UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Technical University of Denmark

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TECHNICAL UNIVERSITY OF DENMARK

DEVELOPMENT AND DESIGN OF A ROBOTIC PERCEPTION SYSTEM: ENCLOSURE, MONO CAMERA INTRINSIC CALIBRATION, LIDAR-CAMERA EXTRINSIC CALIBRATION AND DATA-SET.

Bachelor's thesis

Ingeniería Electrónica y Automática Industrial

AUTHOR: Javier Casas Lorenzo

Tutor: Nalpantidis, Lazaros

Cotutor: Shmidt, Patrick

External cotutor: Hernández Luz, Carles

ACADEMIC YEAR: 2023

Danmarks Tekniske Universitet

DTU

DEVELOPMENT AND DESIGN OF A ROBOTIC PERCEPTION SYSTEM: ENCLOSURE, MONO CAMERA INTRINSIC CALIBRATION, LIDAR-CAMERA EXTRINSIC CALIBRATION AND DATA-SET.

13th of June 2023

Authors

Casas Lorenzo, Javier - s221817

Development and design of a robotic data collection platform

Bachelor Thesis
June, 2023

By
Javier Casas Lorenzo

Supervisor: Lazaros Nalpantidis
Co-supervisor: Patrick Schmidt

Approval

This thesis has been prepared over six months at the Department of Electrical and Photonics Engineering, at the Technical University of Denmark, DTU. The student form part of the Erasmus+ program, studying Industrial Electronic and Automation Bachelor at Universitat Politècnica de València, UPV.

It is assumed that the reader has a basic knowledge in the areas of robotics.

Javier Casas Lorenzo - s221817

Abstract

This project focuses on developing a system for collecting data from a Neuvition Titan M1-A solid-state LiDAR and a Bashler a2A1920 monocular camera. The system is designed to be deployed in a protective case in order to obtain data from construction zones as part of the RoBétArmé project.

The protective case for the sensors was designed using FreeCAD software and fabricated using a laser cutter with acrylic material. A calibration process has been executed to calculate its intrinsic parameters to solve lens distortion in the camera image.

To achieve accurate data fusion, it is essential to determine the relative position of each sensor. An extrinsic calibration method was used for this purpose.

The developed system was tested in a construction area in Copenhagen, yielding results that can be further analyzed and utilized for improvements. This project contributes to the advancement of coordinated systems for data collection in construction environments, enhancing the capabilities of robots in the RoBétArmé project.

The project was made in collaboration with other student Carlos Gascon Bononad in order to compare results. It was therefore divided into different objectives, the ones assesed in this document are: Enclosure desing, intrinsic calibration, Matlab LiDAR to camera extrinsic calibration and data fusion.

Keywords: LiDAR; Camera; Extrinsic Calibration; Intrinsic Calibration; Robot; Perception; Data Fusion; Dataset; 3D Design; ROS; Linux; Git; Enclosure.

Acknowledgements

First of all, I would like to thank to our supervisors Patrick Schmidt and Lazaros Nalpantidis for giving us the opportunity to work in this project with them and all the help that they have provided. It has been a great learning experience that I will keep in out minds for the rest of our life.

A nuestro grupo de amigos que nos ha hecho sentir un poco más cerca de España en los momentos de frío y de poca luz. Gracias *U2 plans* por todos esos billares, cenitas, Polonia y más planes inolvidables. Sere un fekas, pero es un instinto primario huir en busca de sol. Ya nos vendréis a ver, se os quiere.

Lo primero de todo dar gracias a mis padres sois la parte más fundamental de mi vida y sin vosotros no habría llegado hasta aquí. Gracias mama por ponerme las pilas esos días de selectividad y por poder hablar contigo de todo, y a papa por haberme inculcado esa pasión por hacer inventos y la mentalidad curiosa que me ha llevado hasta aquí. A mi rúcula favorita, que aunque ser pesado ha sido mi objetivo como hermano pequeño desde el día 1 te quiero un montón y estoy muy orgulloso lo que estás logrando. También agradecer a mis abuelas por todo el cariño que me han dado y la de veces que me han dado de la comida más rica del mundo. Al Edu que desde Pendeles y el descubrimiento de los bazookas sin retroceso has sido un amigo inseparable y aunque no nos veamos tanto sé que puedo contar contigo para lo que sea. Gracias también a Carlos y Alberto, habéis sido la mejor compañía que se puede pedir en Valencia y nunca me voy a olvidar de los pedazo de viajes y experiencias que hemos vivido y de todo el apoyo que he tenido con vosotros, os llevo en la patata.

Contents

1 Introduction

Robots are currently having a significant impact on our economy. They have been widely employed across various industries to enhance productivity, improve efficiency, and ensure safety. Initially, robots relied heavily on human intervention due to technological limitations. However, computing power and Artificial Intelligence advancements have made robots more autonomous, capable of independent movement and interaction with their surroundings.

Robots are commonly utilized in assembly line production, particularly in tasks where robotic arms are used to handle heavy objects with precision. A prominent example is the automotive industry, where robotic arms are employed to maneuver various components during vehicle manufacturing. However, the movement of traditional robotic arms is limited as they are fixed to a specific location. To address this limitation, the development of mobile robots has gained a lot of attention in recent years. These robots can navigate and transport objects within large areas, such as factory floors. They employ mapping and localization techniques to navigate and avoid obstacles. The combination of both types of robots has greatly improved production systems in the economy.

To function effectively, robots need to perceive and understand their surroundings. This project focuses on creating a system to gather visual data using sensors like LiDAR and cameras. The sensors will be protected and configured to be utilized by the robot, enabling the collection of data for further analysis. The data will be synchronized, and transformation parameters between different frames will be determined to extract consistent features of objects.

The collected data from this project will be used in collaboration with the RoBétArmé project, funded by the European Commission. The RoBétArmé project aims to develop autonomous robots specifically for the construction industry. Construction is a vital sector that contributes 6% to the global GDP, and its significance continues to grow, [Alm+16]. The autonomous robots developed in this project will make the maintenance and monitoring activities in the construction field simpler and more efficient, simplifying operations and improving overall infrastructure management.

2 Problem description

The aim of this thesis is to develop a data collection system for a robot, specifically for visualizing data from a camera and a LiDAR sensor mounted on a Clearpath Husky A200 UGV robot. To protect the sensors from external elements like weather, dust, and potential hazards in construction zones, they will be housed inside a protective casing. This casing will be mounted on a pan-tilt unit attached to the robot's arm.

(a) Robot from back

(b) Robot from lateral

Figure 2.1: Clearpath Husky A200 UGV

To ensure accurate data acquisition, it is necessary to obtain the intrinsic parameters of the camera. These parameters account for lens variations and are crucial for precise image analysis. Additionally, for effective data correlation between the camera and LiDAR sensor, an extrinsic calibration process will be conducted. This calibration aligns the sensors' perspectives to focus on the same field of view and establish a relationship between the data obtained from each sensor. Once the data collection system is set up and calibrated, it will be deployed in construction zones to gather relevant data. This collected data will be further analyzed and utilized by other stakeholders for various purposes and applications.

3 Background

This chapter aims to provide a concise overview of the important topics that are relevant to understanding the project. Each item will be described briefly, as the main purpose of this chapter is to establish a foundation for comprehending the subsequent chapters.

3.1 Software framework for Autonomous Systems

The Robot Operating System [ROS] is a development kit consisting of libraries and packages that facilitate the creation of robotic applications. It is an open-source platform designed to enable easy communication of data and workflows among different software components in a robot system. ROS not only handles data and package management but also facilitates communication with the sensors.

In ROS, the system is structured around *nodes*, which are individual pieces of code and functionalities grouped into packages. These nodes can subscribe to and publish *topics*, which are messages containing data. Additionally, ROS provides *services* that allow for the modification of package parameters. One of the advantages of ROS is its flexibility in programming languages, as it supports both C++ and Python.

3.2 Sensors

The sensors used in the project are a monocular camera and a Light Detection and Ranging (LiDAR). These sensors provide color (RGB) and a point-cloud respectively.

3.2.1 Monocular camera

The camera used for the project is a *Basler a2A1920-160ucBASIC* [Bas]. This camera provides a high resolution (1920 x 1200) with a frame rate up to 164 frames per second (fps).

The visible spectrum of the human eye is able to see light with a wavelength between 400 and 700nm. Depending on this wavelength, the color will change as seen in figure 3.1.

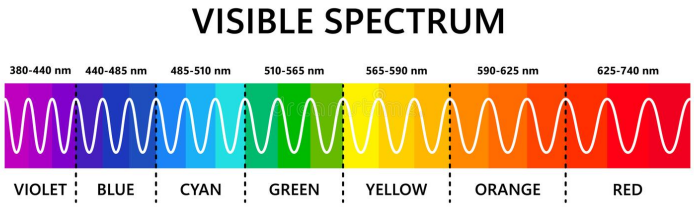


Figure 3.1: Visible light spectrum

The camera follows the same idea, being able to capture visible light (RGB cameras). In the case of the Basler camera, it not only depends on the wavelength range that is able to detect but also the intensity of the detection. Figure 3.2 is shown the sensitivity of the sensor used to capture the data.

