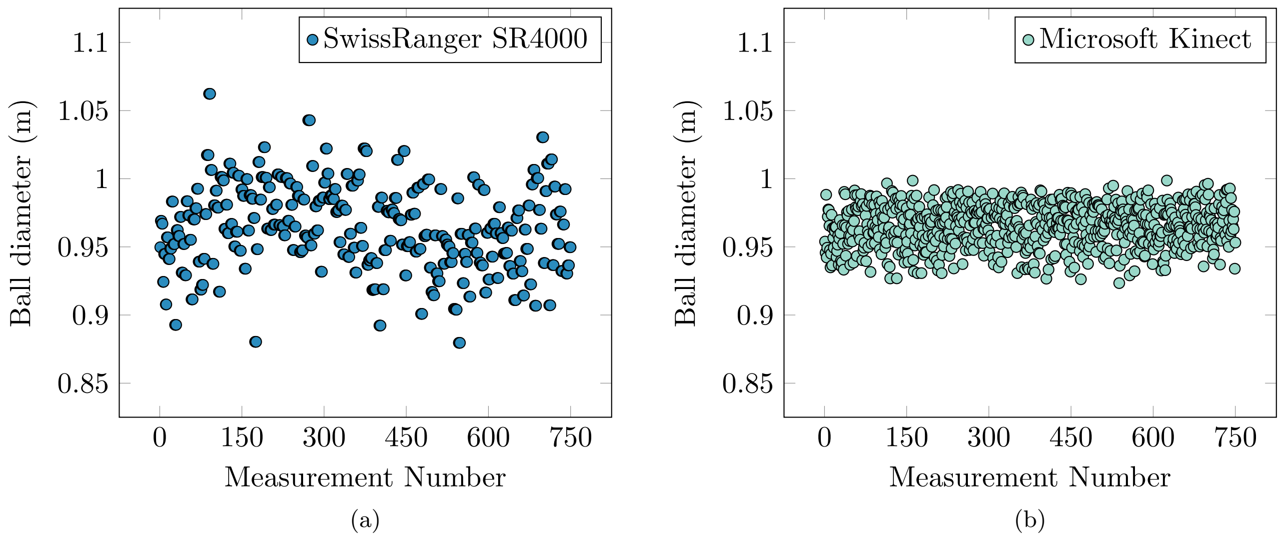


Figure 5.2: Calibration ball diameter measurements using: (a) SwissRanger SR4000; (b) Microsoft Kinect.

5.2 Extrinsic Calibration Evaluation Using ATLASCAR 1

In order to evaluate the performance of the calibration package, two extrinsic calibrations are performed using ATLASCAR 1. The tests presented in this section follow the same principles and procedures used by Pereira [1].

In the first test, the ball is positioned along a grid pattern. The goal is to establish a ground-truth for the ball position and its displacement between grid points. Evaluating not only the estimated geometric transformations, but also the accuracy of the ball center detection methods – with a special focus on the ball detection and pose estimation methodologies for cameras presented in Chapter 3.

The second test evaluates the estimated geometric transformations in a real world application of the calibration package. Thus, the calibration ball is put in motion and the ball center points are automatically acquired.

Both tests are done with the same setup and conditions. The sensor setup is shown in Figure 5.3, and includes the following sensors: two SICKs LMS151, a SICK LD-MRS400001 and a Point Grey FL3-GE-28S4-C camera (as first detailed in Section 2.2).

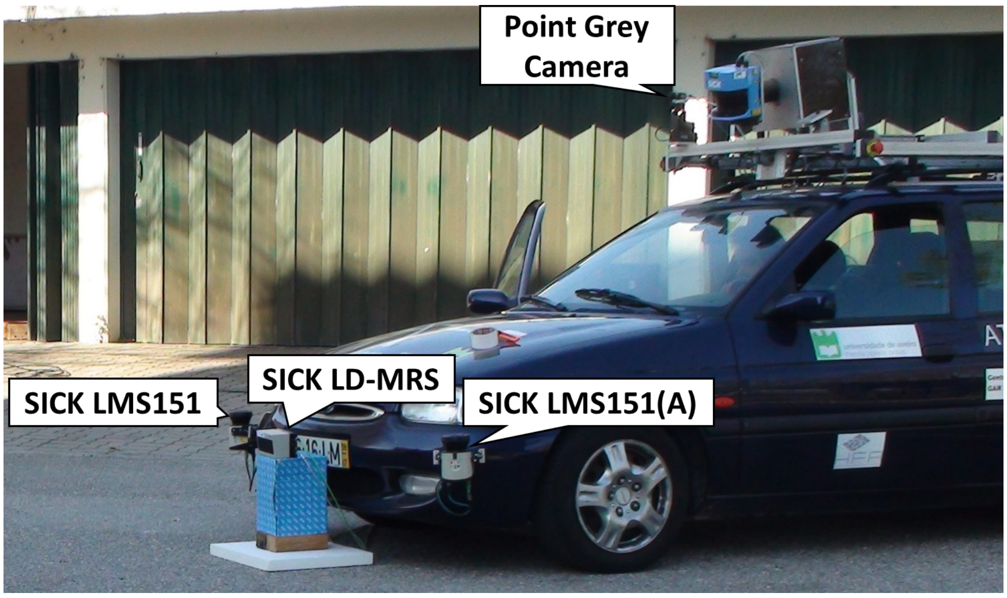


Figure 5.3: ATLASCAR 1 setup used in the calibrations tests.

The methodology employed in both tests can be described as follows:

1. A reference sensor is chosen. Due to the higher accuracy of the Sick LMS151, verified by Pereira [1], one of the two was chosen, and is from now on named SICK LMS151(A);

2. Point cloud acquisition and sensor calibration using the GUI developed in Chapter 4. Figure 5.4 shows the sensors added to the "Calibration Manager";

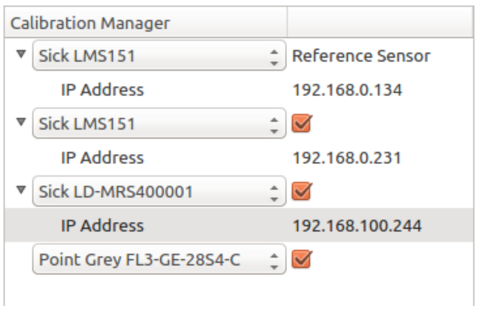
3. Apply the estimated transformations to their correspondent uncalibrated sensor frames;

4. Results evaluation:

(a) for transformations calculated with the 3D rigid body transformation method, discussed in Sections 1.5.2 and 3.4.1, the mean euclidean distance, and respective standard deviation, between corresponding points of the source point cloud and target point cloud (acquired with SICK LMS151(A)), is computed and used to evaluate the resulting transformations;

(b) for the SICK LMS151(A)–camera transformation computed by the extrinsic camera calibration method, described in Section 3.4.2, the reprojection error is used to evaluate its results;

(c) qualitative evaluation of the sensor position and orientation after the calibration.



Calibration Manager	
Sick LMS151	Reference Sensor
IP Address	192.168.0.134
Sick LMS151	✓
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.100.244
Point Grey FL3-GE-28S4-C	✓

Figure 5.4: Sensors added to the "Calibration Manager" for calibration with ATLASCAR 1.

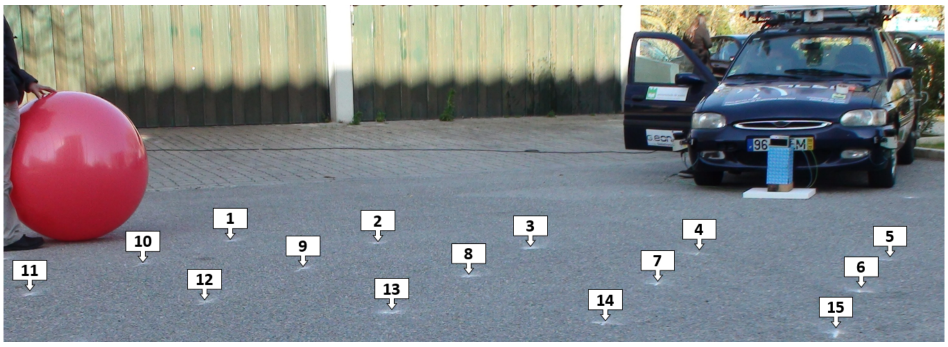
5.2.1 Grid Pattern Calibration

To perform the calibration with a grid pattern, 15 markers were hand-marked on the ground, using a measuring tape, in a 3×5 grid arrangement (see Figure 5.5a). The ball was then placed on top of each marker aligned by its center, and a ball center point is acquired for each sensor, on each grid marker. The GUI parameters used for this calibration test are shown in Figure 5.6; to ensure that acquisition only happens when the ball is placed on top of a grid marker, the "User Prompt" mode is used.

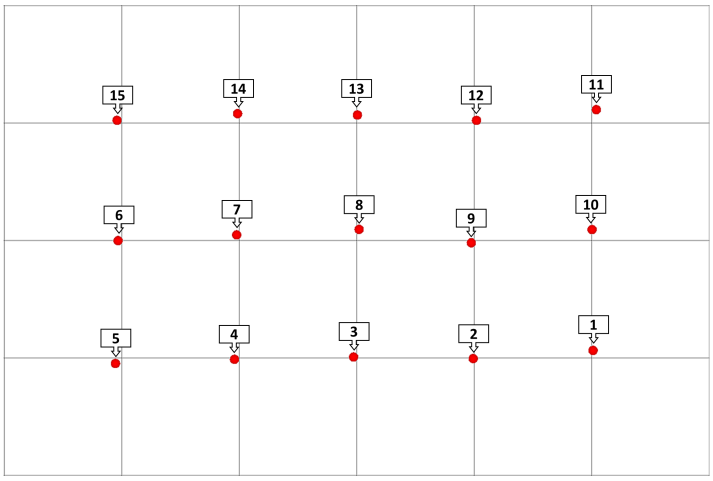
In order to evaluate the grid pattern, Figure 5.5b shows the acquired point cloud for SICK LMS151(A) (reference sensor) overlaid by a virtual grid with 1 m $\times$ 1 m cells. If both the hand-marked grid and SICK LMS151(A) had no associated error, the distance between a ball center on grid point i and a ball center on grid point $i+1$, would be equal to 1 m. To evaluate this condition, the vertical and horizontal Euclidean distance between consecutive ball center positions is calculated and illustrated in Figure 5.5c, which shows that the mean overall distance between ball centers is 1.028 m. A mean error of 28 mm is better than expected, considering the SICK LMS151 has a systematic error of $\pm$30 mm and that the grid was hand-marked using a measuring tape.

The acquired point clouds for each sensor are presented on Figure 5.7a with respect to their own coordinate system. The 3D rigid body transformation method is applied to estimate the geometric transformations between the SICK LMS151(A) and the other sensors, whose point clouds are displayed on Figure 5.7a.

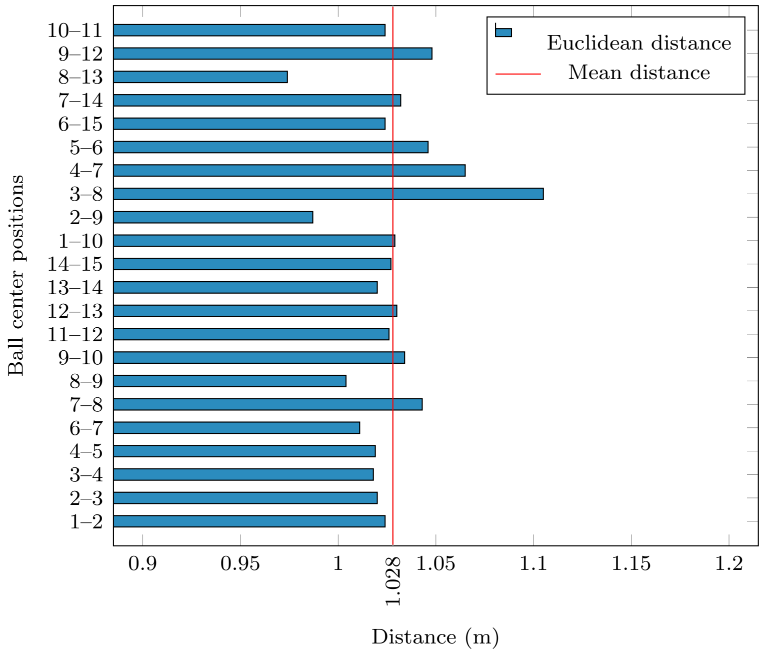
Regarding the camera, there is an alternative method, the extrinsic camera calibration, described in Section 3.4.2, which resorts to a point cloud of ball centers in the image plane (see Figure 5.7b) and the 3D point cloud from SICK LMS151(A).



(a) Hand marked grid points used for calibration.

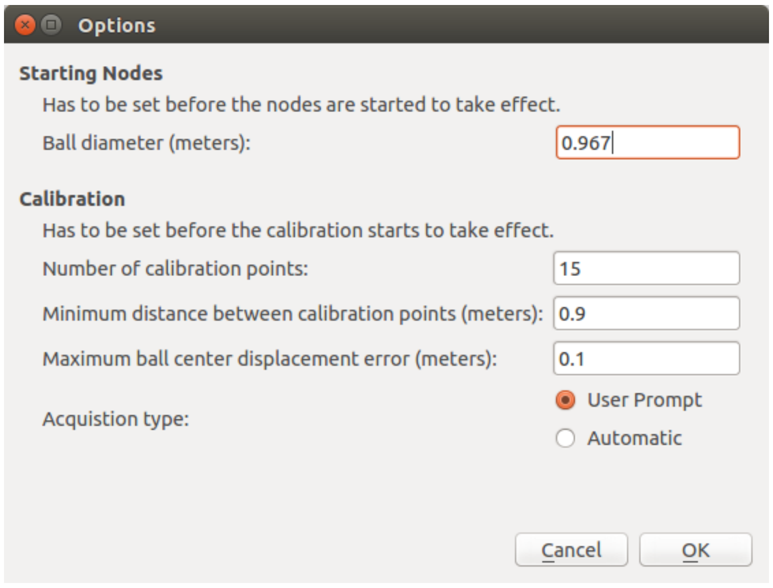


(b) SICK LMS151(A) ball centers point cloud.



(c) Euclidean distance between ball centers acquired by the SICK LMS151(A) sensor.

Figure 5.5: Real world and SICK LMS151(A) grid pattern.



Options

Starting Nodes

Has to be set before the nodes are started to take effect.

Ball diameter (meters): 0.967

Calibration

Has to be set before the calibration starts to take effect.

Number of calibration points: 15

Minimum distance between calibration points (meters): 0.9

Maximum ball center displacement error (meters): 0.1

Acquistion type: [✓ User Prompt] Automatic

Figure 5.6: Grid pattern calibration parameters specified in the “Options” window.