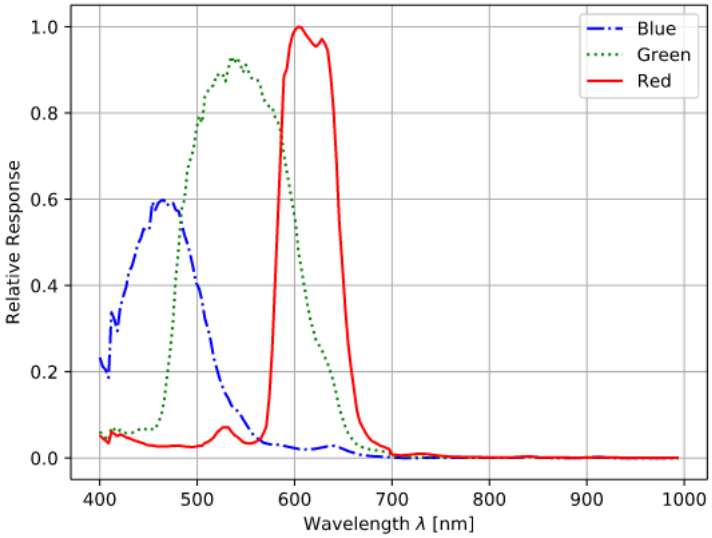
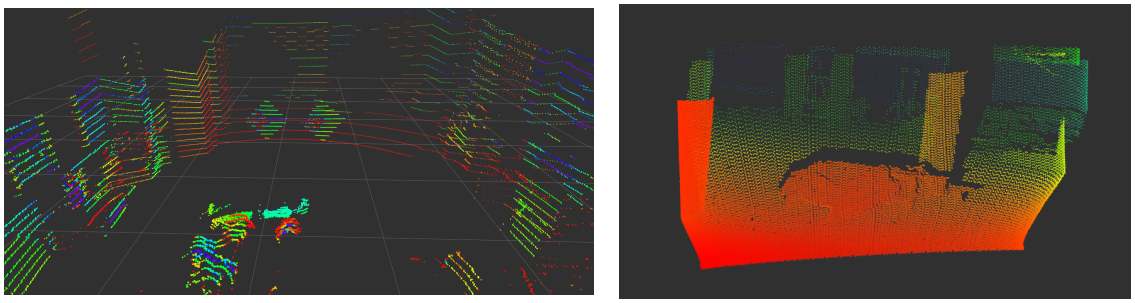


Figure 3.2: Basler camera response

3.2.2 LiDAR

LiDARs are sensors that uses lasers to detect distances on the surroundings. Some LiDARs use mechanical structures to capture the data in a 360º range, but the LiDAR of this project is a solid-state LiDAR. Neuvition Titan M1-A [Neu] uses a single laser beam to capture a forward image, like a camera. With this type of LiDAR, the captured data has higher resolution compared with the data of mechanical LiDARs. In figure 3.3 is shown a comparison between one of the most used mechanical LiDAR and the used in the project.



(a) Velodyne point-cloud

(b) Neuvition Titan M1-A point-cloud

Figure 3.3: Point-cloud comparison

Both distributors provide packages to connect the sensors with ROS, publishing topics with the data.

3.3 Intrinsic Parameters

In order to obtain correct projections of the 3D points in the space to the camera plane we need to account for the camera intrinsic parameters and correct the image in order to compensate for the wide angle lens distortion. This is relevant because when calculating the camera-LiDAR extrinsic parameters the results of the frames obtained would be far from the truth as the used images would be distorted. In an ideal pinhole camera we would only have to account for the camera matrix (K), that considers the optical center

and focal length of the camera. However the light rays passing by the camera lenses and hitting the sensor get bent and translated because of the lenses and there is a need to find several coefficients to model equations that compensate those rays distortions.

3.4 Extrinsic Parameters

LiDAR and camera sensors provide information about the surroundings that are crucial for autonomous systems. However, in order to effectively acquire and combine data from these sensors, it is necessary to establish a connection between what they are perceiving. This is achieved through an extrinsic calibration process that determines the transformation relationship between the sensors, enabling the fusion of their data. This transformation relationship can be defined as a 4x4 transformation matrix that combines rotation (yaw, pitch, roll) and translation (x,y,z) assuming that the multiplication of the matrix and a point in frame A returns the same point in frame B.

$$\begin{bmatrix} X_B \\ Y_B \\ Z_B \\ 1 \end{bmatrix} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_X \\ r_{2,1} & r_{2,2} & r_{2,3} & t_Y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_A \\ Y_A \\ Z_A \\ 1 \end{bmatrix} \tag{3.1}$$

4 Related work

This chapter will explain different methods for each calibration that can be used.

4.1 Intrinsic Calibration Method

4.1.1 Lens distortion background

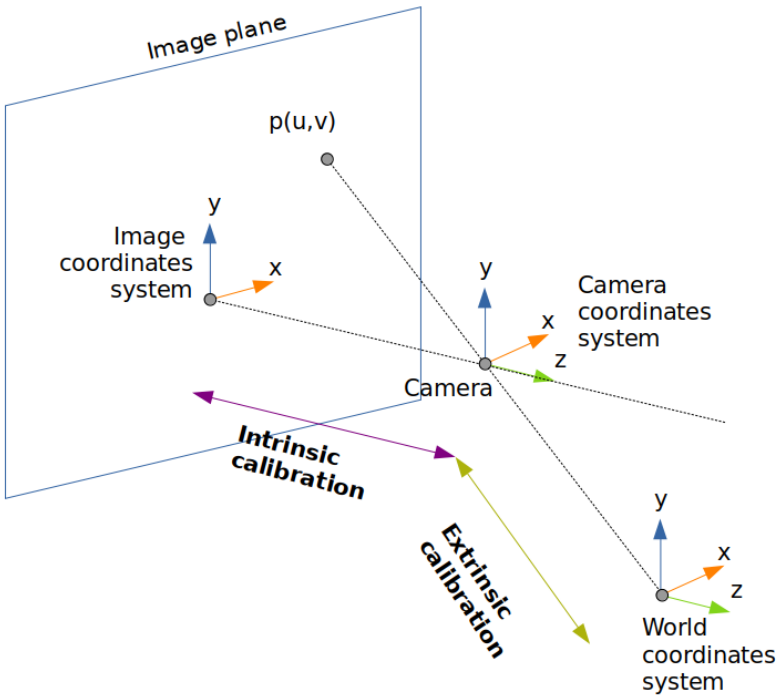


Figure 4.1: Intrinsics and Extrinsics

As explained in [Mata], in an ideal pinhole camera, there is no lens distortion as there is a really small aperture where light rays pass through and project on the image plane. Therefore the parameters that comprehend the intrinsics of a pinhole camera can be represented in the camera intrinsic matrix, K, which is:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

Where:
c_x, c_y - Optical center in pixels.
f_x, f_y - Focal length in pixels.
s - Skew coeficient.

We need to account for the lens distortion, which can be divided into two types:

- **Radial**: Radial distortion is produced by the bend of the light rays gradually increasing at the edges of the lens.

- **Tangential**: Tangential distortion happens when the image plane and the lens are not parallel.

Radial distortion can be modeled as:

$$x_{distorded} = x(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6)$$
$$y_{distorded} = y(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6)$$

Where:
x, y - Undistorted pixel locations which are non-dimensional parameters calculated by the translation of the point in pixels to the optical center and dividing it by the focal length in pixels.
k_1, k_2, k_3 - Radial distortion coefficients of the lens.
$r^2 = x^2 + y^2$

And tangential distortion can be modeled as:

$$x_{distorded} = x + [2 \cdot p_1 \cdot x \cdot y + p_2 \cdot (r^2 + 2 \cdot x^2)]$$
$$y_{distorded} = y + [p_1 \cdot (r^2 + 2 \cdot y^2) + 2 \cdot p_2 \cdot x \cdot y]$$

Where:
x, y - Undistorted pixel locations which are non-dimensional parameters calculated by the translation of the point in pixels to the optical center and dividing it by the focal length in pixels.
p_1, p_2 - Tangential distortion coefficients of the lens
$r^2 = x^2 + y^2$

4.1.2 Zhang's method

When calibrating cameras we need the distortion parameters to get the intrinsic matrix, and we need the intrinsic matrix to obtain the distortion parameter. The approach will be based on Zhang's method [Zha00] (that we could understood in depth thanks to the article [Ayy]) which computes the intrinsic parameters assuming the distortion parameters are 0 and then with the computed parameters we calculate the distortion parameters, then we do several iterations using the results obtained in the processes.
For this methodology, we will make use of the Rotation and translation from the checkerboard to the camera frame, as well as the homography. As we can observe in Figure 4.1 we can say that:

$$P_{cam} = [R|t] \cdot P_{obj} \; where : R = [r_1 \; r_2 \; r_3]$$

Combining them with the Intrinsics of the lens and the homography we can obtain the following two equations:

$$p_{img} = K[R|t]P_{obj} \;\; and \;\; p_{img} = \lambda H P_{obj}$$

P_{obj} has the shape of $(X, Y, Z, 1)^T$ as Z is always zero because the checkerboard target is an X-Y plane we can remove the third column of the rotation vector, as well as the Z coordinate of P_{obj}. Then:

$$p_{img} = K[r_1 \; r_2 \; t] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

So we can conclude that:

$$\lambda H = K[r_1\ r_2\ t\ ]$$

Now for every view of the chessboard, we will add 8 constraints from the homography matrix, 6 unknowns, due to the translation and rotation of the target and every view will share the same 4 unknowns from the intrinsic parameters. Therefore we will need at least two different poses of the checkerboard in order to obtain the intrinsic parameters. Calculating the homography for each view we can solve for the intrinsic parameters. H can be expressed as a vector of columns:

$$H = [h_1\ h_2\ h_3]$$

Then:

$$\lambda[h_1\ h_2\ h_3] = K[r_1\ r_2\ t]$$

Solving this equation:

$$\begin{bmatrix} r_1 \\ r_2 \\ t \end{bmatrix} = \begin{bmatrix} \lambda K^{-1}h_1 \\ \lambda K^{-1}h_2 \\ \lambda K^{-1}h_3 \end{bmatrix}$$

r_1 and r_2 are orthonormal meaning:

$$r_1^T r_2 = 0 \Rightarrow h_1 (K^{-1})^T K^{-1} h_2 = r_1^T r_2 = 0$$

$$and:$$

$$r_1^T r_1 = r_2^T r_2 \Rightarrow h_1 (K^{-1})^T K^{-1} h_1 = h_2 (K^{-1})^T K^{-1} h_2$$

We will define B, which is:

$$B = (K^{-1})^T = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix} = \begin{bmatrix} 1/f_x^2 & 0 & -c_x/f_x^2 \\ 0 & 1/f_y^2 & -c_y/f_y^2 \\ -c_x/f_x^2 & -c_y/f_y^2 & c_x/f_x^2 + c_y/f_y^2 + 1 \end{bmatrix}$$

Then every one of the equations obtained from r_1 and r_2 being orthonormal can be represented as:

$$h_i^T B h_j\ where\ 1 \leq i, j \leq 2$$

$$h_i^T B h_j = v_{ij}^T b = [h_{i1}h_{j1}\ h_{i1}h_{j2} + h_{i2}h_{j1}\ h_{i2}h_{j2}\ h_{i3}h_{j1} + h_{i1}h_{j3}\ h_{i3}h_{j2} + h_{i2}h_{j3}\ h_{i3}h_{j3}] \begin{bmatrix} B_{11} \\ B_{12} \\ B_{22} \\ B_{13} \\ B_{23} \\ B_{33} \end{bmatrix}$$

Then we need to find out B, and for that, we can solve the following system of linear equations with the help of the previous equations.

$$\begin{bmatrix} v_{12} \\ (v_{11} - v_{22})^T \end{bmatrix} b = 0$$

We can indeed see that the b vector is constant as it is a function of the camera. However, the v vector changes with each of the views from the checkerboard, what we will do is to compute the homography from each of the views and append it to the v_{12} and $(v_{11} - v_{22})^T$

of each view obtaining a long column vector. Then we will solve for b using singular value decomposition. Obtaining B we can get the camera matrix parameters:

$$c_y = (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2)$$

$$\lambda = B_{33} - (B_{13}^2 + c_y(B_{12}B_{13} - B_{11}B_{23}))/B_{11}$$

$$f_x = \sqrt{\lambda/B_{11}}$$

$$f_y = \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)}$$

$$c_x = -B_{13}f_x^2/\lambda$$

Then we can compute the extrinsic parameters for the rotation and the translation:

$$r_1 = \lambda K^{-1}h_1 \qquad r_2 = \lambda K^{-1}h_2 \qquad r_3 = r_1 \times r_2 \qquad t = \lambda K^{-1}h_3$$

Now we have the intrinsic matrix and the rotation and translation parameters. With this data we can proceed to compute the distortion parameters. The image-points that we obtained from the different photos of the checkerboard are the distorted points $(x_d, y_d)^T$ while we can compute the place were the points should lie on an ideal projection $(x_c, y_c)^T$ by using the equations we previously defined

$$p_{img} = K[R|t]P_{obj} \;\; and \;\; p_{img} = \lambda H P_{obj}$$

With both of the points using the equation for tangential and radial distortions:

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{bmatrix} x_d \\ y_d \end{bmatrix} + \begin{bmatrix} 2p_1 x_d y_d + p_2(r^2 + 2x_d^2) \\ 2p_2 x_d y_d + p_1(r^2 + 2y_d^2) \end{bmatrix}$$

Finally as this equations are linear we can simply obtain the solution for the coefficients by stacking more rows of equations using the points from different views and then using a least-squares method.

4.2 Extrinsic Calibration Method

4.2.1 Automatic Extrinsic Calibration of a Camera and a 3D LiDAR using Line and Plane Correspondences

This method [ZLK18] uses a checkerboard in order to obtain the extrinsic LiDAR to Camera calibration parameters given in the form of a rotation matrix and a translation vector. Its main advantages when compared to other methods is that the minimum number of poses of the calibration target required to obtain calibration results is only one. This is achieved by using the plane from the checkerboard as well as the 3D lines that outline it. The method is implemented in MATLAB with the MATLAB function estimateLiDARCameraTransform

A. Problem definition

Our aim is to obtain the transform from LiDAR to camera composed of the rotation matrix R_L^C and the translation vector t_L^C for this purpose this method makes use of the checkerboard plane π and its 4 borders L_{ij} where j is the number of the line {1,2,3,4}

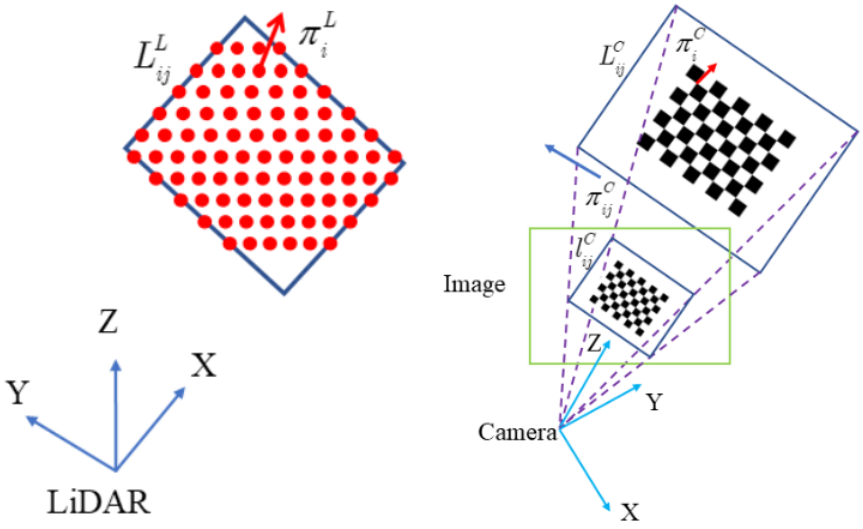


Figure 4.2: Planes seen from the LiDAR and the camera on a pose i and the lines that define them.

B. Automatic Feature Extraction

The approach for extracting the plane and lines from the LiDAR is the following: first of all, the algorithm uses RANSAC in order to find the checkerboard plane. Then the boundary of each of the scan lines is found by projecting the point into the checkerboard plane and the projecting that point on the scan-line as can be seen in figure 4.4. Finally we can apply RANSAC at the points of each of the lines in order to get read of any outliers. For

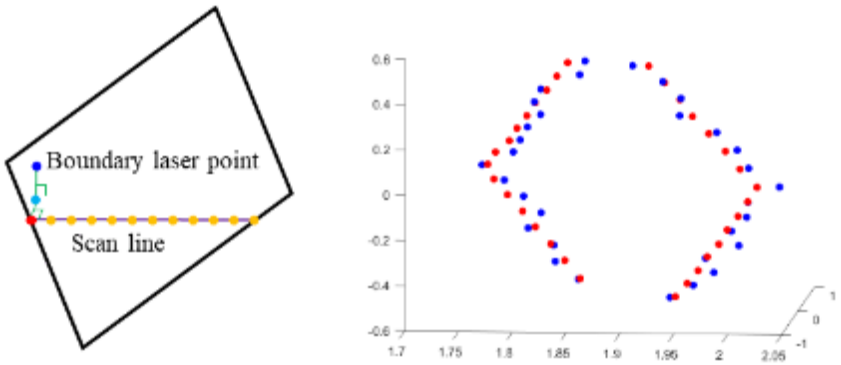


Figure 4.3: Denoise of the boundary points. In red: filtered points, blue: unfiltered points

the camera the approach is to obtain the images and detect the checkerboards. Then the planar parameters are obtained with the homography between the checkerboard and its image. Then the lines of the image are obtained by the LSD algorithm. We choose the four lines enclosing the detected checkerboard. Then obtain the 3D boundary lines with the intersection of the back-projected plane in 2D and the checkerboard plane.

C. Extrinsic calibration

The extrinsic calibration is complex and properly defined in the paper. As a summary:

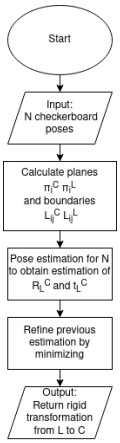


Figure 4.4: Algorithm Flowchart

5 Implementation

5.1 Design Case

There was a need for installing the different sensors of this project in an enclosure that was water-tight in order to protect the components from accidental hits, as well as to keep the inside dry. Also it was important for the enclosure to be rigid and to maintain the components within fixed frames from each-other. This chapter reflects the process undergone since the first material selection steps to the final mounting of the sensors in the mobile robot.