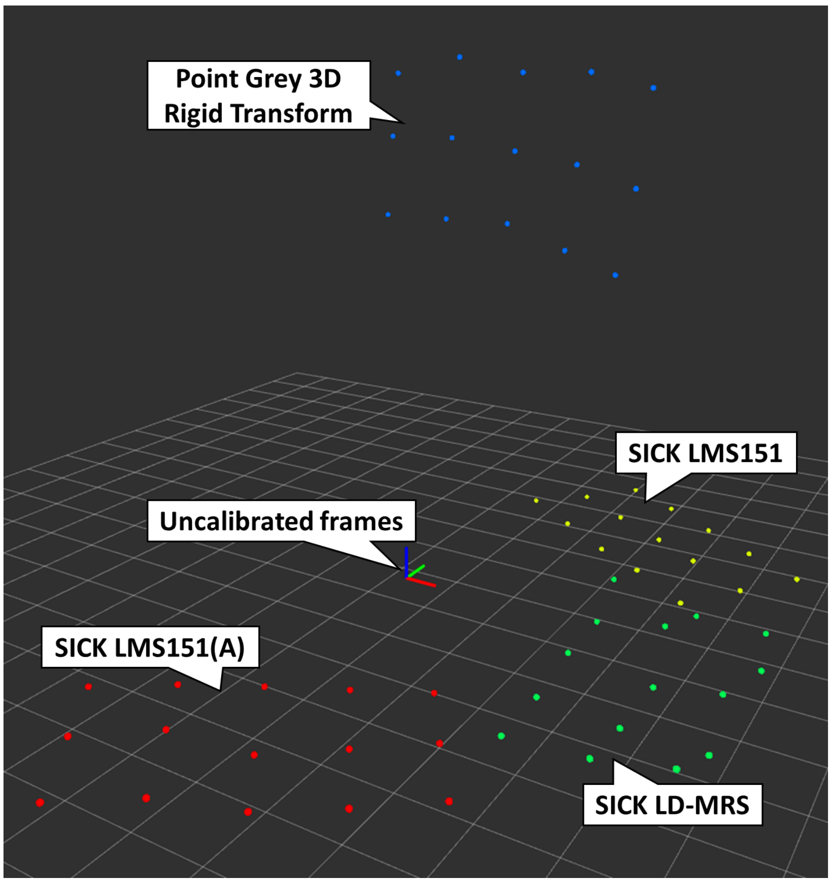
Resulting transformations from both methods are then applied to the point cloud and pose of each sensor. To evaluate the quality of the point cloud fitting obtained through the 3D rigid body transformation method, Table 5.1 is presented. The mean absolute error and standard deviation are based on the euclidean distance between corresponding points of a sensor point cloud and the SICK LMS151(A) point cloud. The SICK LD-MRS400001 performed better than expected, as its absolute mean error (135 mm) is less than half of its systematic error announced by the vendor (± 300 mm). Concerning the Point Grey camera, Table 5.1 shows an error of 157 mm, the highest of the three sensors calibrated in this test. The error indicates that the camera's point cloud does not match well with the point cloud from the reference sensor.

Table 5.1: Absolute mean error and standard deviation for each sensor calibration, using the 3D rigid body transformation method, against Sick LMS151(A).

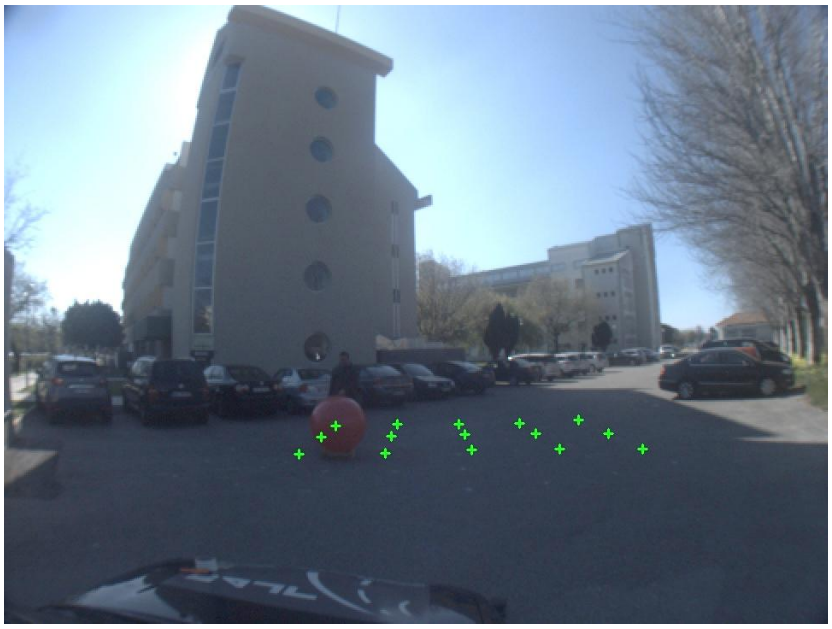
	Absolute mean error [mm]	Standard deviation [mm]
SICK LMS151	50	27
SICK LD-MRS	135	80
Point Grey Camera	157	75

Concerning the extrinsic camera calibration method, the same metric cannot be applied since the camera's point cloud is in the image coordinate system. However, it is possible to project the SICK LMS151(A) point cloud to the image plane, using the geometric transformation estimated by the calibration method and the intrinsic parameters of the camera. A mean reprojection error can then be computed to evaluate the point cloud fitting – in this case, the mean reprojection error is 3.5 pixels. Figure 5.8 shows the initial ball centers detected by the camera and the projected SICK LMS151(A) world points.

To visually assess the lasers and camera pose relative to the car as well as the point cloud



(a) Point clouds to be used with the 3D rigid body transformation method.



(b) Ball centers to be used with the extrinsic camera calibration method.

Figure 5.7: Grid pattern uncalibrated point clouds in their respective coordinate systems.

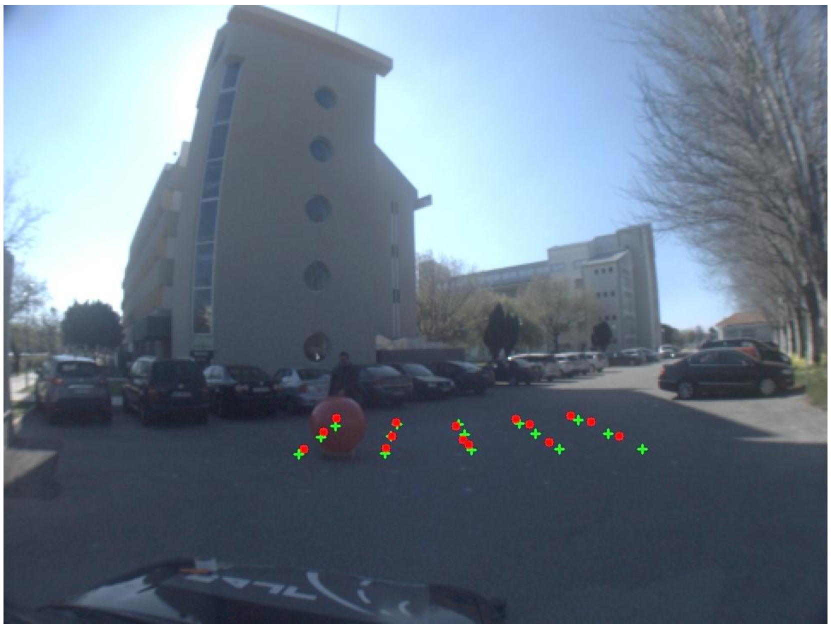


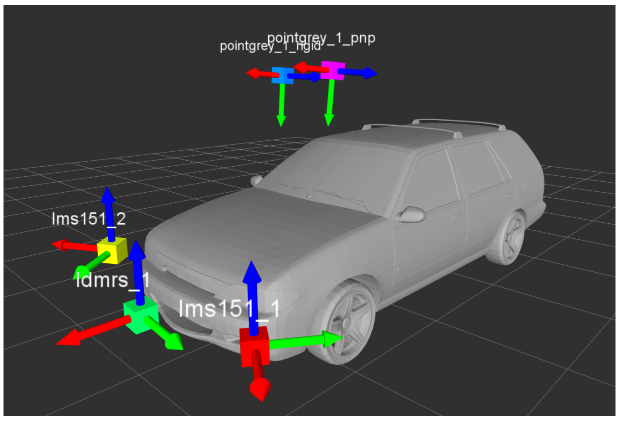
Figure 5.8: Grid pattern ball centers detected by the camera in its image plane (green + symbol) and projected ball centers from SICK LMS151(A) (red dots).

alignment, Figure 5.9 is presented. To ease the visual assessment of the sensor arrangement, a Ford Escort SW 98 3D model was placed in the scene. Although the 3D model is based on the same car model, it does not include any of the extra equipment used on ATLASCAR 1, like the supports for the sensors, thus it is merely a visual guideline.

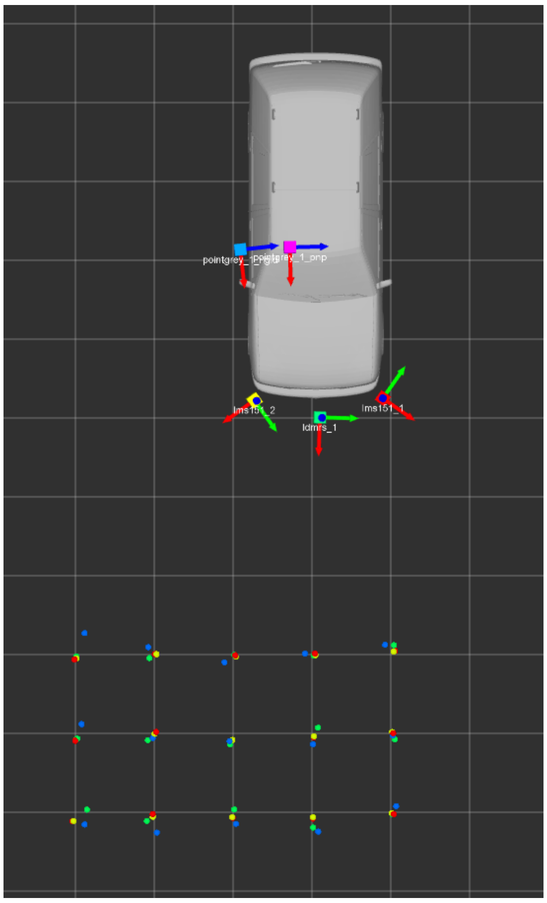
A comparison between Figure 5.3 and 5.9 shows that the estimated pose for the SICK LD-MRS is very close to its real position, however the SICK LMS151 seems to be a few centimeters above its real position.

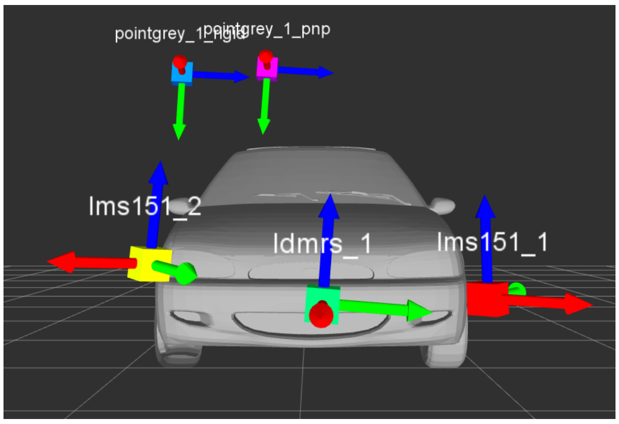
Regarding the two methods used to estimate the camera pose, by comparing Figures 5.3 and 5.9, it is possible to observe that the camera pose estimated by the 3D rigid body transformation method has a significant error, while the extrinsic camera calibration method yielded better results. The source for such a significant error with the 3D rigid body transformation method is likely related to the fact that the ball center coordinates in the camera coordinate system, (X_c, Y_c, Z_c), are calculated based on the detected diameter. Since Z_c is inversely proportional to the detected diameter (see Eq. (3.14)) and is then used to calculate X_c and Y_c (see Eq. (3.7)), even a slight variation of one or two pixels in the detected ball diameter during calibration causes errors in the range of centimeters in all X_c, Y_c and Z_c coordinates. This effect can also explain the significant error obtained for the point cloud fitting between the SICK LMS151 and Point Grey in Table 5.1.

It was observed that even though the detected diameter often suffers small variations, the detected ball center is much more stable, which explains the better results obtained with the extrinsic camera calibration method.



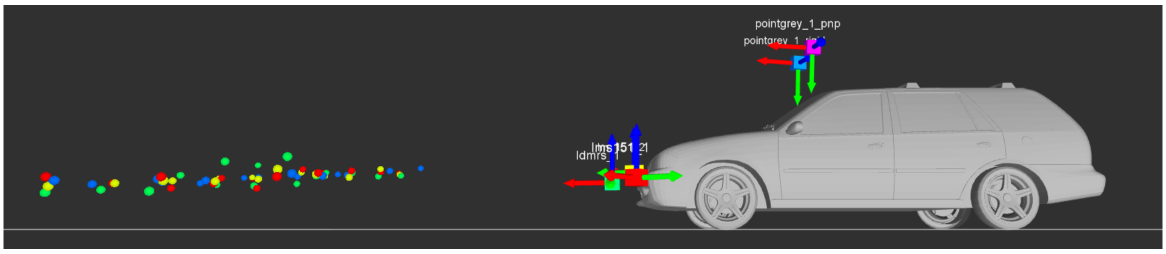
(a) 3D view.





(b) Front view.

(c) Top view with calibrated point clouds.

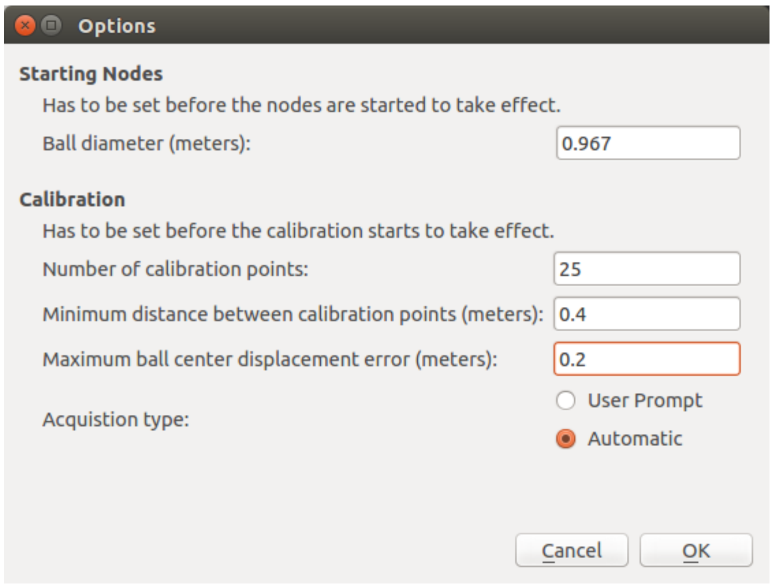


(d) Side view with calibrated point clouds.

Figure 5.9: Sensor arrangement on a Ford Escort SW 98 3D model after grid pattern calibration. SICK LMS151(A) is shown in red, SICK LD-MRS400001 in green, SICK LMS151 in yellow, Point Grey camera using the 3D rigid body transformation method in blue and Point Grey camera using the extrinsic camera calibration method in magenta.

5.2.2 Random Points Calibration

The developed calibration package does not require the ball to follow any particular pattern. Thus, this test is intended to evaluate a calibration in a real world application, using points acquired automatically without predefined positions. Since the ball center points are acquired automatically during the ball's motion and in order to improve results, the number of calibration points is increased to 25 points per point cloud. Figure 5.10 details the other parameters in the GUI "Options" window.