5.1.1 Choice of the material

The first part assessed was the selection of the materials [CPI] to use in the enclosure, as this would determine the fabrication method and the design that would be made. It is often seen in different real life examples such as action cameras that polycarbonate is used because of its good durability properties as well as it is less likely to crack under stress when compared to other transparent plastics. Acrylic on the other hand is more rigid, wich is good in order to avoid the displacement between the frames of the LiDAR and the mono-camera, it has also a better light transmittance, of 92% which means the mono camera sensor will receive more light. Acrylic is also generally cheaper than polycarbonate, easy to mechanize and easier to glue together, so it was the final material chosen for the enclosure. It was also proposed to use acrylic for the front part of the box and wood on the other sides, however this option was soon discarded due to the porosity of the wood not making a great candidate for a water tight enclosure.

5.1.2 Selection of a parametric design software

Then a method for fabricating the enclosure had to be selected. Laser cutting was chosen as it was easily available for thesis projects at DTU Skylab. It is fast for prototyping, and easy to use. Provided that we would use a laser cutter, the design would have to be provided in a vector format to the cutting software, with the most popular being .svg or .dxf. The design could have been made entirely into 6 different 2D pieces, however for better visualization of all of the pieces assembled together we decided that the best methodology to follow would be to make a 3D design first and then convert each of the pieces into .svg format.

There are 2 main options of software available for use that were analysed in order to make the 3d model:

FreeCAD	Fusion 360
Parametric modeling.	Parametric Modeling.
Steep learning curve, extensive Wiki community.	Ease of use and quality tutorials.
Open Source.	Proprietary Software.
Free.	Free licenses only for educators and students.
Lots of available macros and environments,can create your own with python.	Apps and plugins available, many of them are not free.
Available in Windows, MacOs and Linux distributions	Available only in Windows and MacOs

FreeCAD 0.20.2 was finally selected [Fre] as it was available for our Ubuntu OS, it has a big open-source community and lots of macros and environments can be installed in order to ease the process of the design of our case.

Due to being completely new to the process of CAD design and to FreeCAD, a series of videos and some research in its Wiki were followed in order to get to know the basics:

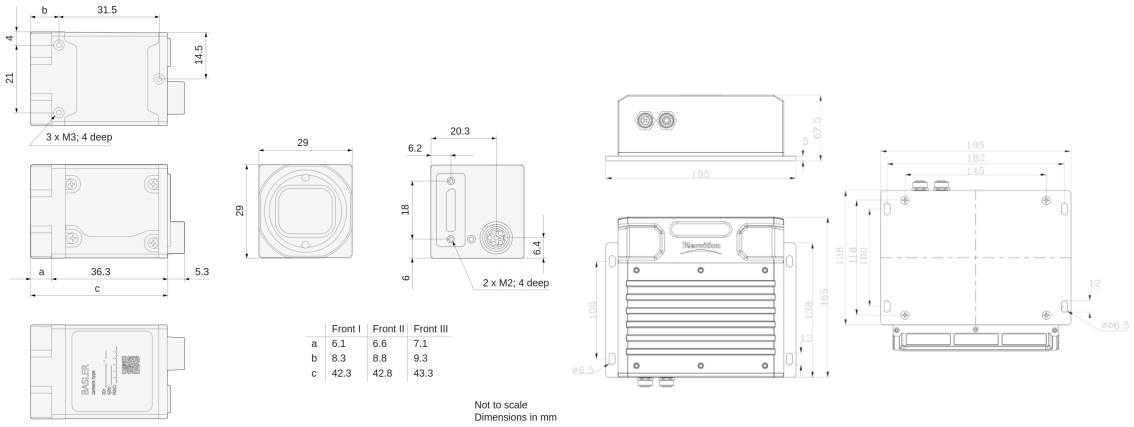
- FreeCAD tutorial (YouTube playlist)
- Finger-Slotted box
- FreeCAD Wiki

5.1.3 Design of the enclosure

The main idea for the design of the enclosure was to make an acrylic cuboid that holds the sensors When designing the cuboid a finger-slotted method for the joints was chosen for two main reasons: Mainly because it would provide more rigidity when glued together when compared to a plain corner joint, but also as it would facilitate the aligning of all of the pieces for the gluing stage.

Sketching

In order to make the finger-slot box, the first attempt was to make all of the pieces of the box by hand. For this we would need to reach into the data-sheets of each of the components in order to get their dimensions. Figure 5.7. Then a sketch for the bottom part of the box was made in order to estimate the measurements of the final box considering the thickness of the acrylic material as well as the dimensions of the mono-camera and LiDAR. The planned dimensions for the box were: **200*280*96 mm**.



(a) Camera Sketch (b) LiDAR Sketch

Figure 5.1: Dimensions of the sensors

Box design
The methodology for creating a box in FreeCAD was the following:

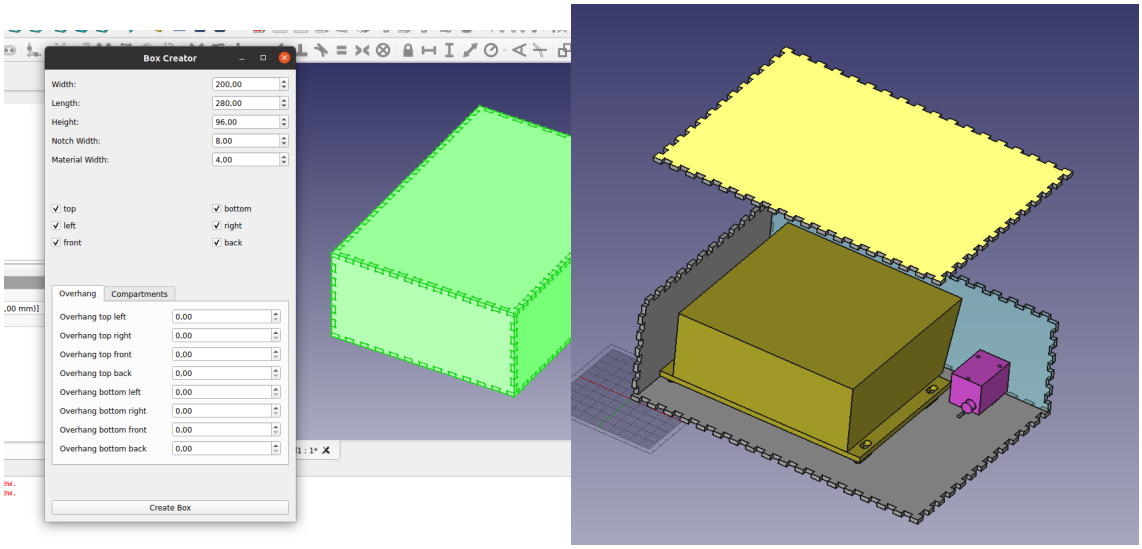
1. Create a body.
2. Make a sketch inside that body with the shape of one of the faces of the box.
3. Fully constraint that sketch.
4. Pad the sketch in order to make a solid part.
5. Make a new sketch with the places of the different holes.
6. Fully constrain the sketch.
7. Use the pocket feature in order to remove material from the object.
8. Repeat for each of the asymmetric faces of the cuboid shaped enclosure.
9. Translate all of the created bodies in order to form the box.

Designing and constraining the pockets for all of the finger slot joints on all of the different faces was a really tedious process. It was made better by making use of parametric design with the spreadsheet workspace and specifying the different relevant dimensions of the box such as the thickness of the walls the size of the box or the shape of the finger-joint pattern. Then with the variables specified in the spreadsheet we could use two really interesting tools to make the process way faster.

- **PartDesign LinearPattern:** This tool can make a linear pattern of a feature(s) in an specified direction, with a selected number of occurrences and between a selected length.
- **Sketcher Symmetry:** This tool can copy a geometry and paste it across a selected line or a sketch axis.

With this tools and the variables recorded in our spreadsheet we can make the design of our part parametric, which means than instead of assigning numerical values to the Linear pattern or to the constraints of the different geometries we can just change the box values

in the spreadsheet and the whole design will generate according to this parameters. This is really useful for future projects as we could change the size of our enclosure according to our needs f.e. when adding more sensors.
Later in the process we discovered doing bibliography research that a macro does this exact same task. Macros in FreeCAD can be programmed in python, added easily with the Add-On manager and can objectively speed-up the design process. For this case *BoxCreator.FCMacro* was used as can be appreciated in Figure 5.2a.



(a) BoxCreator Macro (b) Final Design

Figure 5.2: FreeCAD design process

After making the box, the only thing needed was to add the pockets for the mounting of the camera and the LiDAR sensors, the next steps were followed:

1. Create a new body.
2. Copy the bottom part of the box.
3. Add it as a base feature to the newly created body.
4. Create a new sketch.
5. Constraint the sketch.
6. Pocket the sketch.

As an addition the LiDAR and the mono-camera were modeled based on their CAD dimensions in order to check their fitting before finally cutting the acrylic on the laser cutter, see Figure 5.2b.
It is worth mentioning that the design of both of the components was a great learning experience, where some other FreeCAD tools were used such as Datum planes or translation of the components.

Making of the SVG file
For the laser cutting of our 6 different pieces we need to arrange them in such way that they fit in an acrylic board and we also need to provide the CNC cutting program a 2D

file in a format that can be interpreted in order to send the correct instructions to the laser cutter.
Luckily there is a open-source workspace for FreeCAD, *LCInterlocking*, that is designed for changing our pieces from 3D to 2D SVG. The procedure consists on selecting the bodies that we want and click export on the tools bar, this will create a 2D vector shape of the object. After doing the same step with the rest of the objects we can place them in a way that they fit in our laser cutter. Then select File/Export and export it as an SVG format.

5.1.4 Assembling the case

The case was Laser cut in the DTU Skylab facilities by first importing the 2D file into CorelDRAW 2021, then selecting the outline and turning it red rgb(255,0,0) wich is the key color for cutting. Finally it was sent by clicking print to the laser cutter, being careful to select the adequate material (3mm acrylic).

The components were attached firmly to the enclosure base using M3 and M6 bolts, nuts and washers for the camera and the LiDAR respectively. An H frame of 2020 and 4040 T-Slot aluminum that was previously made to attach sensors to the mobile data collection platform was used to add more rigidity to the base. Figure 5.3 The rest of the case wasn't assembled as the deployment of the sensors was done in an indoor controlled environment.

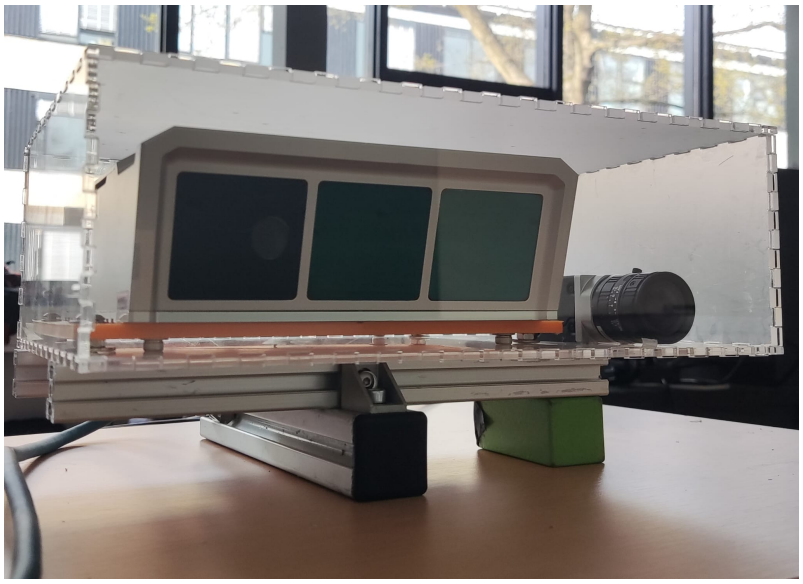


Figure 5.3: Final enclosure

5.2 Intrinsic Calibration

The method used for the intrinsic calibration of the camera was the Zhang's method described in 4.1.2 which has been widely used since its release in 2000 because of its reliability and the simple calibration target used (checkerboard). Several libraries such as OpenCV [Ope] or MATLAB offer a simple implementation for the intrinsic calibration of a camera.

5.2.1 Intrinsic calibration using Python and OpenCV

First of all we define the object-point vector which is comprised of the position of the checkerboard corners in 3D. As we saw before, we consider the checkerboard to be in

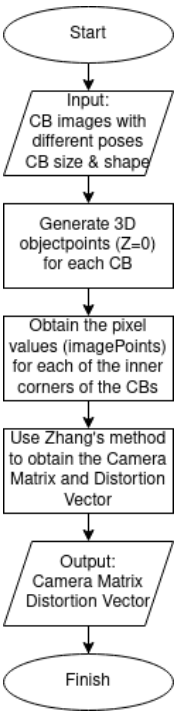


Figure 5.4: Calibration implementation flowchart

the X-Y plane so the Z component of those points will be zero. Then it will have the following format:

$$(0,0,0),(1,0,0),(2,0,0),....,(x_{corners},y_{corners},0)$$

Then a loop goes trough each of the images and extracts the 2D position of the corners of the checkerboard in the image with the help of OpenCV *findChessboardCorners*.

```
ret, corners = cv.findChessboardCorners(img, (x_corners,y_corners), None)
```

If the return of the function is True it means it was able to find the corners and we will append them to an array of vectors (image-points), then do the same for the object-point vector appending it to an object-points array and move on to the next image of the loop.

We should end up with two arrays of vectors of equal size, one for the image-points and one for the object-points. With this information we can do the implementation in OpenCV of Zhang's method by means of OpenCV *calibrateCamera*

```
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, img.
    shape[::-1], None, None)
```

The most relevant parameters obtained with this function are *mtx*: the camera matrix and *dist*: the distortion vector. This parameters will be later used to undistort the image.

5.2.2 ROS camera_calibration

ROS camera_calibration [Jam] is a python based node that helps with the calibration of a monocular or stereo camera. For its installation:

```
$ rosdep install camera_calibration
```

Then initialize the camera:

```
$ roslaunch pylon_camera pylon_camera_node.launch
```

Open a new terminal and start the camera_calibration node.

```
$ rosrun camera_calibration cameracalibrator.py --size 7x5 --square 0.08
    image:=/pylon_camera/image_raw camera:=/pylon_camera
```

We can observe that we need to specify some parameters, first of all the CB *size* which is the number of inner corners. Then the *square* size in meters and finally the *image* and *camera* topics.

An interface like the one in Figure 5.5 will appear. The app will automatically start taking

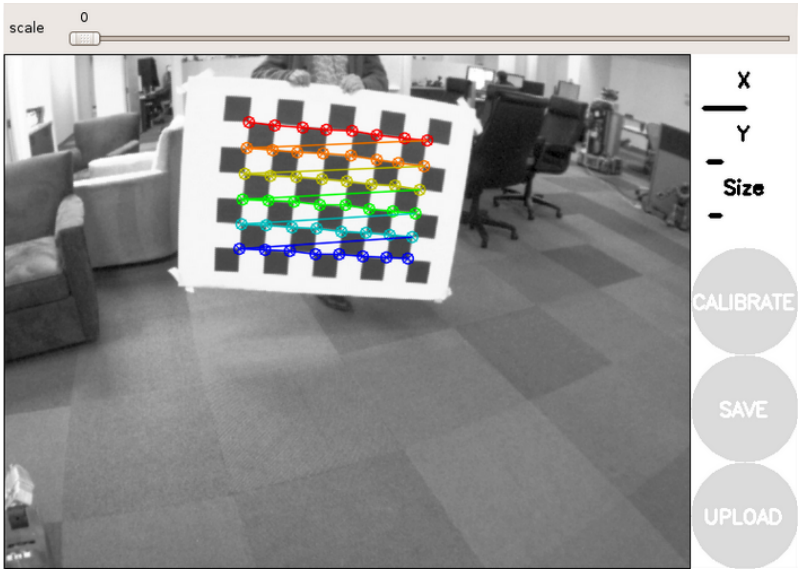


Figure 5.5: Camera_calibration app

samples of the checkerboard until it has enough data to calibrate the camera. We then have to move the checkerboard to different positions and poses until the bars on the top right corner for X Y and Size fill up. Once that is completed the calibrate button can be pressed, obtaining then an image of the camera calibrated. If the results are satisfactory(having straight edges on the image) click the commit button which will save the parameters into a YAML file. We can then specify the url of that file in our camera *default.yaml* file in order to get a rectified image from the camera.

5.2.3 MATLAB

For the intrinsic calibration using MATLAB the cameraCalibrator app was used. Figure 5.6 First of all type cameraCalibrator on the MATLAB prompt. This will open the application. Then we can add images from the checkerboard in different positions from 10-20 should

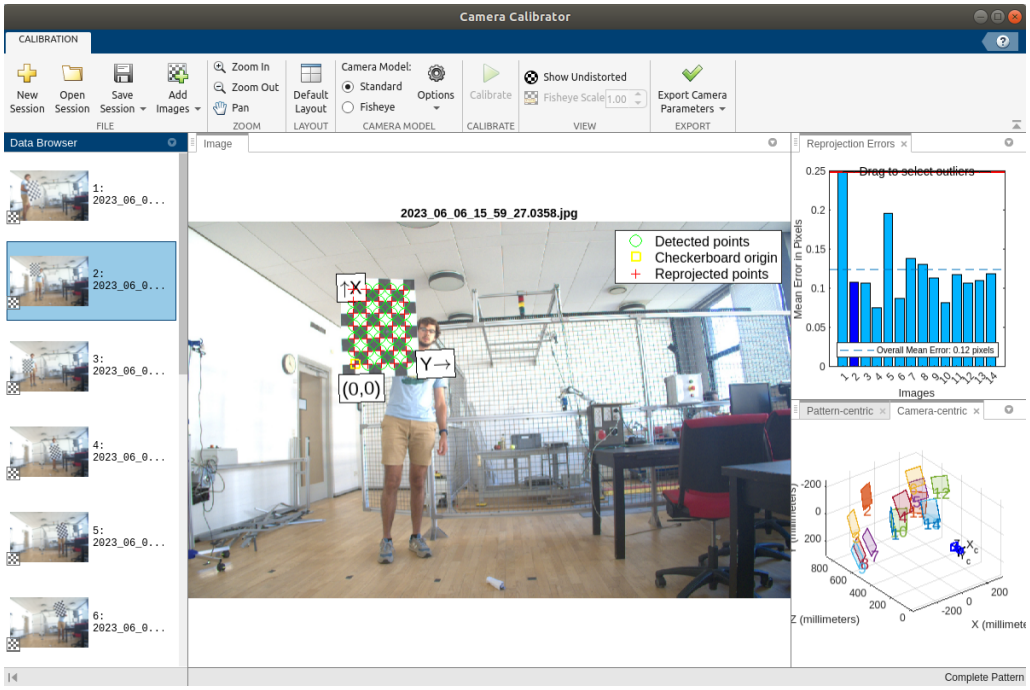


Figure 5.6: cameraCalibration app

be enough to get good calibration results. We should also input the checkerboard square size in millimeters.