Options

Starting Nodes
Has to be set before the nodes are started to take effect.
Ball diameter (meters): 0.967

Calibration
Has to be set before the calibration starts to take effect.
Number of calibration points: 25
Minimum distance between calibration points (meters): 0.4
Maximum ball center displacement error (meters): 0.2
Acquistion type: User Prompt [✓ Automatic]

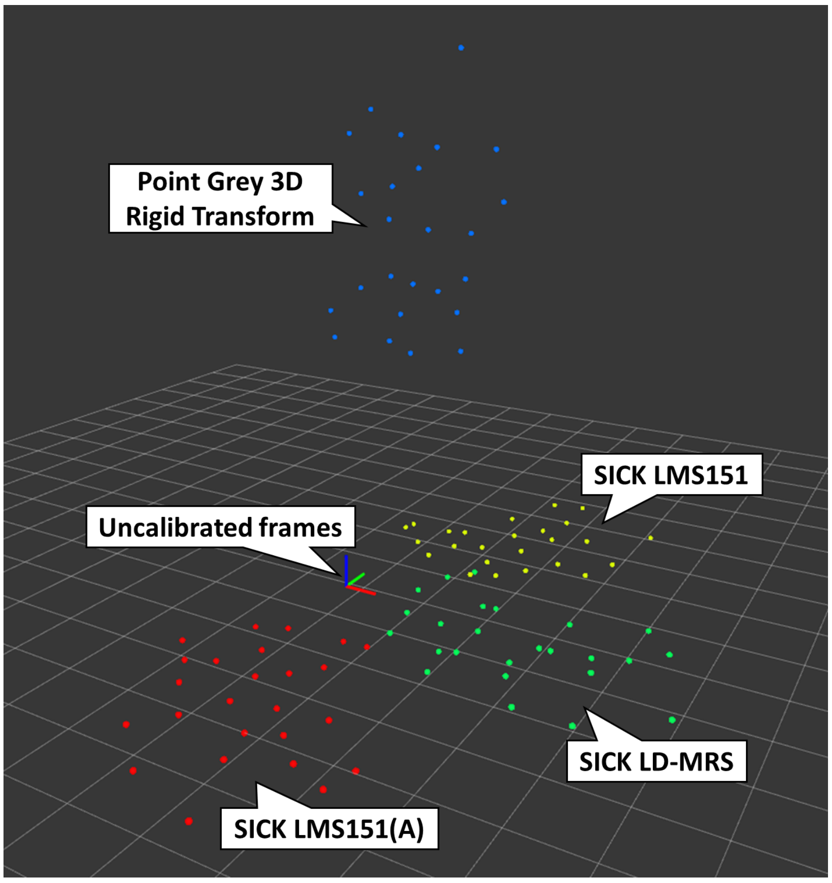
Figure 5.10: Random points calibration parameters specified in the "Options" window.

The acquired point clouds for each sensor with respect to their own frames are shown in Figure 5.11a for point clouds that use the 3D rigid body transformation method; Figure 5.11b shows the detected ball centers in the image plane, used by the extrinsic camera calibration method.

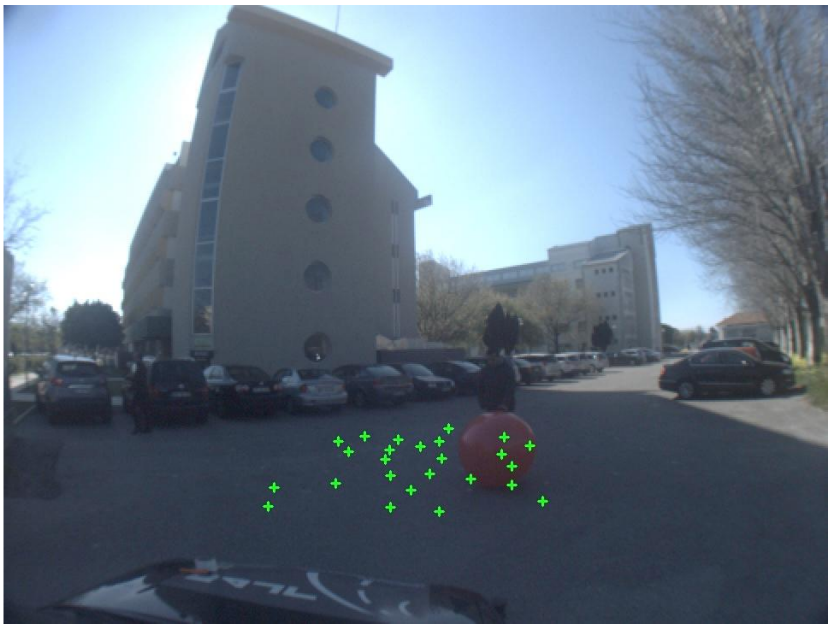
The SICK LMS151(A) sensor is once again chosen as the reference sensor. After the calibration process, the estimated transformation between the SICK LMS151(A) and the other sensors is applied to their corresponding point cloud. Then, as in the previous test, the point cloud fitting mean absolute errors and standard deviation for the 3D rigid body transformation method are computed and presented in Table 5.2. Even though, the number of points in this test is higher than on the grid pattern test, the mean absolute errors and standard deviation do not present significant deviation from the errors and standard deviation verified in the grid pattern test (see Table 5.1).

Regarding the extrinsic camera calibration method, SICK's LMS151(A) 3D point cloud is projected to the image plane and a mean reprojection error of 3.6 pixels is obtained; Figure 5.12 shows the projected SICK's LMS151(A) points and the ball centers detected by the Point Grey camera in its image plane. A comparison with the reprojection error obtained on the first test (3.5 pixels) also shows no significant change, similarly to the 3D rigid body transformation method.

The sensors pose after calibration is shown on Figure 5.13 with the Ford Escort SW 98 3D model. A comparison between Figures 5.3, 5.9 and 5.13, shows that the SICK LMS151



(a) Point clouds to be used with the 3D rigid body transformation method.



(b) Ball centers to be used with the extrinsic camera calibration method.

Figure 5.11: Uncalibrated, randomly acquired, point clouds in their respective coordinate systems.

Table 5.2: Absolute mean error and standard deviation of each sensor calibration, using the 3D rigid body transformation method, against Sick LMS151(A).

	Absolute mean error [mm]	Standard deviation [mm]
SICK LMS151	67	58
SICK LD-MRS	126	59
Point Grey Camera	149	82

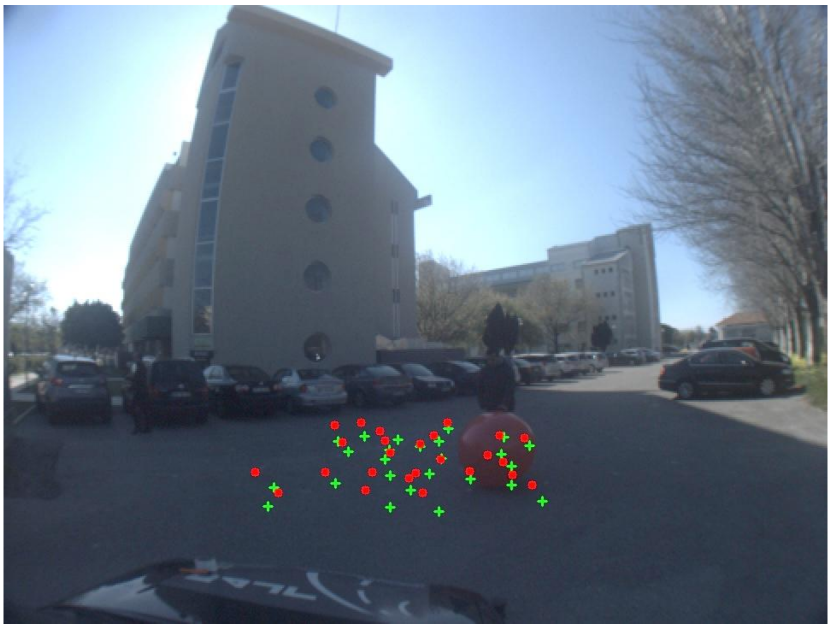
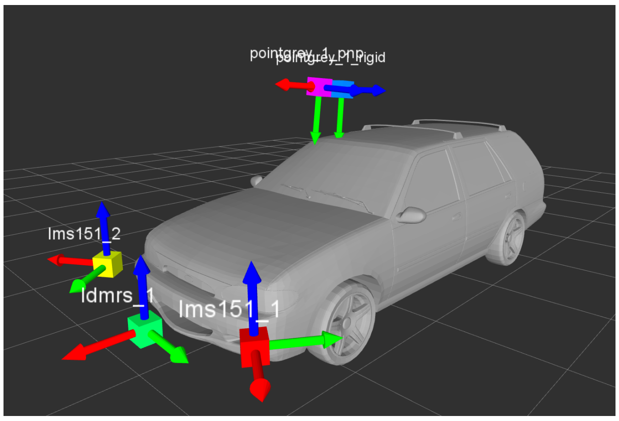
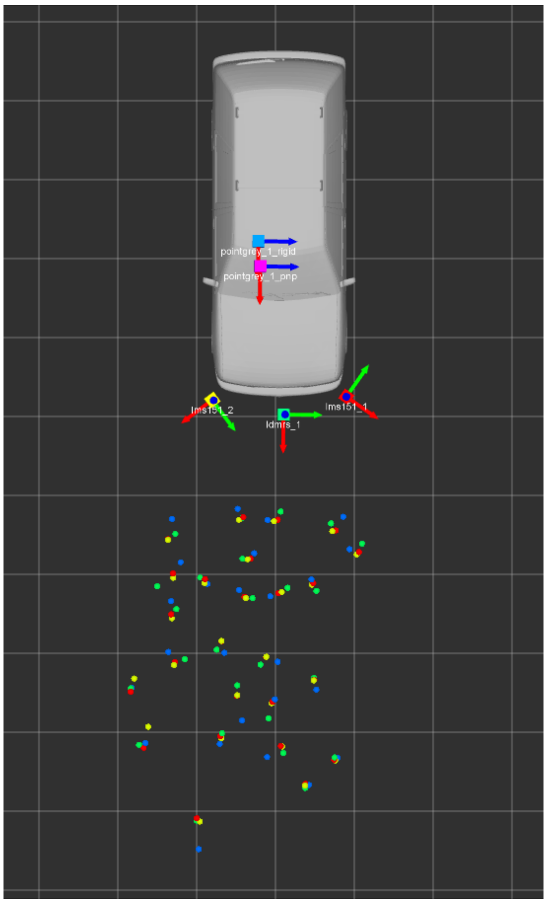


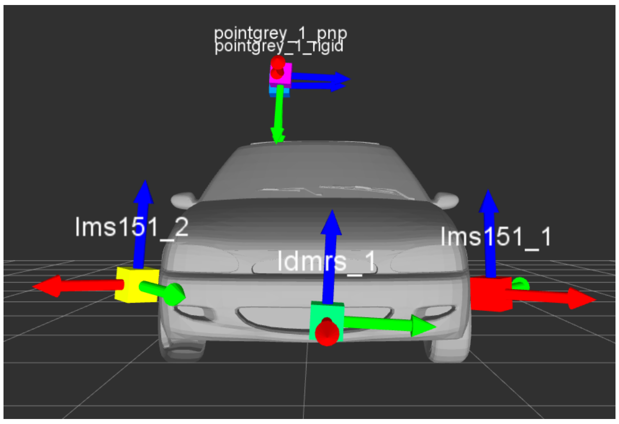
Figure 5.12: Randomly acquired ball centers by the camera in its image plane (green + symbol) and projected ball centers from Sick LMS151(A) (red dots).

and Point Grey camera pose, obtained from the 3D rigid body transformation method, have improved on this test, while the SICK LD-MRS400001 position is identical on both tests. The overall improvement on the 3D rigid body transformation results, especially concerning the Point Grey camera, is likely related to the increase in number of calibration points. The extrinsic camera calibration method results appear to be slightly worse than the 3D rigid body transformation method. However, since the 3D model is not an exact replica of ATLASCAR 1 and the exact transformation between the SICK LMS151(A) and the Point Grey camera is not known, no conclusion can be drawn from Figure 5.13 about which method is more accurate.



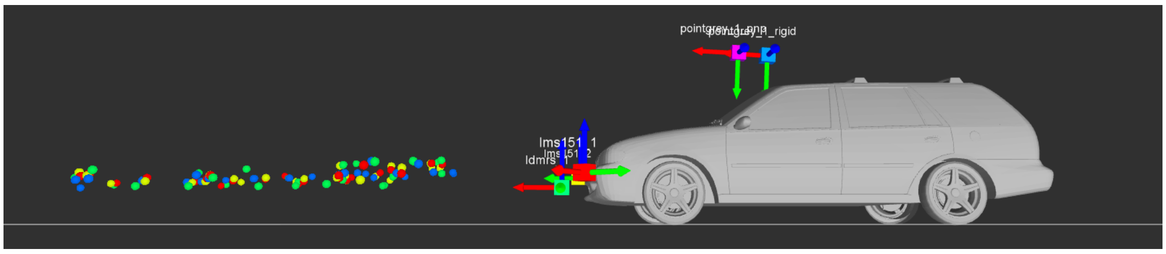
(a) 3D view.





(b) Front view.

(c) Top view with calibrated point clouds.



(d) Side view with calibrated point clouds.

Figure 5.13: Sensor arrangement on a Ford Escort SW 98 3D model after calibration with random points. SICK LMS151(A) is shown in red, SICK LD-MRS400001 in green, SICK LMS151 in yellow, Point Grey camera using the 3D rigid body transformation method in blue and Point Grey camera using the extrinsic camera calibration method in magenta.

5.3 Sensor Data Fusion

Sensor data fusion is described in detail in [50]; in this section, the geometric transformations provided by the extrinsic calibration process described in the previous section are applied to each sensor (or source sensors). Effectively combining sensor data in a reference frame, the target sensor frame.

The following sections present sensor fusion for two tests with different setups: the first test carries out sensor data fusion with the geometric transformations obtained from the tests presented in the previous section, allowing direct comparison between the calibration procedures and methods utilized; the second test, is indoors and introduces the Microsoft Kinect 3D-depth sensor, which offers a different perspective from all the other supported sensors.

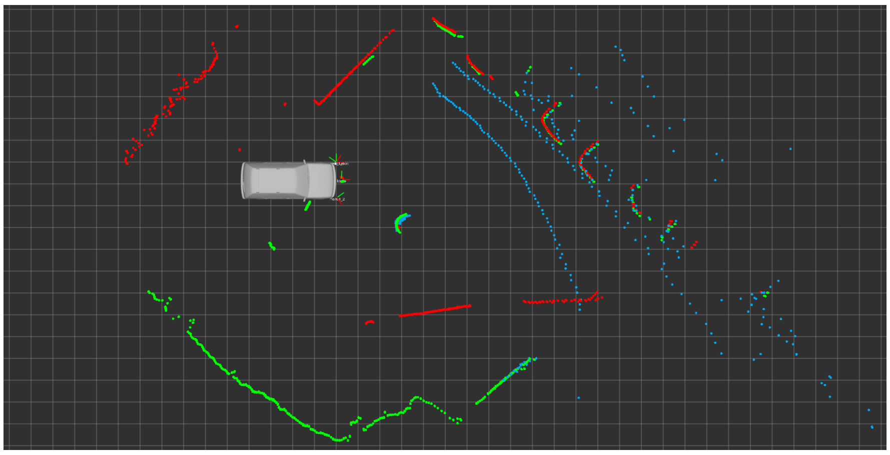
5.3.1 Sensor Data Fusion Using ATLASCAR 1

In this section, sensor fusion with ATLASCAR 1 (setup shown in Figure 5.3) is carried out using the calibration results presented in Sections 5.2.1 and 5.2.2, for grid pattern calibration and random points calibration, respectively.

Figure 5.14 shows a top view of the data fusion between the three laser scanners – two SICKs LMS151 and one LD-MRS400001 – for the grid pattern test and random points test. Similarly, Figure 5.15 presents a front view of the same three laser scanners. Data fusion between the Point Grey camera and the three laser scanners for the grid pattern and random points tests is shown in Figure 5.16. Which also presents sensor fusion with the 3D rigid body transformation and extrinsic camera calibration methods.

By analysing Figure 5.15, it is possible to observe that the SICK LMS151(A) is not on the same plane as the SICK LMS151 and SICK LD-MRS400001. Figures 5.14 and 5.16 both show that the SICK LMS151(A) is in fact tilted around its X-axis, as it is detecting the ground on one side and is higher than the other laser scanners on the opposite side. These observations indicate that either the car or the SICK LMS151(A) support on the car is tilted.

It is also possible to observe in Figure 5.16 that the best results for the grid pattern calibration test are obtained by the extrinsic camera calibration method, shown in Figure 5.16b; as for the random points calibration test, the best results are from the 3D rigid body transformation, shown in Figure 5.16c. These results agree with the analysis done in Sections 5.2.1 and 5.2.2, although the difference between the 3D rigid body transformation and extrinsic camera calibration methods, for the random points test is now much more obvious. Between Figures 5.16b and 5.16c it is not clear which provides the best overall results for sensor data fusion as they are both very similar.


(a) Grid pattern calibration.

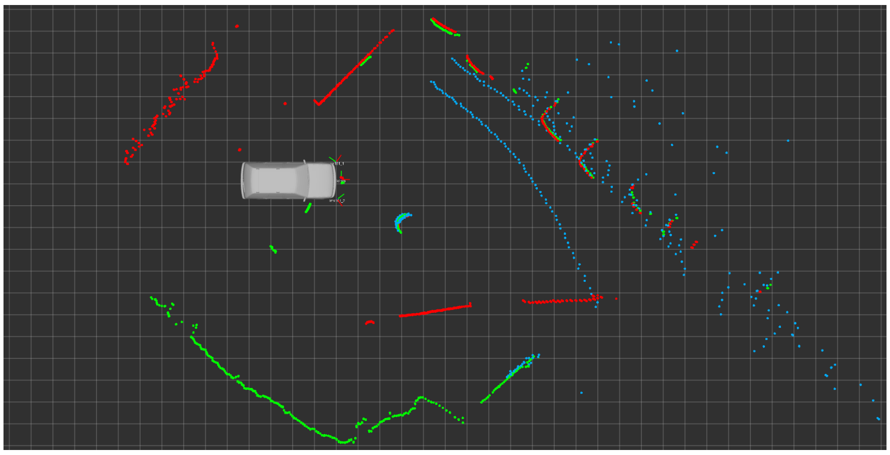
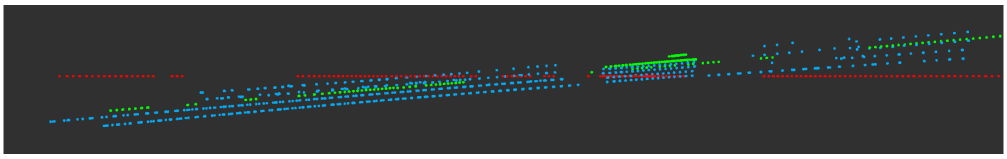

(b) Random points calibration.

Figure 5.14: ATLASCAR 1 sensor fusion top view between the SICK LMS151(A) (red), SICK LMS151 (green) and SICK LD-MRS400001 (cyan).


(a) Grid pattern calibration.

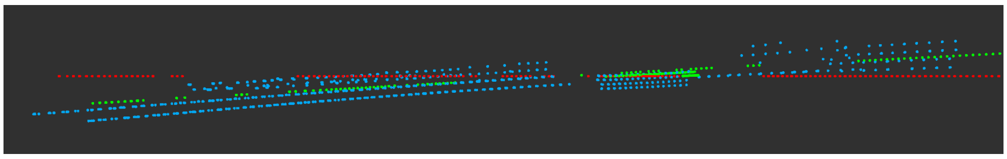

(b) Random points calibration.

Figure 5.15: ATLASCAR 1 sensor fusion front view between the SICK LMS151(A) (red), SICK LMS151 (green) and SICK LD-MRS400001 (cyan).

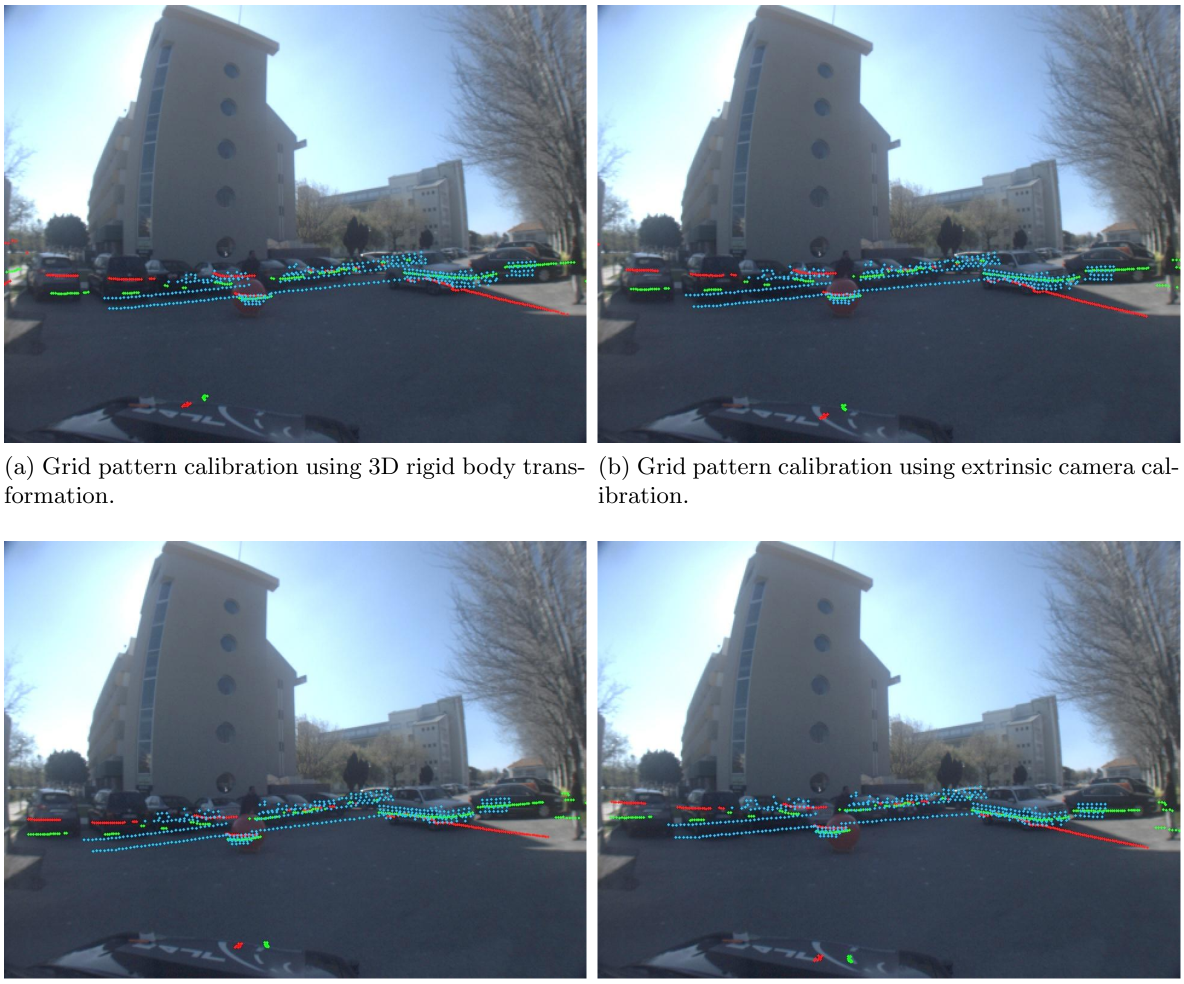

(a) Grid pattern calibration using 3D rigid body transformation.
(b) Grid pattern calibration using extrinsic camera calibration.
(c) Random points calibration using 3D rigid body transformation.
(d) Random points calibration using extrinsic camera calibration.

Figure 5.16: ATLASCAR 1 sensor fusion in the Point Grey camera image plane with SICK LMS151(A) (red), SICK LMS151 (green) and SICK LD-MRS400001 (cyan).

5.3.2 Indoors Sensor Data Fusion with Microsoft Kinect

Microsoft Kinect offers a unique perspective of the world when compared to the other supported sensors, it has the ability to generate a detailed 3D point cloud with RGB registration. Thus, within its operating range, it also provides more information about its surroundings than any other supported sensor. The test presented here provides ball detection results for Microsoft Kinect and takes advantage of its unique features to evaluate sensor data fusion with four other sensors – two SICKs LMS151, a SICK LD-MRS400001 and a Point Grey FL3-GE-28S4-C camera, Figure 5.17 presents the setup.

Before proceeding with sensor data fusion, it is necessary to perform an extrinsic calibration of the sensor setup. As in the previous tests, one of the SICK LMS151 is chosen as the reference sensor and is therefore named SICK LMS151(A). The multisensor calibration GUI, described in detail in Chapter 4, is utilized to perform the extrinsic calibration. Given the

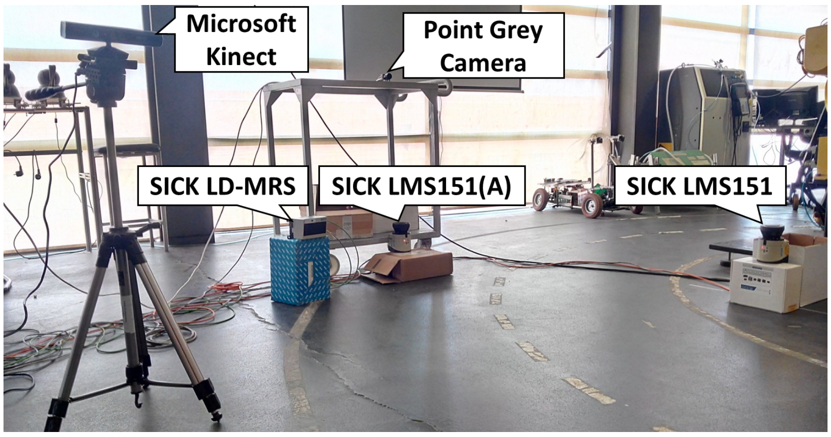
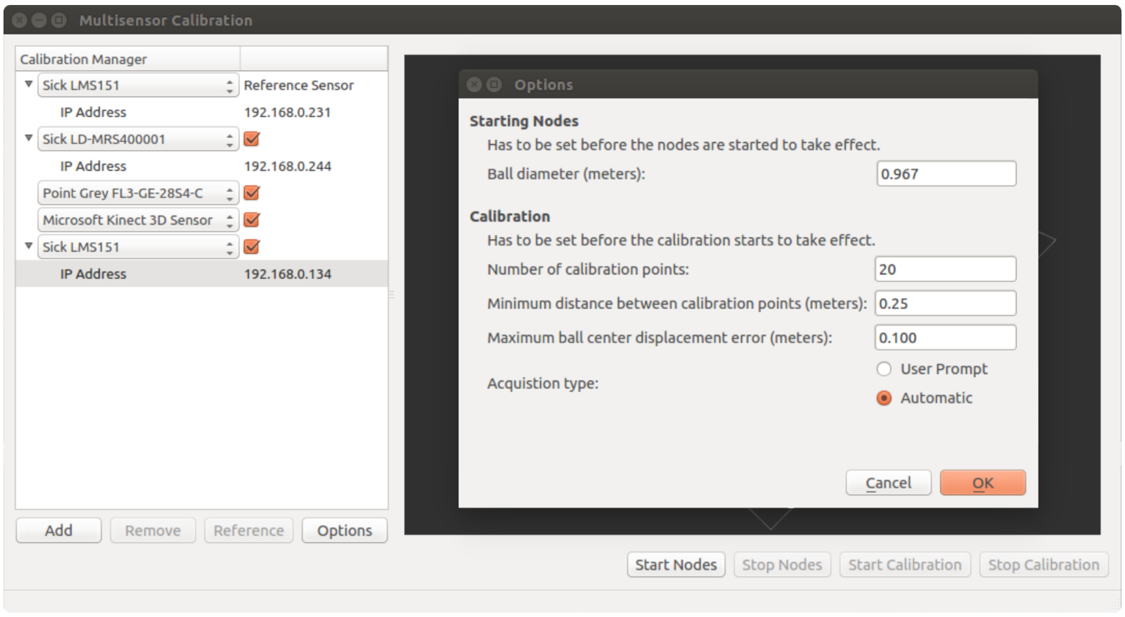


Figure 5.17: Indoors sensor fusion setup.

much lower operating range of Microsoft Kinect, when compared to the other sensors, and the reduced space available for the test, the minimum distance between calibration points has been reduced to 0.25 m, instead of the default 0.5 m. Sensor configuration and calibration options are displayed in Figure 5.18. Similarly to the calibration tests performed with ATLASCAR 1, the extrinsic calibration results and reprojection error is presented in Figures 5.19 and 5.20, respectively.


Multisensor Calibration

Calibration Manager	
Sick LMS151	Reference Sensor
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.0.244
Point Grey FL3-GE-28S4-C	✓
Microsoft Kinect 3D Sensor	✓
Sick LMS151	✓
IP Address	192.168.0.134

Options

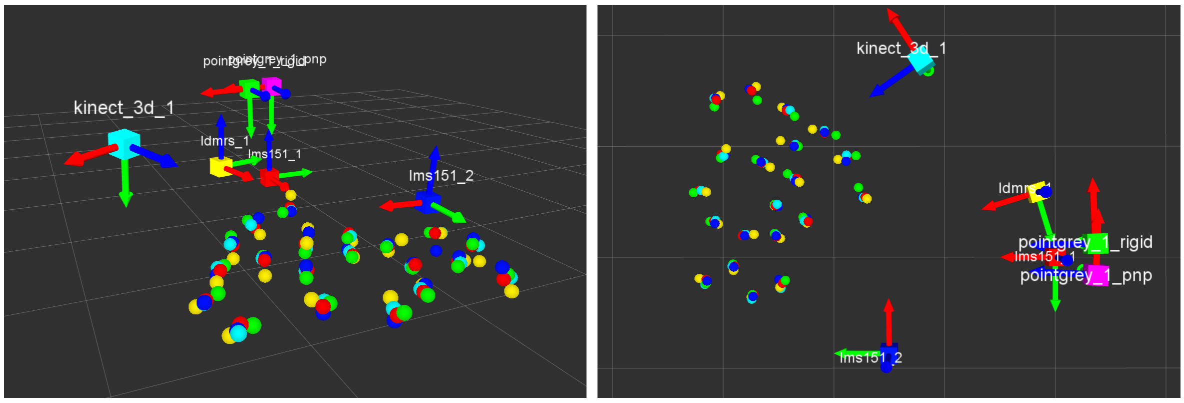
Starting Nodes
Has to be set before the nodes are started to take effect.
Ball diameter (meters): 0.967

Calibration
Has to be set before the calibration starts to take effect.
Number of calibration points: 20
Minimum distance between calibration points (meters): 0.25
Maximum ball center displacement error (meters): 0.100
Acquistion type: User Prompt [✓ Automatic]

Figure 5.18: “Calibration Manager” and “Options” window for the indoors test.