After that we can click on the calibrate button. When the calibration is done we can visualize the rectified image, as well as a comparison of the originally detected points and the re-projection when the camera is rectified.

The app also plots the mean error of the calibration images in pixels which help us evaluate the quality of the result. If we're satisfied with the results we can export them as a .mat file. The calibration results will be the camera matrix, as well as two coefficients for the radial distortion and another two for the tangential distortion.

5.3 Extrinsic Calibration

5.3.1 Automatic Extrinsic Calibration of a Camera and a 3D LiDAR using Line and Plane Correspondences

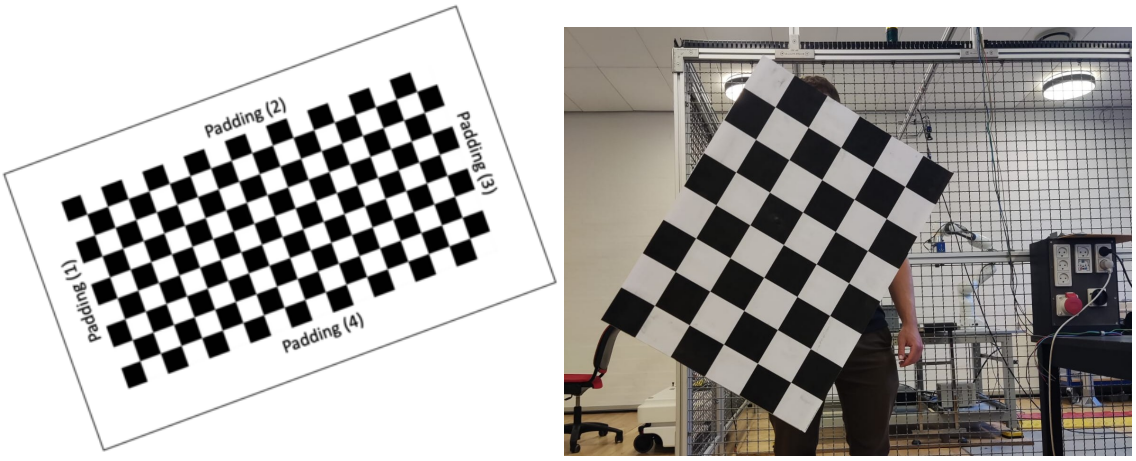
As it was mentioned in section 4.2.1 the method could be implemented in MATLAB with some previous processing [Matb].

Making of the target

For the calibration to work we need a rigid rectangular planar surface in which to stick the checkerboard. There is no problem for the rigid surface to be larger than the checkerboard as long as we define the padding later in our program [Matc]. In this case an 8x6 checkerboard with 80mm squares was used and pasted onto a 3mm birch plywood plank. See figure 5.7b.

Also the pointcloud data obtained by the neuvition LiDAR showed a variance from the measurements between the white and black squares, where the white squares seemed to be closer to the LiDAR than the black ones, this can be due to white being more reflective and the LiDAR having trouble when getting accurate measurements from the low reflectance of the checkerboard. The consequences of this problem as well as an alter-

native solution will be addressed later.



(a) Calibration target paddings (b) Used checkerboard

Figure 5.7: Calibration targets

Gathering of the calibration data
In order to implement the calibration, different poses of the checkerboard must be captured with both the LiDAR and the camera. It is important then, as the target is going to be displaced, that data is picked at the same time from camera and LiDAR. For this purpose we will gather synchronised data.

The images are saved in a folder as .jpg while the point-clouds are saved in the .pcd format. Then we can manually choose the best samples of images and their corresponding point-cloud as data to feed to our program.

Implementation of the calibration
There are four main steps figure 5.8 to follow in order to obtain the LiDAR to camera transform:

First the intrinsic parameters of the camera are computed with the help of Zhang's method as explained in 4.1.2. For the shake of simplicity the intrinsic calibration is made with MATLAB, with the help of *estimateCameraParameters* as the parameters obtained will be in the proper format for its use in the next functions.

Then we will estimate the 3D checkerboard points from the images using the *estimateCheckerboardCorners3d* function [Matd]. It is possible to obtain them because it is known the checkerboard size its padding and the intrinsics of the camera.

Later the plane segment in the point-clouds is detected, we can specify the minimum distance for clustering, as well as a ROI in the form of [xmin, xmax, ymin, ymax, zmin, zmax] in order to only consider certain points of the point-cloud for the computing of the plane. Then with the help of the MATLAB function *detectRectangularPlanePoints* we can detect the rectangular planes of the point-cloud.

Finally, with the previously obtained 3D points from the camera and LiDAR data, we can apply 4.2.1 that can be implemeted with the help of *estimateLiDARCameraTransform*. This will return the transform from LiDAR to Camera in the form of a rotation matrix and a translation vector.

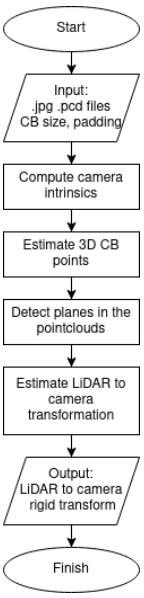


Figure 5.8: Implementation diagram

LiDAR Camera Calibrator app

This app, figure 5.9, allows for fast LiDAR-camera extrinsic calibration. It follows the same logic mentioned above, but offers better visualization of the data and makes the selection of the ROI easier. To use the app we first need to install the LiDAR package for MATLAB, as well as the dependent packages. Then simply start the app by writing LiDARCameraCalibrator [Mate] in the MATLAB prompt.

First select Import>Import Data and choose the images and the point-clouds folders to be imported. Then the checkerboard dimensions as well as the padding are inserted.

The app will accept the data for calibration if it finds the checkerboard planes in both the image and the point-cloud.

It was found that it was pretty determinant for a data pair to be accepted, to modify the region of interest in order to contain mainly the checkerboard plane. It is also possible to modify other parameters such as the cluster threshold or the dimension tolerance that can have an effect in finding planar regions within the data given by the LiDAR.

After enough accepted pairs are obtained click on the calibrate button. Then the calibration parameters can be exported and used in further applications.

The app also provides 3 bar plots to evaluate the calibration errors, one for translation, another one for the angle error and a final one that displays the re-projection error for each of the pairs. It also provides a final colored point-cloud showing the fusion of the results.