Qualitative analysis of Figure 5.19 shows good results regarding the extrinsic sensor calibration of the setup shown in Figure 5.17. It is also noticeable the proximity between the ball centers detected by Kinect and the ones detected by SICK LMS151(A), which demonstrates the accuracy of its 3D-depth sensor and ball detection algorithm.

After the calibration process, the estimated transformations are applied to the data of each source sensor – SICK LMS151, SICK LD-MRS400001, Point Grey camera and Microsoft



(a) 3D view.

(b) Top view.

Figure 5.19: Indoors test calibration results. SICK LMS151(A) is shown in red, SICK LMS151 in blue, SICK LD-MRS400001 in yellow, Point Grey camera using the 3D rigid body transformation method in green, Point Grey camera using the extrinsic camera calibration method in magenta and Microsoft Kinect 3D-depth sensor in cyan.

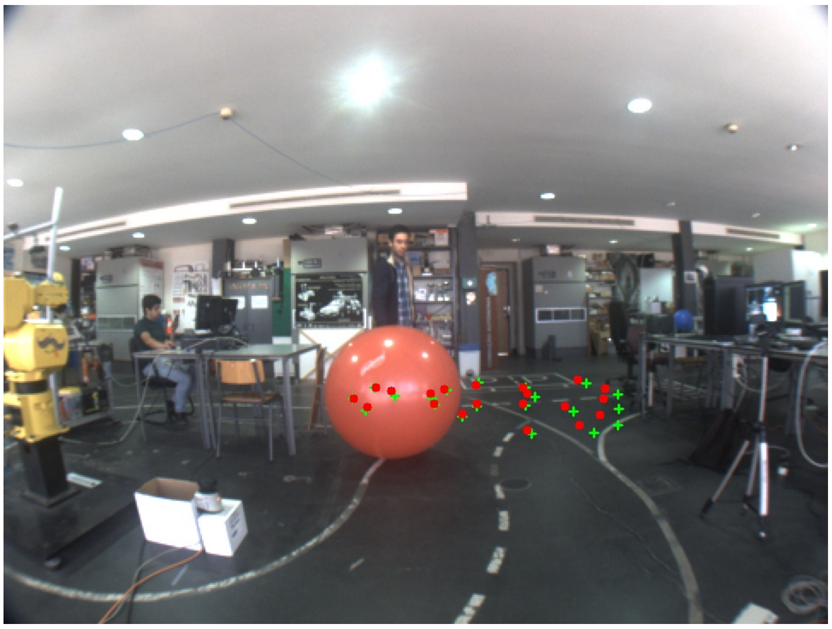


Figure 5.20: Indoors test camera ball centers in its image plane (green + symbol) and projected ball centers from SICK LMS151(A) (red dots).

Kinect 3D-depth sensor. Data fusion between the three laser scanners and the Point Grey camera is shown in Figure 5.21. Figure 5.22 presents sensor data fusion between the three laser scanners and Microsoft Kinect 3D-depth sensor. Kinect data is not projected into the camera image plane in Figure 5.21, as it would clutter the images.

The comparison between Figures 5.17 and 5.19 shows that both the 3D rigid body transformation and extrinsic camera calibration methods provided close results for the Point Grey camera pose. However, from Figure 5.21 it is noticeable that data transformed using the extrinsic camera calibration fits the objects in the scene better than the data transformed using the 3D rigid body transformation method (e.g. see the calibration ball and box supporting the SICK LMS151).

Regarding data fusion between the three laser scanners and Microsoft Kinect (Figure 5.22), it is clear that the three laser scanners follow the curvature of the ball and transition to the checkerboard plane along with the RGB registered[1] point cloud acquired from Microsoft Kinect 3D-depth sensor. Note that data from the SICK LD-MRS400001 is not very precise at such low operating ranges which explains the significant fluctuation in its data.

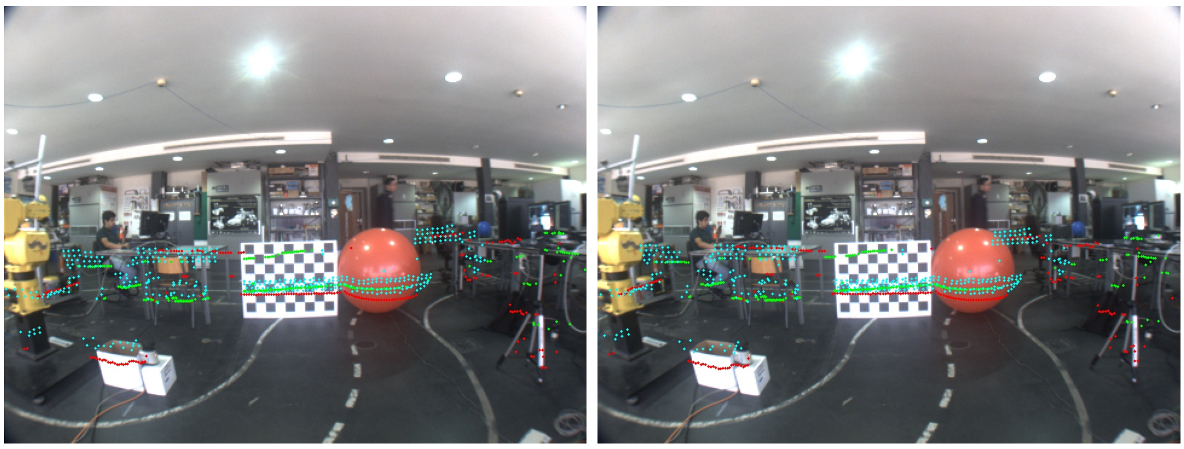
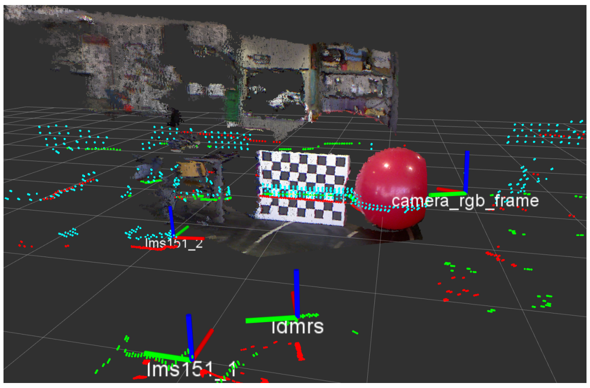
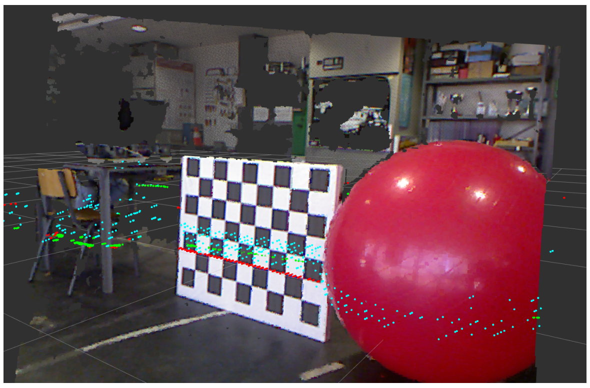

(a) Camera pose obtained through the 3D rigid body transformation method. (b) Camera pose obtained through the extrinsic camera calibration method.

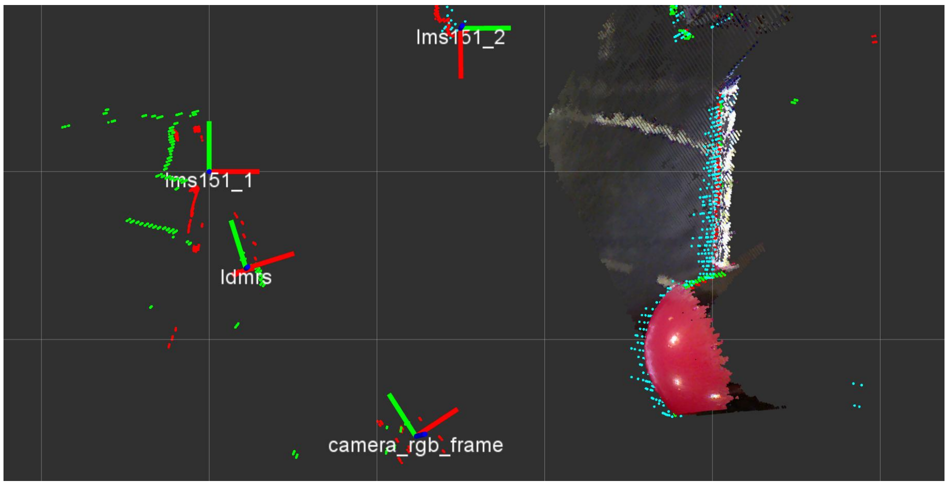
Figure 5.21: Indoors test sensor fusion in the Point Grey camera image plane with SICK LMS151(A) (red), SICK LMS151 (green) and SICK LD-MRS400001 (cyan).

[1]RGB registration is done by OpenNI and is only used for visualization purposes.

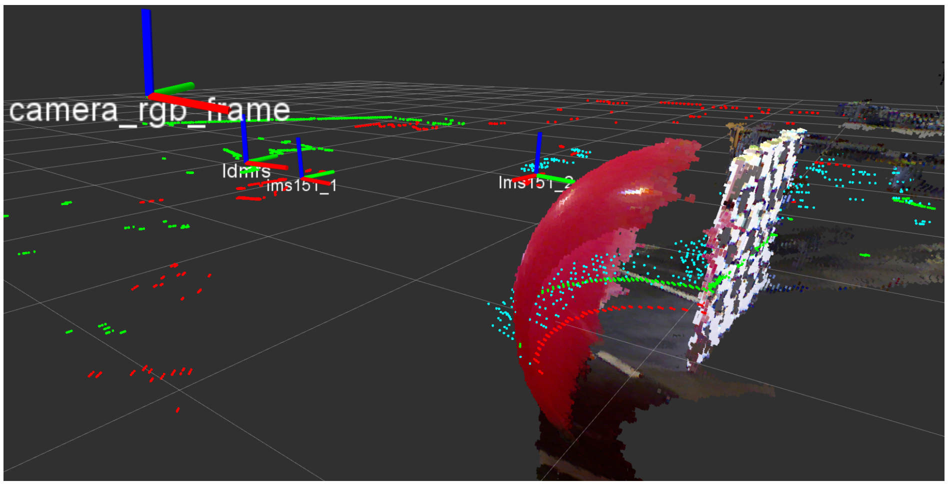


(a) 3D view.


(b) Kinect point of view.



(c) Top view.



(d) Ball to checkerboard plane transition.

Figure 5.22: Indoors test calibration results. SICK LMS151(A) is shown in red, SICK LMS151 in green, SICK LD-MRS400001 in cyan and Microsoft Kinect 3D-depth sensor point cloud is shown with RGB registration.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In the fields of autonomous driving and advanced driver assistance systems, perception of the vehicle's surroundings is key for collision avoidance and path planning; vehicles often resort to several sensors of different types to gather information about its surroundings. The work presented in this document extends an automatic extrinsic calibration software, which computes the geometric transformations between each sensor and a reference sensor, to vision-based sensors. The ultimate goal is to apply the transformations to each sensor, effectively merging all sensor data in a single frame. To allow the calibration package to be easily expandable to other sensors, and usable with any sensor configuration, a GUI was developed.

To integrate vision-based sensors in the calibration package it is first necessary to detect the calibration target, a red ball. Two approaches have been presented to detect the ball: the first is based on Hough circle transform; the second is named "Approximated Contour". Hough circle transform proved to be unreliable and required constant user input during calibration, as the ball got closer or further away from the camera. The "Approximated Contour" approach revealed to be much more robust than Hough circle transforms, as it only requires the definition of HSV intervals for color thresholding to reliably detect the ball. To take advantage of the automatic nature of the calibration package, uniform lighting is recommended so the HSV intervals can be set at the start of the calibration procedure, thus avoiding user input during the procedure. In order to estimate the geometric transformation between the camera and the reference sensor, two methods are proposed: the 3D rigid body transformation and the extrinsic camera calibration methods. Results show that the extrinsic camera calibration method performed better in two of three extrinsic calibration tests; and when it performed worse, its estimated sensor pose is close to the one estimated by the 3D rigid body transformation. However, the tests sample is too small to conclude, with certainty, which method performs better. Therefore, both methods are included in the calibration package.

A GUI for the calibration package has also been successfully developed in this work, providing an easy-to-use tool which allows the package to be easily expandable to more sensors and any number of the supported sensors, in any configuration, can be calibrated. Additionally, it allows the user to easily monitor the calibration process through the integrated 3D visualizer based on Rviz. To illustrate the process of expanding the calibration package to new sensors, Microsoft Kinect 3D-depth sensor has also been successfully integrated into the calibration package. The biggest challenge when integrating a new sensor is developing an algorithm to detect the ball; however, the calibration package already covers most of the main sensor data types (2D, 3D and vision-based). Thus, existing ball detection algorithms can serve as guidelines for new implementations. Throughout the development of this work the GUI has been extensively used as it greatly simplifies extrinsic calibration tests.

Finally, sensor data fusion in a single frame has been successfully achieved. Tests showed consistent results with different sensor configurations and environments – outdoors, using ATLASCAR 1, and indoors. Data fusion also allowed to evaluate the accuracy of the extrinsic calibration by observing if the data from each sensor fits the objects and transitions between objects found in the scene. Since data fusion is consistent with the scene, it can be concluded that the results provided by the calibration package are accurate.

Parts of this work have been published in "Self calibration of multiple LIDARs and cameras on autonomous vehicles" [51].

6.2 Future Work

Future work may include extensive extrinsic calibration tests to evaluate which of the two camera calibration methods, 3D rigid body transformation or extrinsic camera calibration, is the most accurate. Extrinsic calibration tests showed that the estimated geometric transformations by the calibration package are accurate. Sensor data fusion tests also confirmed those results; however, it is not possible to measure how accurate those results are without knowing the exact geometric transformation between the sensors being calibrated. Thus, new tests should be carried out with the sensors in known positions and orientations.

The SICK LMS151 and SICK LD-MRS400001 ball detection algorithms require information about their position relative to the ball equator – above or below. Removing this requirement could be achieved by bouncing the ball in front of the sensors and measuring the diameter variation from the ball sections.

More sensors should be integrated into the calibration package; a good start would be stereo vision, since Pereira [1] has already developed most of the process, even though its results are lacking.

Regarding the GUI, there are many minor improvements that can be made; however, there are also some high-priority features that need to be implemented: if the requirement for information about the SICK LMS151 and SICK LD-MRS400001 positions relative to the ball equator is not removed, a sensor-specific option is needed, so the user can provide that information (currently it is always assumed that those sensors are below the equator); a dialog window can be developed to add new sensors to the GUI, removing the need to change its source code when a new sensor is added; finally, the ability to save and load, sensor configurations and options from a file is also interesting and could be developed.

The next step after achieving sensor data fusion is building a navigation map to plan ATLASCAR 1, or ATLASCAR 2, motion using techniques like Velocity Obstacles, Rapidly-exploring Random Tree (RRT), Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE), SPArse Roadmap Spanner (SPARS) or others.

Bibliography

[1] M. Pereira, 'Automated calibration of multiple LIDARs and cameras using a moving sphere', Master Thesis, University of Aveiro, 2015.

[2] S. Wasielewski and O. Strauss, 'Calibration of a multi-sensor system laser rangefinder / camera', in *IEEE Intelligent Vehicles Symposium*, 1995, pp. 472–477.

[3] Q. Z. Q. Zhang and R. Pless, 'Extrinsic calibration of a camera and laser range finder (improves camera calibration)', *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, vol. 3, pp. 2301–2306, 2004.

[4] K. Levenberg, 'A method for the solution of certain non-linear problems in least squares', *Quarterly Journal of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[5] D. W. Marquardt, 'An Algorithm for Least-Squares Estimation of Nonlinear Parameters', *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[6] D. Scaramuzza, A. Harati and R. Siegwart, 'Extrinsic self calibration of a camera and a 3D laser range finder from natural scenes', *IEEE International Conference on Intelligent Robots and Systems*, pp. 4164–4169, 2007.

[7] L. Quan and Z. Lan, 'Linear N-point camera pose determination', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 8, pp. 774–780, 1999.

[8] Z. Zhang, 'Iterative point matching for registration of free-form curves and surfaces', *International Journal of Computer Vision*, vol. 13, no. 2, pp. 119–152, 1994.

[9] M. Ruan and D. Huber, 'Calibration of 3D sensors using a spherical target', in *2014 International Conference on 3D Vision, 3DV 2014*, IEEE, 2014, pp. 187–193.

[10] M. Franaszek, G. S. Cheok, K. S. Saidi and C. Witzgall, 'Fitting Spheres to Range Data From 3-D Imaging Systems', *IEEE Transactions On Instrumentation And Measurement*, vol. 58, no. 10, pp. 3544–3553, 2009.

[11] M. Almeida, P. Dias, M. Oliveira and V. Santos, '3D-2D laser range finder calibration using a conic based geometry shape', *Proceedings of the 9th International Conference on Image Analysis and Recognition - ICIAR 2012*, vol. 7324, pp. 312–319, 2012.

[12] P. Besl and N. McKay, 'A Method for Registration of 3-D Shapes', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[13] B. L. Will Schroeder, Ken Martin, *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, 4th Ed. Kitware, 2006, p. 528.

[14] D. Coimbra, 'LIDAR Target Detection and Segmentation in Road Environment', Master Thesis, University of Aveiro, 2013, p. 104.

[15] J. Xavier, M. Pacheco, D. Castro, A. Ruano and U. Nunes, 'Fast Line, Arc/Circle and Leg Detection from Laser Scan Data in a Player Driver', in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, IEEE, 2005, pp. 3930–3935.

[16] B. K. P. Horn, 'Closed-form solution of absolute orientation using unit quaternions', *Journal of the Optical Society of America A*, vol. 4, no. 4, p. 629, 1987.

[17] K. S. Arun, T. S. Huang and S. D. Blostein, 'Least-Squares Fitting of Two 3-D Point Sets', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 698–700, 1987.

[18] SICK AG, *2D laser scanners LMS1xx / LMS15x / Outdoor*. [Online]. Available: `https://www.sick.com/de/en/detection-and-ranging-solutions/2d-laser-scanners/lms1xx/lms151-10100/p/p141840` (visited on 19/05/2016).

[19] ——, 'LMS1xx Laser Measurement Sensors', SICK AG, Waldkirch, Germany, Tech. Rep., 2015.

[20] ——, *3D laser scanners LD-MRS / LD-MRS 4 Layer / Outdoor / Long Range*. [Online]. Available: `https://www.sick.com/de/en/detection-and-ranging-solutions/3d-laser-scanners/ld-mrs/ld-mrs400001/p/p112355` (visited on 19/05/2016).

[21] ——, 'LD-MRS Laser Measurement Sensor', SICK AG, Waldkirch, Germany, Tech. Rep., 2014.

[22] Point Grey Research, 'Flea3 GigE Technical Reference', Point Grey, Richmond, Canada, Tech. Rep., 2013.

[23] S. B. Gokturk, H. Yalcin and C. Bamji, 'A time-of-flight depth sensor - System description, issues and solutions', *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 35–45, 2004.

[24] MESA Imaging, 'SR4000 Data Sheet', MESA Imaging AG, Zurich, Switzerland, Tech. Rep., 2011.

[25] J. Han, L. Shao, D. Xu and J. Shotton, 'Enhanced computer vision with Microsoft Kinect sensor: A review', *IEEE Transactions on Cybernetics*, vol. 43, no. 5, pp. 1318–1334, 2013.

[26] J. Smisek, M. Jancosek and T. Pajdla, '3D with Kinect', in *IEEE International Conference on Computer Vision*, 2011, pp. 1154–1160.

[27] Microsoft, *Kinect for Windows Sensor Components and Specifications*. [Online]. Available: `https://msdn.microsoft.com/pt-pt/library/jj131033.aspx` (visited on 21/05/2016).

[28] P. Dias, M. Matos and V. Santos, '3D reconstruction of real world scenes using a low-cost 3D range scanner', *Computer-Aided Civil and Infrastructure Engineering*, vol. 21, no. 7, pp. 486–497, 2006.

[29] J. Dias, 'Distanciómetro 3D baseado numa unidade laser 2D em movimento contínuo', Master Thesis, University of Aveiro, 2009.

[30] R. Pascoal and V. Santos, 'Compensation of Azimuthal Distortions on a Free Spinning 2D Laser Range Finder for 3D Data Set Generation', *10th Int. Conference on Mobile Robots and Competitions ROBOTICA 2010*, pp. 41–46, 2010.

[31] D. Matos, 'Deteção do Espaço Navegável para o ATLASCAR usando informação 3D', Master Thesis, University of Aveiro, 2013.

[32] R. B. Rusu and S. Cousins, '3D is here: point cloud library', *IEEE International Conference on Robotics and Automation*, pp. 1–4, 2011.

[33] Point Cloud Library (PCL), *About PCL*. [Online]. Available: `http://www.pointclouds.org/about/` (visited on 24/05/2016).

[34] G. Bradski, 'The OpenCV Library', *Dr Dobbs Journal of Software Tools*, 2000. [Online]. Available: `http://www.drdobbs.com/open-source/the-opencv-library/184404319`.

[35] OpenCV, *OpenCV Documentation*, 2016. [Online]. Available: `http://docs.opencv.org/2.4/index.html` (visited on 24/05/2016).

[36] Qt Company, *About Qt*, 2015. [Online]. Available: `https://wiki.qt.io/About%7B%5C_%7DQt` (visited on 24/05/2016).

[37] ——, *Qt Documentation: Qt Platform Abstraction*, 2016. [Online]. Available: `http://doc.qt.io/qt-5/qpa.html` (visited on 24/05/2016).

[38] ——, *Qt Documentation: Signals & Slots*, 2016. [Online]. Available: `http://doc.qt.io/qt-5/signalsandslots.html` (visited on 24/05/2016).

[39] ——, *Qt Documentation: Using the Meta-Object Compiler*, 2016. [Online]. Available: `http://doc.qt.io/qt-5/moc.html` (visited on 24/05/2016).

[40] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng, 'ROS: an open-source Robot Operating System', in *ICRA Workshop on Open Source Software*, 2009.

[41] ROS, *roslaunch/XML*, 2016. [Online]. Available: `http://wiki.ros.org/roslaunch/XML` (visited on 01/06/2016).

[42] LAR-DEMUA, *LAR Toolkit V4*, 2015. [Online]. Available: `http://lars.mec.ua.pt/lartk4/` (visited on 24/05/2016).

[43] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, 1st Ed., M. Loukides, Ed. O'Reilly, 2008, vol. 1, p. 555.

[44] J.-Y. Bouguet, *Complete Camera Calibration Toolbox for Matlab*, 1999. [Online]. Available: `http://www.vision.caltech.edu/bouguetj/calib%7B%5C_%7Ddoc/`.

[45] C. Kimme, D. Ballard and J. Sklansky, 'Finding circles by an array of accumulators', *Communications of the ACM*, vol. 18, no. 2, pp. 120–122, 1975.

[46] U. Ramer, 'An iterative procedure for the polygonal approximation of plane curves', *Computer Graphics and Image Processing*, vol. 1, no. 3, pp. 244–256, 1972.

[47] D. H. Douglas and T. K. Peucker, 'Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature', *Classics in Cartography: Reflections on Influential Articles from Cartographica*, vol. 10, no. 2, pp. 15–28, 1973, ISSN: 0317-7173.

[48] ROS, *rqt_graph: Package Summary*, 2014. [Online]. Available: `http://wiki.ros.org/rqt%7B%5C_%7Dgraph` (visited on 02/06/2016).

[49] ——, *ROS/Concepts*, 2014. [Online]. Available: `http://wiki.ros.org/ROS/Concepts` (visited on 02/06/2016).

[50] D. L. Hall and J. Llinas, *Handbook of Multisensor Data Fusion*, 2nd Ed. CRC Press, 2008.

[51] M. Pereira, D. Silva, V. Santos and P. Dias, 'Self calibration of multiple LIDARs and cameras on autonomous vehicles', *Robotics and Autonomous Systems*, 2016.

Appendix A

Calibration Procedure

A.1 Package Installation

The calibration package has been used and developed under Ubuntu 14.04 and ROS indigo.

1. Download the calibration package and its ROS package dependencies from the following repository: https://github.com/davidtvs/calibration;

2. Before compiling the ROS packages, install the following system dependencies: FlyCapture 2.x SDK and libmesasr-dev (see Section 2.3.5 for a short description);

3. Compile the packages;

A.2 Automatic Calibration

This section will detail how to use the calibration application in automatic mode.

1. Run the application with the command "rosrun calibration_gui calibration_gui". The GUI main window will be displayed, as shown in Figure A.1;

2. Click the "Add" button to add a new sensor to the list. Select the desired sensor from the combo box and fill its sensor-specific options. The "Remove" button removes a sensor from the list. "Reference" makes the currently selected sensor on the list the reference sensor. In Figure A.2, as an example, the ATLASCAR 1 sensor configuration is added to the list and one of the SICK LMS151 is chosen as reference.

3. Click the "Options" button to access the calibration options, as shown in Figure A.3. Set the desired options for the calibration procedure and set "Acquisition type" to "Automatic". Click "OK" to save the options.

4. Sensor nodes can now be launched by clicking the "Start Nodes" button. Note that only sensors on the list with a checked checkbox will be launched and calibrated. As shown in Figure A.4, a message box will appear to confirm the ball diameter;

    4.1. If a Point Grey camera is used, three windows will be displayed (see Figure A.5): a window with the image received from the camera where a circle representing the detected ball is drawn; a window containing a binary image of the image received from the camera; and a control window with six trackbars, two for each HSV component. On the control window, using the trackbars, the user sets the desired HSV range for color thresholding. The binary image is the result of the color thresholding process applied to the image received from the camera.

5. Click "Start Calibration" in order to start calibrating the chosen sensors. A message box is displayed to confirm calibration options, as shown in Figure A.6. Answering "Yes" will start the calibration procedure;

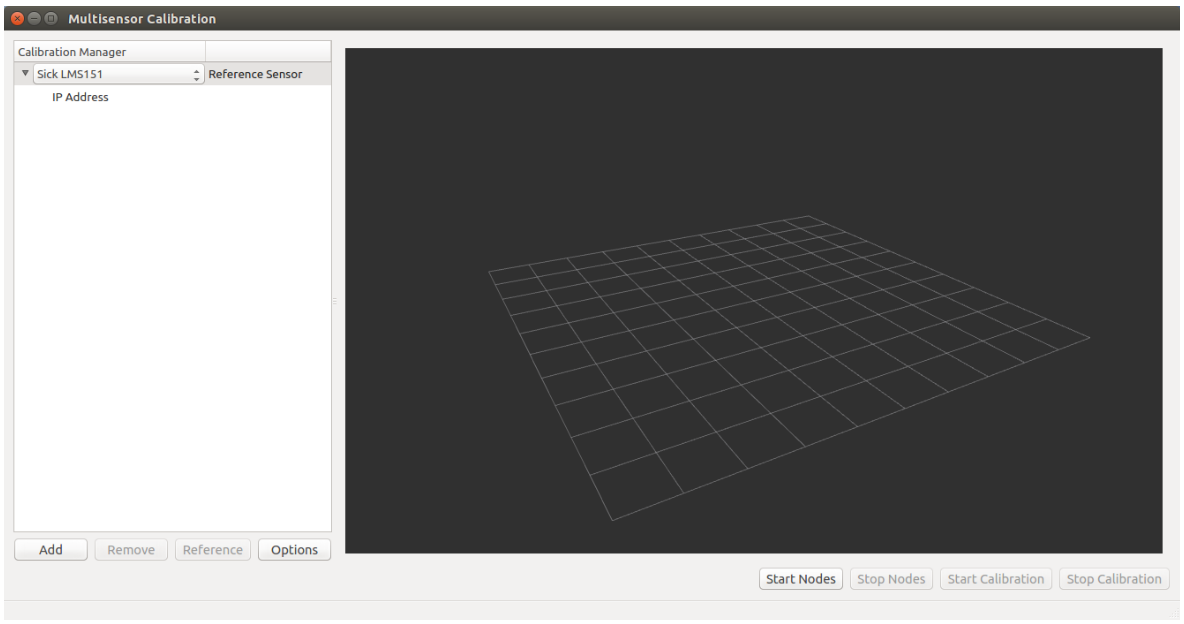
   5.1. Ball center points are now acquired and drawn on the visualizer (Figure A.7) automatically, as the ball moves, until the user-defined number of calibration points is reached. Note that the calibration procedure can be interrupted at any time by clicking on the "Stop Calibration" or "Stop Nodes" button;

   5.2. A message box will notify the user when the number of calibration points is reached (see Figure A.8). At this point the sensors poses are already estimated and displayed on the 3D visualizer;

6. The user can perform another calibration by clicking on "Start Calibration" or return the GUI to its initial state by clicking "Stop Nodes".