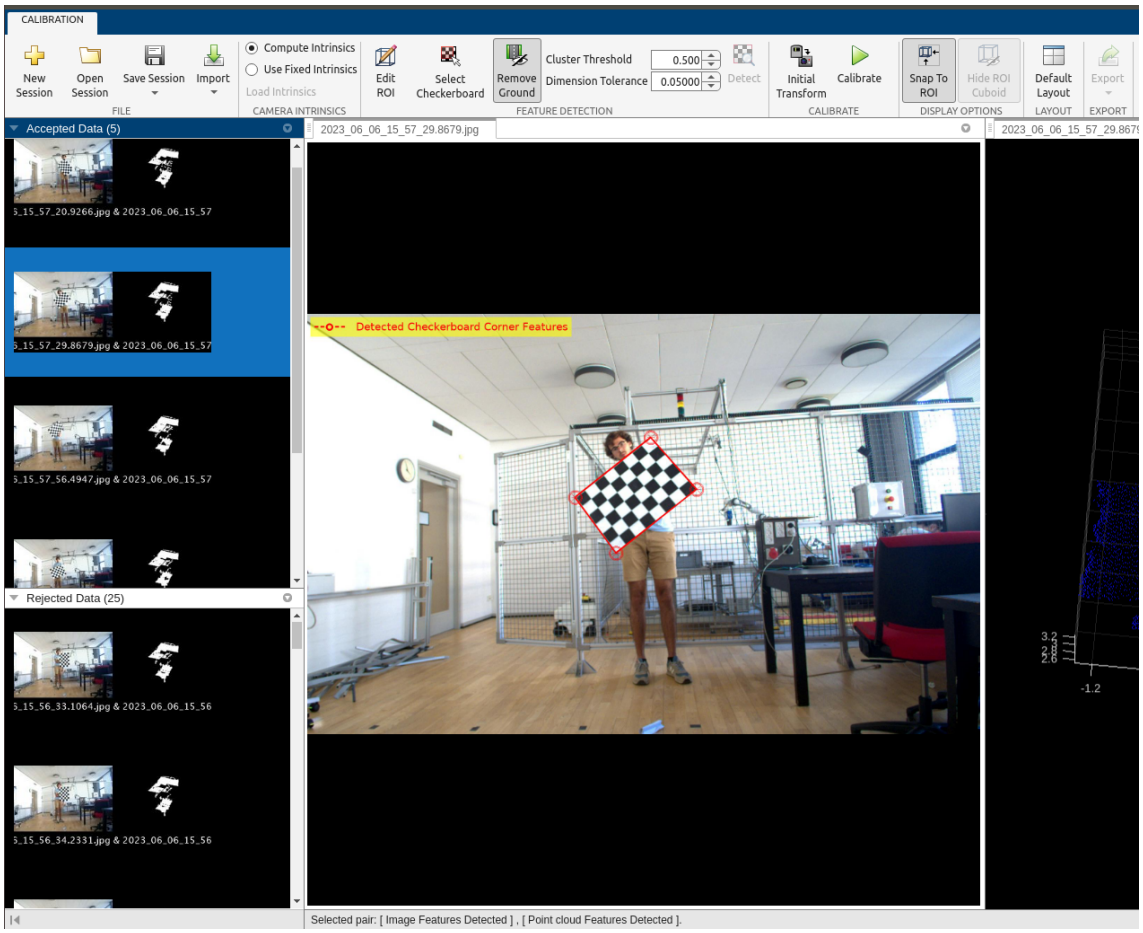


Figure 5.9: LiDAR Camera Calibrator app interface

6 Final results

This chapter summarizes the results obtained during the project, mainly the intrinsic and extrinsic calibration as well as the data collected during the testing.

6.1 Intrinsic calibration results

The calibration of the camera was made with the three different methods mentioned in section 5.2.

6.1.1 Showcase of the different results

Python & OpenCV

$$K = \begin{bmatrix} 1177.56786 & 0 & 906.282368 \\ 0 & 1176.97430 & 543.142150 \\ 0 & 0 & 1 \end{bmatrix}$$

Radial coefficients

$$(k_1, k_2, k_3) = (-0.250571918, 0.123913316, -0.0369855407)$$

Tangential coefficients

$$(p_1, p_2) = (-4.27097213e-04, -1.53169632e-04)$$

ROS camera_calibration

$$K = \begin{bmatrix} 1179.97894 & 0 & 926.64154 \\ 0 & 1181.48564 & 555.55689 \\ 0 & 0 & 1 \end{bmatrix}$$

Radial coefficients

$$(k_1, k_2, k_3) = (-0.223102, 0.065771, 0.000000)$$

Tangential coefficients

$$(p_1, p_2) = (0.000210, 0.000224)$$

MATLAB cameraCalibrator

$$K = \begin{bmatrix} 1194.15720 & 0 & 918.98551 \\ 0 & 1194.56706 & 552.89826 \\ 0 & 0 & 1 \end{bmatrix}$$

Radial coefficients

$$(k_1, k_2, k_3) = (-0.231815542, 0.0839317475, 0)$$

Tangential coefficients

$$(p_1, p_2) = (0.000210, 0.000224)$$

As we can see in Figure 6.2 the straight lines in real life, such as columns or the edges of the calibration target, are straightened, which means all of the intrinsic calibration methods seem to be working properly within some degree of error. The focal length and the focal

center obtained only varies in a few pixels between the methods which can be due to different calibration targets being used for each method. Overall we can conclude that all of the methods worked properly and that the errors in the calibration can be attributed to the lack of sampling of different poses of the checkerboard, irregularities in the calibration pattern or not having a perfectly planar surface in which to stick the checkerboard.

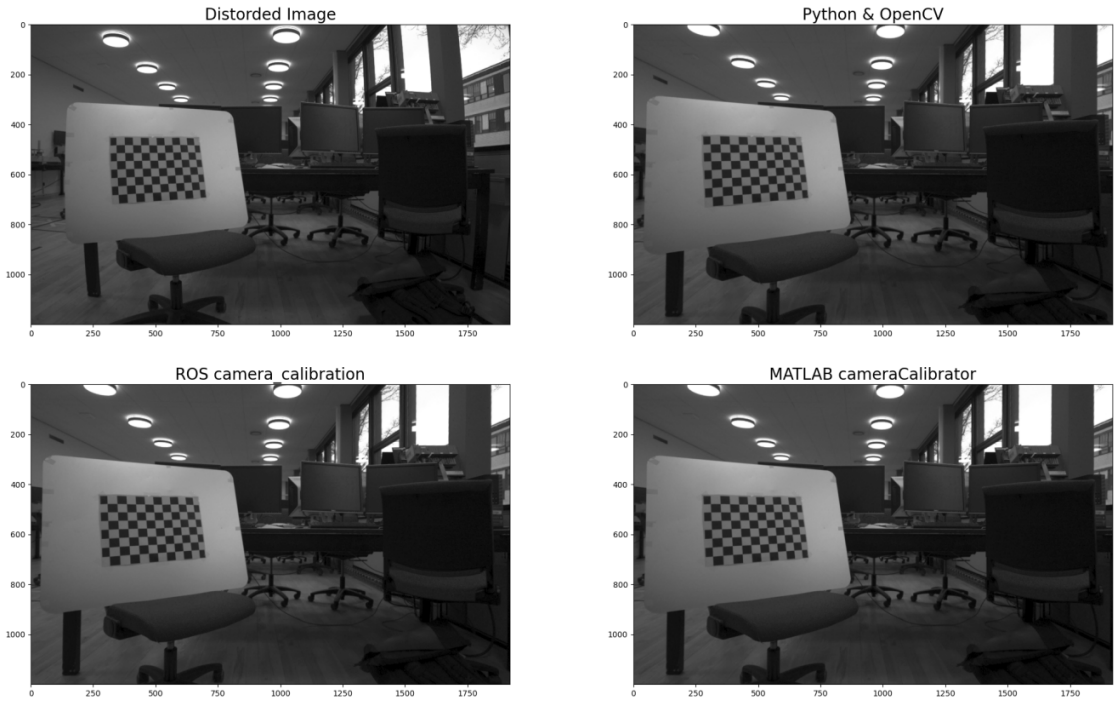


Figure 6.1: Comparison between the RAW image and the different calibration methods

6.2 Extrinsic calibration results

The results of the extrinsic calibration show different values for each method. The transform results obtained with the method 5.3.1 developed in this thesis:

$$\begin{bmatrix} -0.9997 & -0.0180 & -0.0158 & 0.0690 \\ 0.0181 & -0.9998 & -0.0057 & -0.0145 \\ -0.0156 & -0.0060 & 0.9999 & -0.1501 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

And the transform provided by the velo2cam method [Bel+] method implemented in another bachelor thesis developed in the context of the same project is:

$$\begin{bmatrix} -0.9991 & -0.0018 & -0.0429 & 0.112 \\ 0.0030 & -0.9997 & -0.0261 & 0.0129 \\ -0.0428 & -0.0262 & 0.9987 & 0.057 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Both transformation matrix are applied to the LiDAR frame to obtain the points from this sensor frame to the camera frame. Changing to the Euler axis, the translation and rotation are:

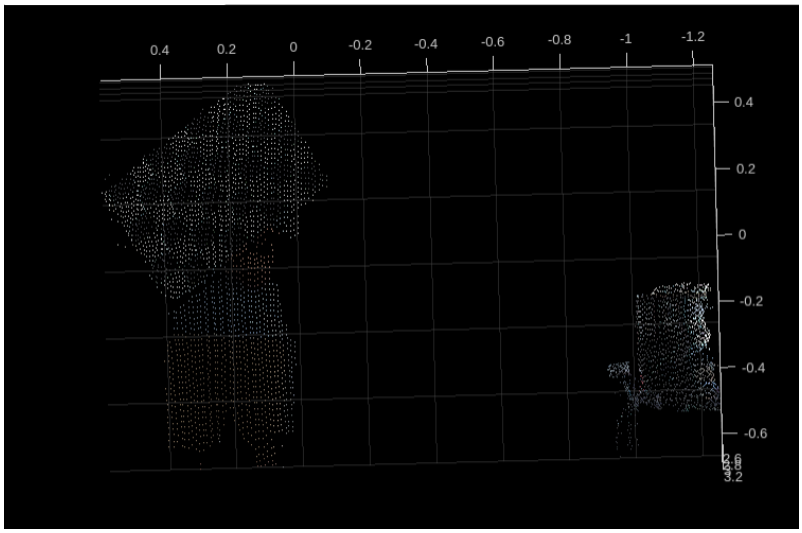


Figure 6.2: Combination of the data with method 5.4.2

Parameters	Matlab	Velo2cam
Translation (X)	0.0690	0.112
Translation (Y)	-0.0145	0.0129
Translation (Z)	-0.1501	0.057
Yaw	178.96	179.83
Pitch	0.89	2.4536
Roll	-0.3438	-1.5073

Table 6.1: Translation and Rotation Vectors

The translation values are in meters, and the angles are in degrees. As it is observable, the rotation values are practically the same while the translation values are much more different. For a better comparison, it has been observed in *rviz*:

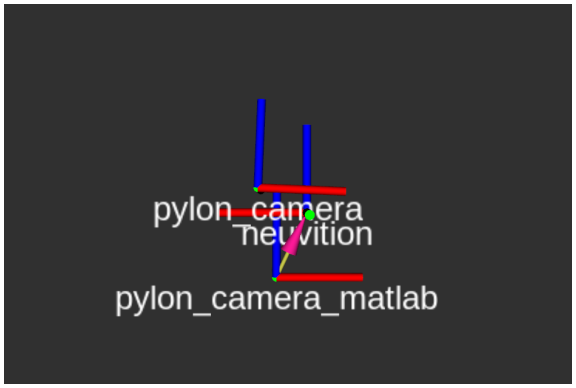


Figure 6.3: Top view of the frames

Figure 6.3 shows the frames in a top view, being the blue axis as the Z-axis of the sensors, which is where they are facing. The LiDAR frame is *neuvition* while the *pylon_camera* is the camera frame obtained by the velo2cam method and the *pylon_camera_matlab* frame by the MATLAB method. The rotation of both frames is similar. But the translation is a bigger problem, MATLAB method has moved the camera frame backwards in the Z-axis.