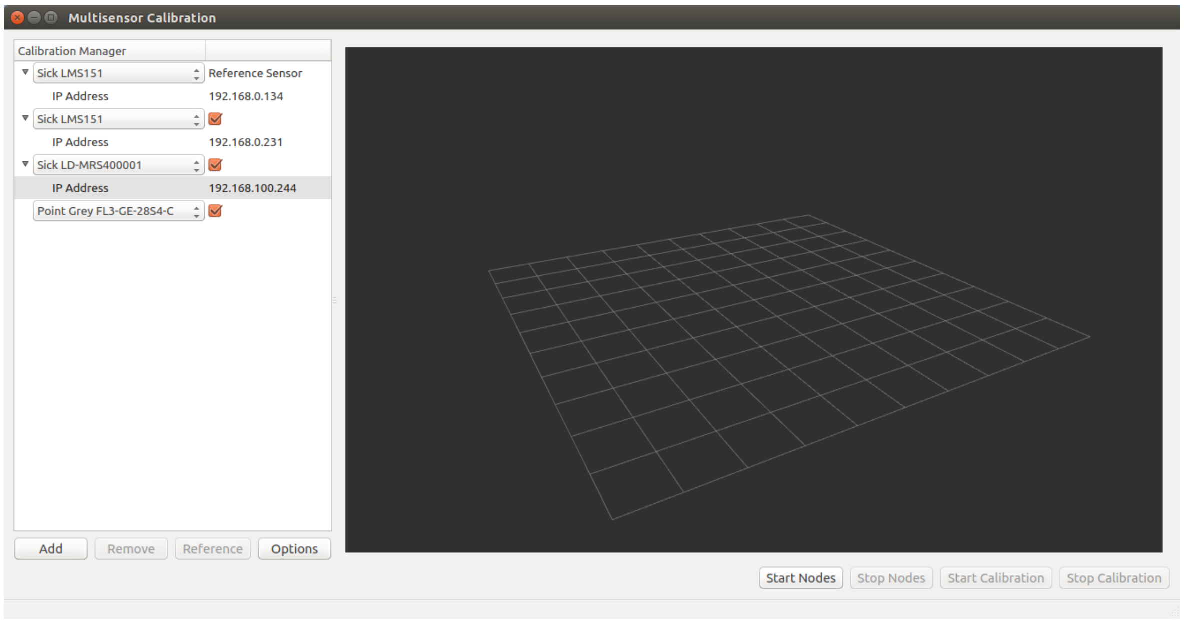
   6.1. Clicking on "Start Calibration" sends the application to the previous step;

   6.2. Clicking on "Stop Nodes" will display a message box to confirm if the user wants to terminate all nodes, as shown in Figure A.9. If the answer is "Yes", the "Terminating nodes...Please wait." message is displayed (see Figure A.10) and the user must wait until it disappears, at that point, every node has been safely terminated.



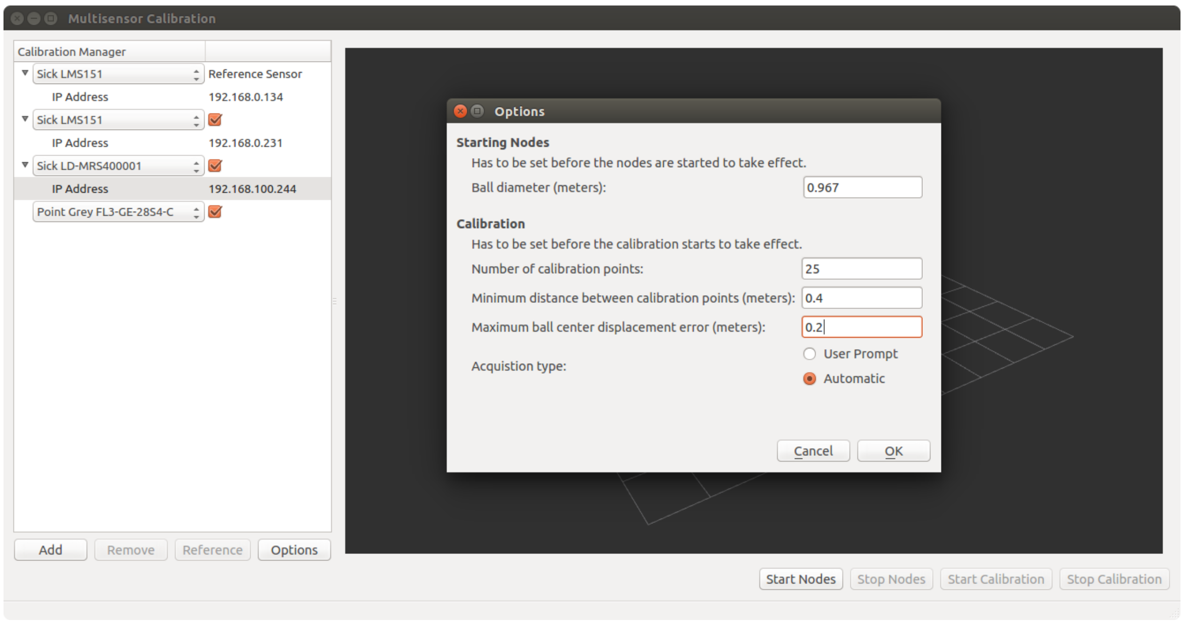
Calibration Manager	
Sick LMS151	Reference Sensor
IP Address	

Figure A.1: GUI main window after being launched.



Calibration Manager	
Sick LMS151	Reference Sensor
IP Address	192.168.0.134
Sick LMS151	✓
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.100.244
Point Grey FL3-GE-28S4-C	✓

Figure A.2: Adding sensors to be calibrated.



Sick LMS151	Reference Sensor
IP Address	192.168.0.134
Sick LMS151	✓
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.100.244
Point Grey FL3-GE-28S4-C	✓

Starting Nodes

Has to be set before the nodes are started to take effect.

Ball diameter (meters): 0.967

Calibration

Has to be set before the calibration starts to take effect.

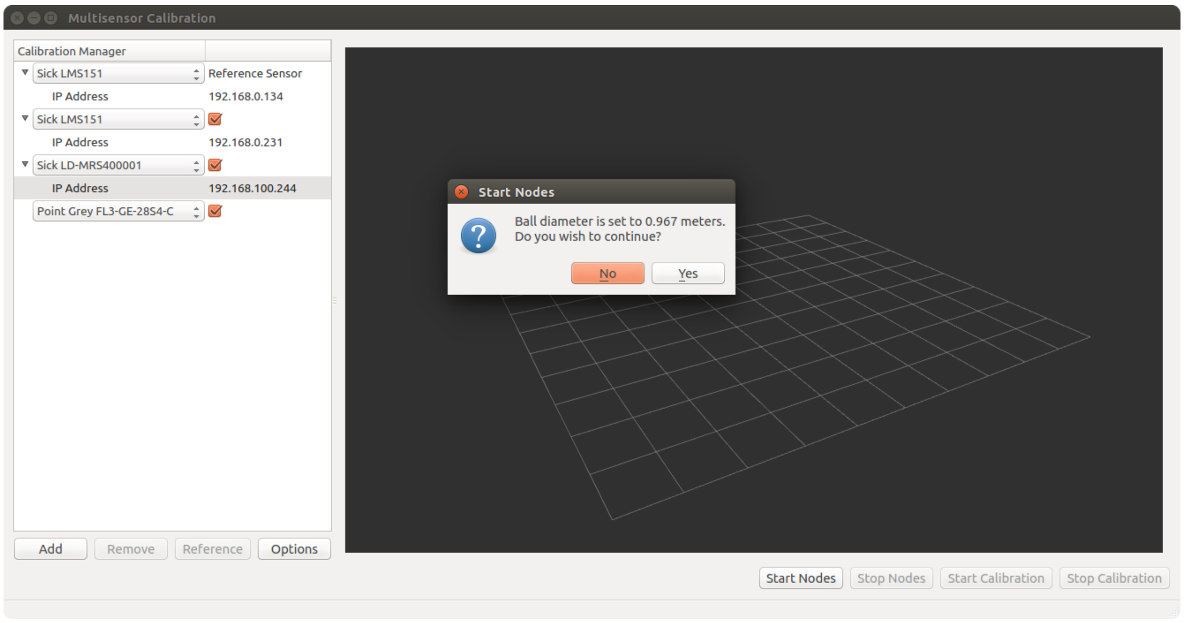
Number of calibration points: 25

Minimum distance between calibration points (meters): 0.4

Maximum ball center displacement error (meters): 0.2

Acquistion type: User Prompt [✓ Automatic]

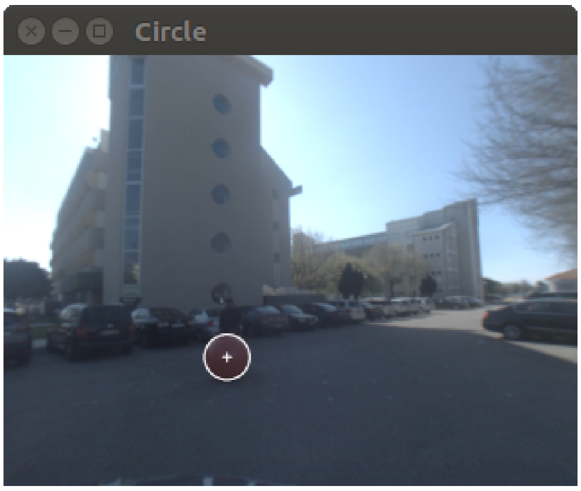
Figure A.3: Automatic calibration options.



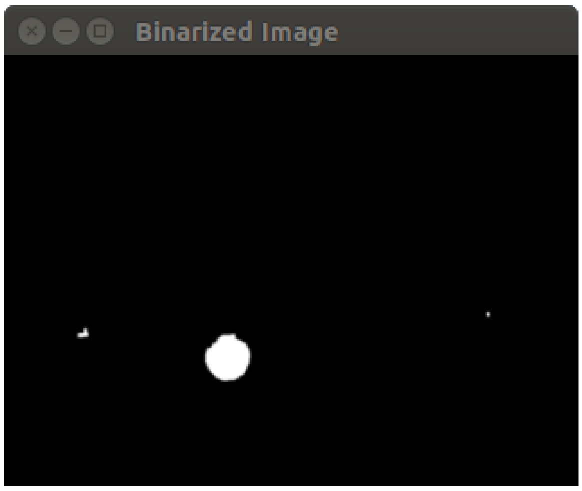
Sick LMS151	Reference Sensor
IP Address	192.168.0.134
Sick LMS151	✓
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.100.244
Point Grey FL3-GE-28S4-C	✓

Ball diameter is set to 0.967 meters.
Do you wish to continue?

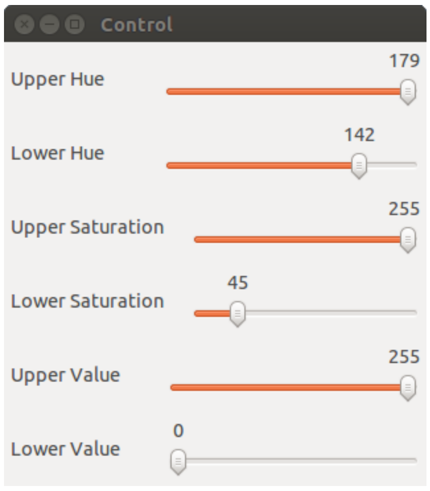
Figure A.4: Message box confirming ball diameter.



(a) Acquired image with detected ball.



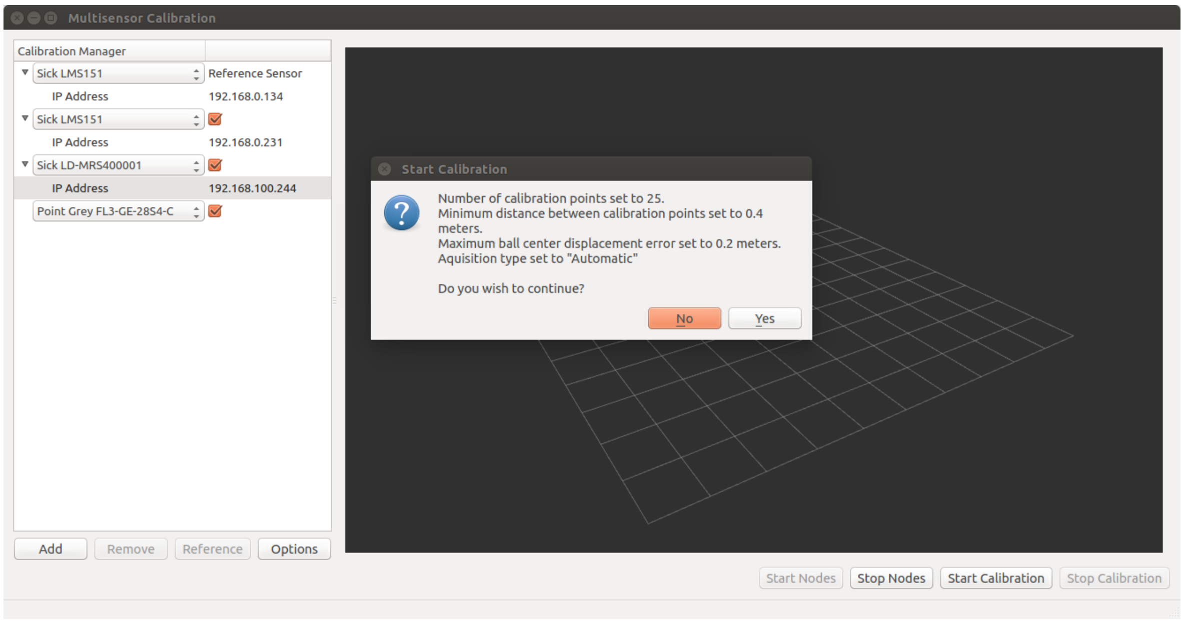
(b) Binary image after applying color thresholding to Figure A.5a with the color range specified in Figure A.5c.



Upper Hue 179
Lower Hue 142
Upper Saturation 255
Lower Saturation 45
Upper Value 255
Lower Value 0

(c) Window to control HSV color range.

Figure A.5: Windows launched when a Point Grey camera is used.



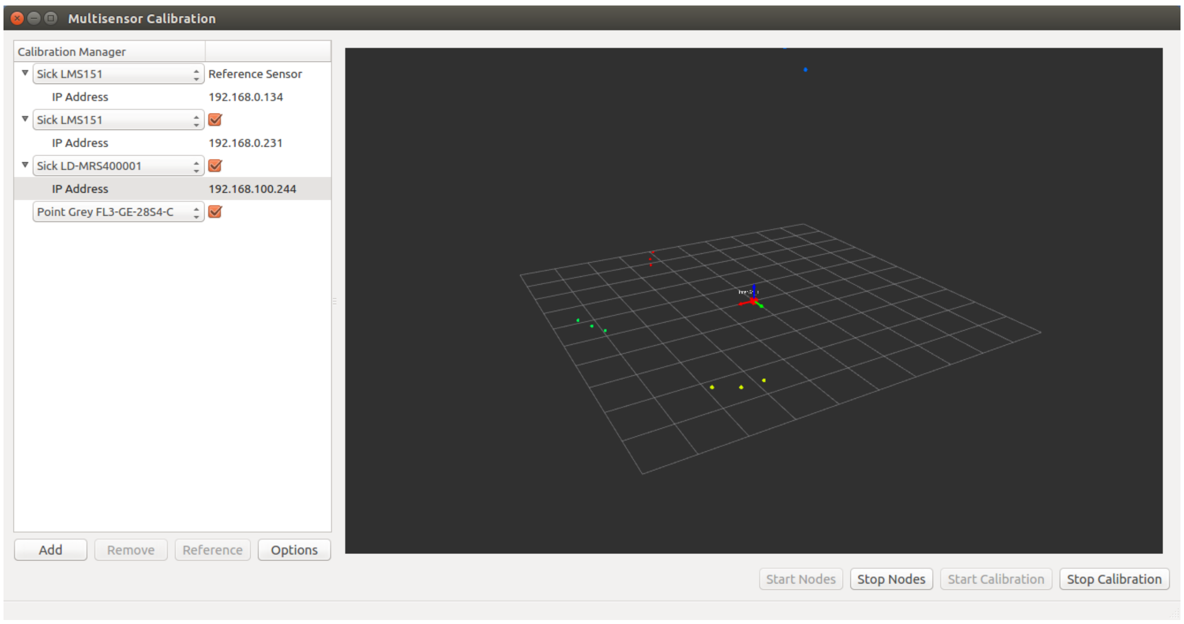
Sick LMS151 — Reference Sensor
IP Address 192.168.0.134
Sick LMS151
IP Address 192.168.0.231
Sick LD-MRS400001
IP Address 192.168.100.244
Point Grey FL3-GE-28S4-C

Start Calibration

Number of calibration points set to 25.
Minimum distance between calibration points set to 0.4 meters.
Maximum ball center displacement error set to 0.2 meters.
Aquisition type set to "Automatic"

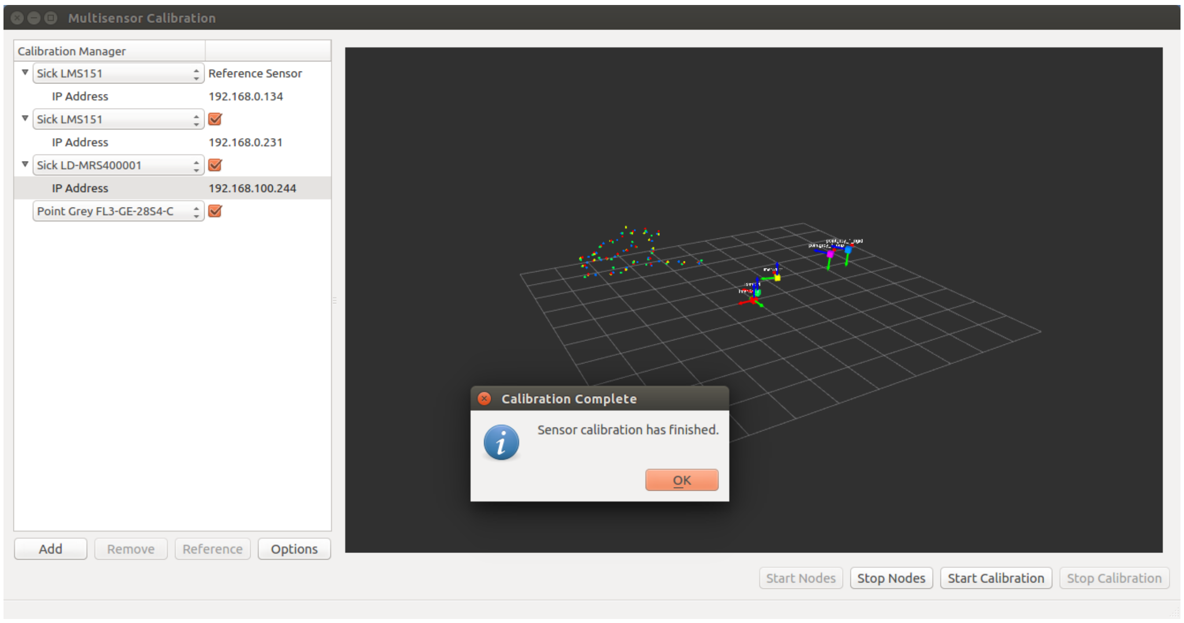
Do you wish to continue?

Figure A.6: Message box confirming the calibration options before starting the calibration process.



Sick LMS151 — Reference Sensor
IP Address 192.168.0.134
Sick LMS151
IP Address 192.168.0.231
Sick LD-MRS400001
IP Address 192.168.100.244
Point Grey FL3-GE-28S4-C

Figure A.7: Ball center points automatically acquired and drawn in the 3D visualizer.

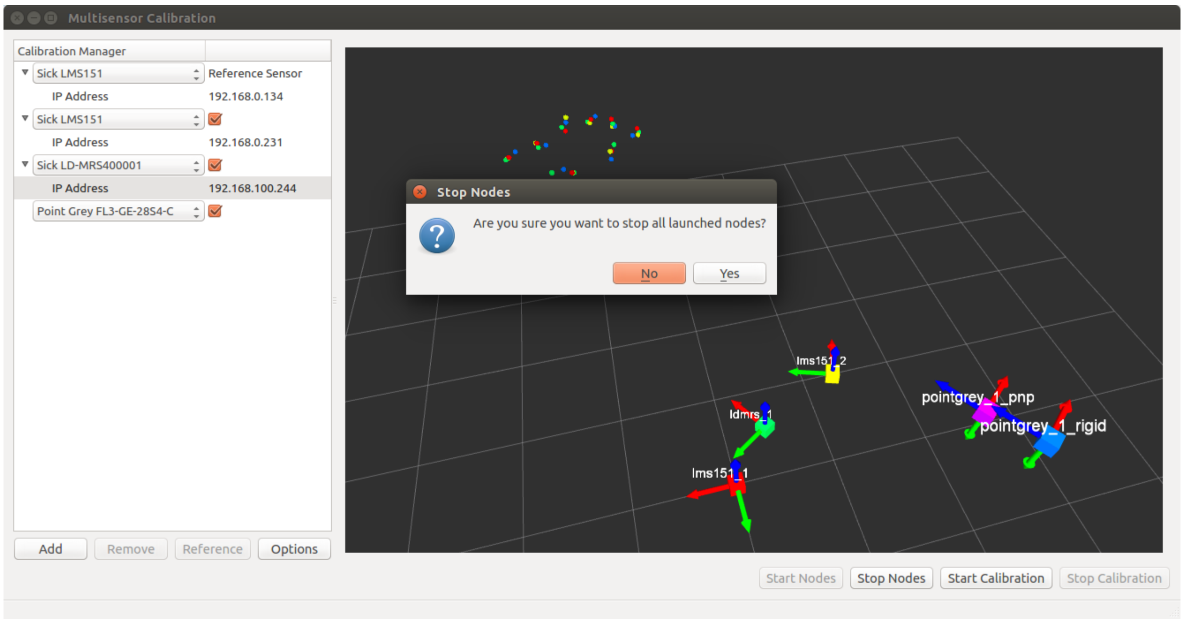


Calibration Manager	
Sick LMS151	Reference Sensor
IP Address	192.168.0.134
Sick LMS151	✓
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.100.244
Point Grey FL3-GE-28S4-C	✓

Calibration Complete

Sensor calibration has finished.

Figure A.8: Message box notifying that calibration has finished.

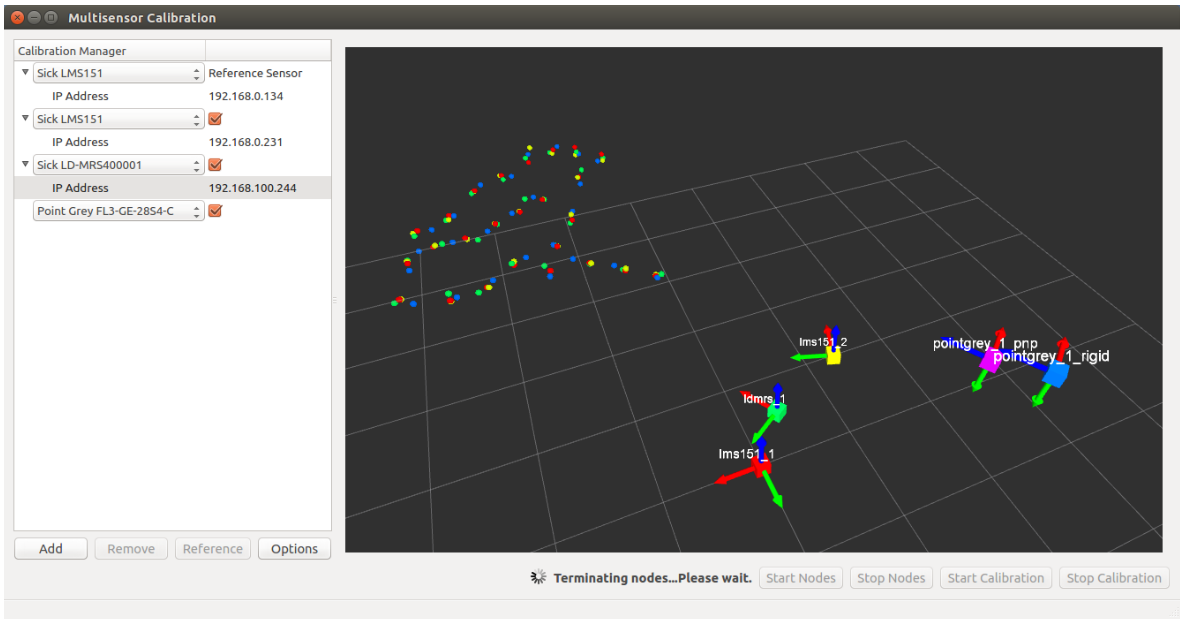


Calibration Manager	
Sick LMS151	Reference Sensor
IP Address	192.168.0.134
Sick LMS151	✓
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.100.244
Point Grey FL3-GE-28S4-C	✓

Stop Nodes

Are you sure you want to stop all launched nodes?

Figure A.9: Message box confirming if the user wants to stop every node.


Multisensor Calibration

Calibration Manager

Sick LMS151	Reference Sensor
IP Address	192.168.0.134
Sick LMS151	✓
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.100.244
Point Grey FL3-GE-28S4-C	✓

Terminating nodes...Please wait.

Figure A.10: Terminating nodes message.

A.3 User Prompt Calibration Mode

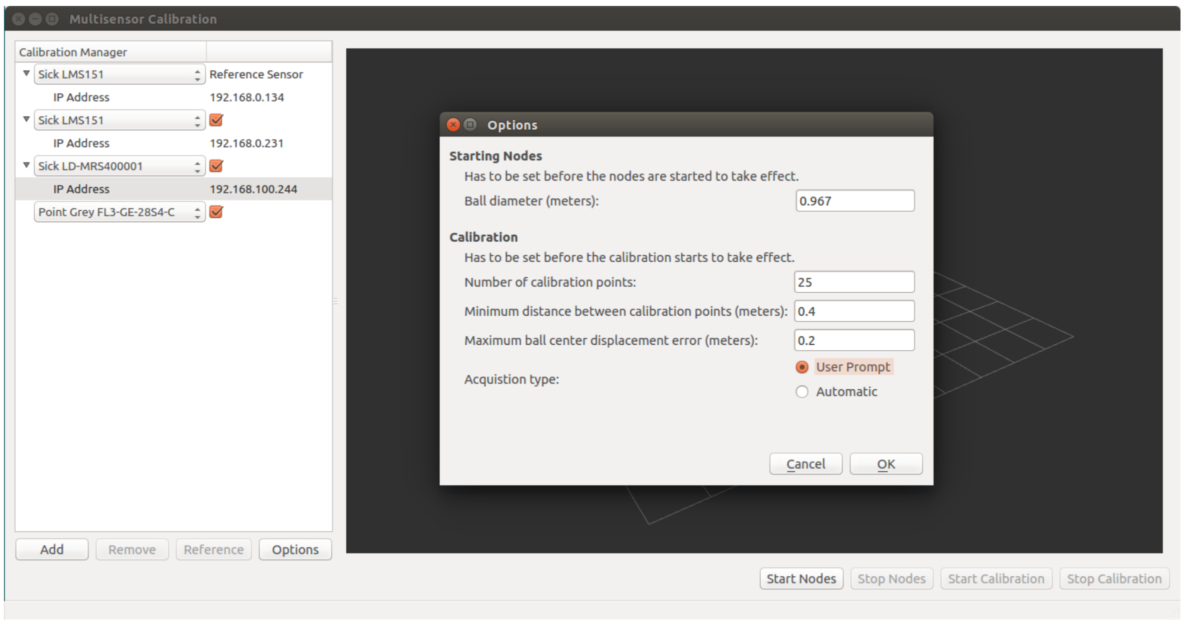
This section will detail how to use the calibration application in user prompt mode. The calibration procedure is very similar to the one presented in the previous section, except for the following steps:

6 As shown in Figure A.11, “Acquisition type” must be set to “User Prompt”.

and,

8.1 Ball center points are acquired only after the user presses “OK” on the message box shown in Figure A.12. When a new point is acquired, the message box is displayed again until the user wishes to acquire a new point.

For the remaining steps see Appendix A.2.

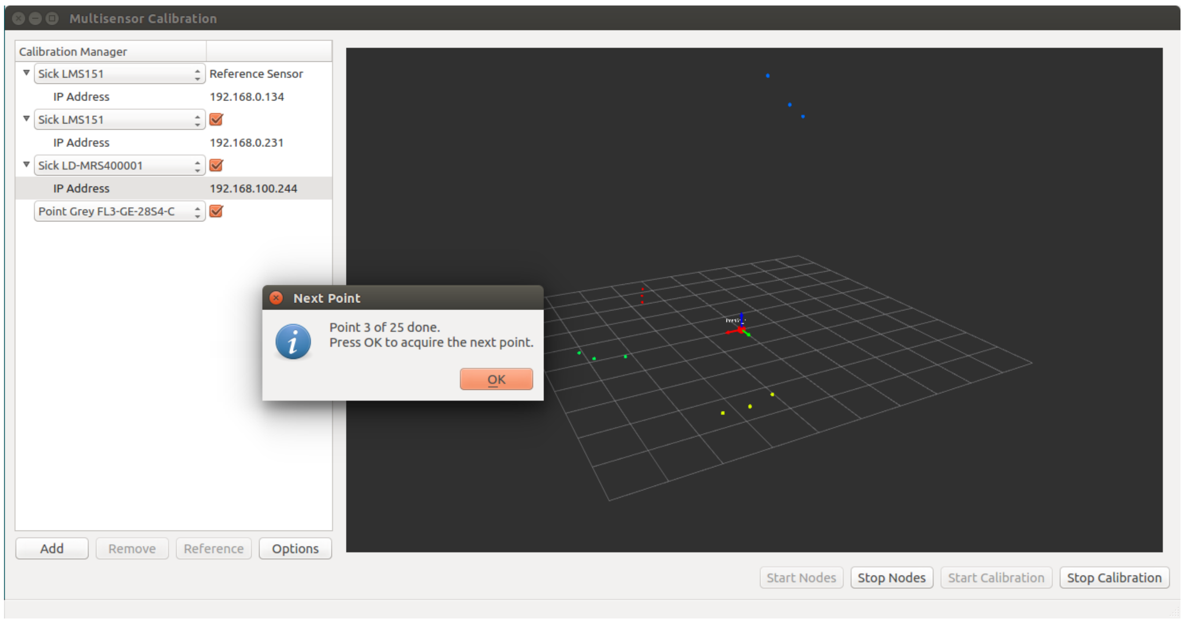


Sick LMS151	Reference Sensor
IP Address	192.168.0.134
Sick LMS151	✓
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.100.244
Point Grey FL3-GE-28S4-C	✓

Starting Nodes
Has to be set before the nodes are started to take effect.
Ball diameter (meters): 0.967

Calibration
Has to be set before the calibration starts to take effect.
Number of calibration points: 25
Minimum distance between calibration points (meters): 0.4
Maximum ball center displacement error (meters): 0.2
Acquistion type: [✓ User Prompt] Automatic

Figure A.11: Semi-automatic calibration options.



Sick LMS151	Reference Sensor
IP Address	192.168.0.134
Sick LMS151	✓
IP Address	192.168.0.231
Sick LD-MRS400001	✓
IP Address	192.168.100.244
Point Grey FL3-GE-28S4-C	✓

Point 3 of 25 done.
Press OK to acquire the next point.

Figure A.12: Message box prompt in “User Prompt” mode.