The method velo2cam obtains the frame with more similarities, generating the camera moved forward in X-axis, as planned during the enclosure design to avoid the camera perceiving the LiDAR lateral.

Due to the calibration results, the method used for extracting data is the velo2cam method, explained in section **??**.

6.2.1 Errors

The errors from the MATLAB extrinsic calibration are given when implementing the function *estimateLidarCameraTransform*. We can observe three different errors, translation, rotation and reprojection. Errors are plotted for each image for better interpretation. Figure 6.4

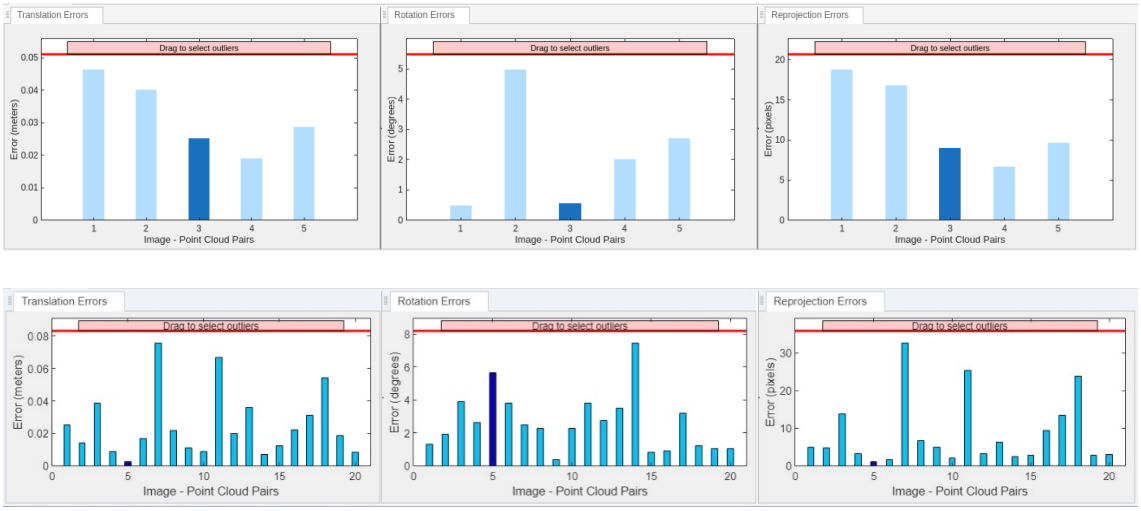
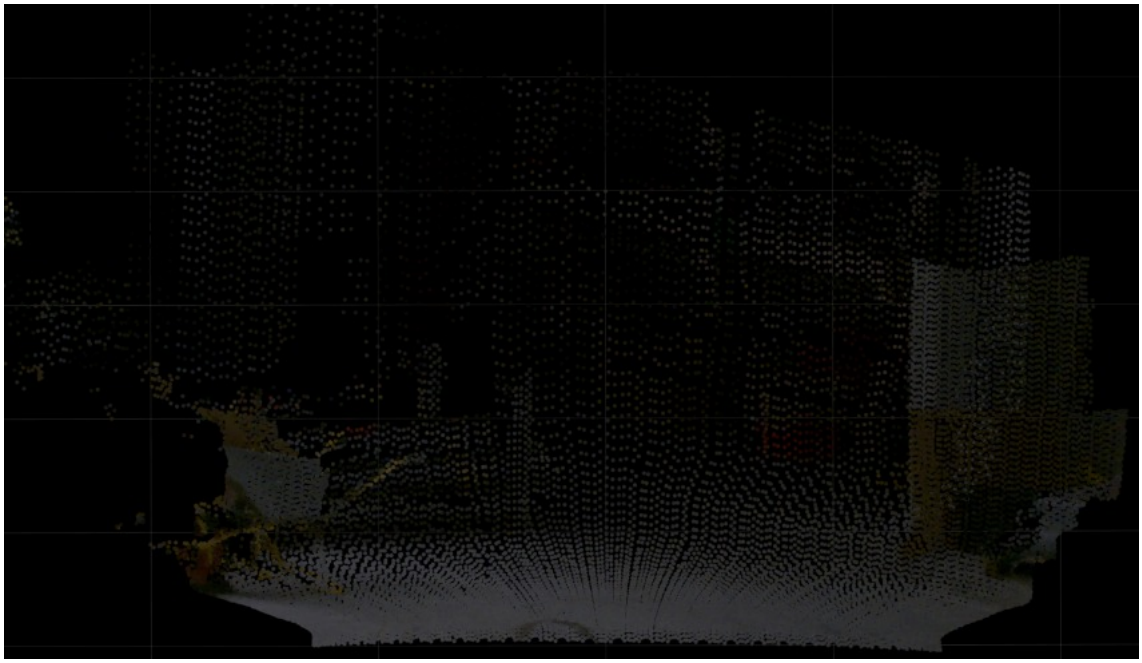


Figure 6.4: Errors for 5 and 20 samples respectively

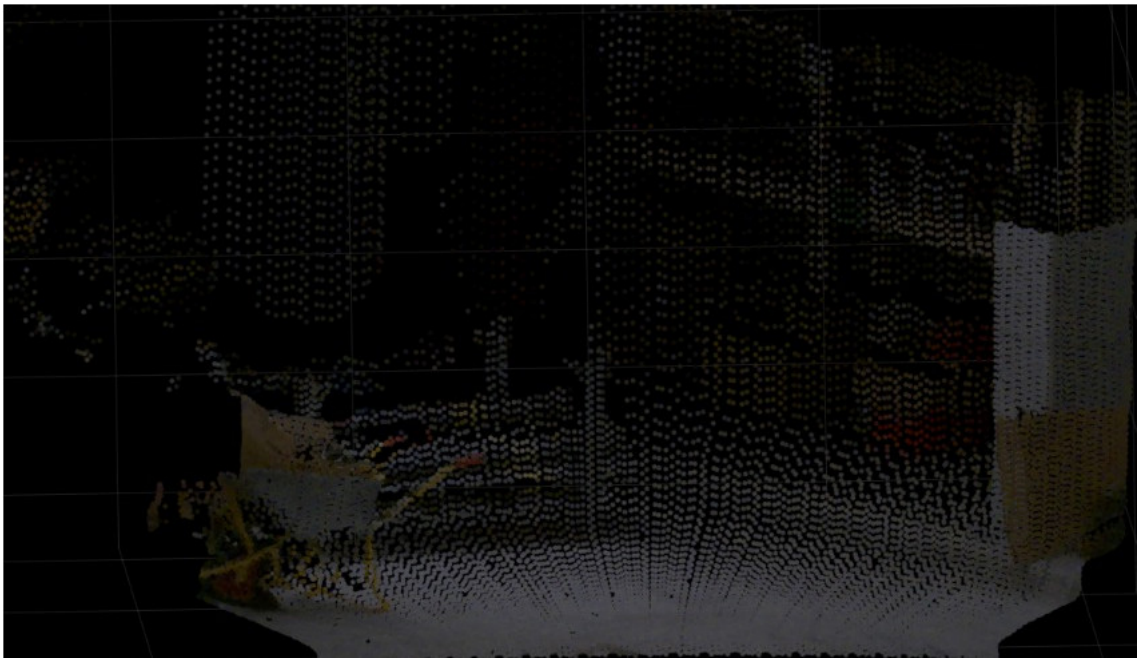
The same 100 samples were used for the calibration and in both cases the ROI was modified to the checkerboards region in the point-cloud. In the case of the 20 samples the Cluster Threshold and the Dimension Tolerance were set to lower and higher values respectively in order to obtain more acceptance of the samples. We can see that the reprojection error is lower in the 20 samples case, with the exception of some outliers, but it hasn't improved significantly. This may be due to some images being blurry due to the movement of the checkerboard or to some of the planes in the point-cloud found with this more permissive parameters not fitting the real position of the plane perfectly.

6.2.2 Results compare

After collecting data, both extrinsic calibrations were compared, combining images and point-clouds. It has been created a 3D model with an image and a *.pcd* file, to compare them.



(a) Results MATLAB calibration



(b) Results velo2cam calibration

Figure 6.5: Comparision of 3D model

Comparing figure 6.5, the low difference between both calibration models is observable. The error among the pixels is around 10 pixels, as obtained in the MATLAB calibration. MATLAB method get better results for far objects while the velo2cam method for near objects.

6.3 Data obtained

To obtain former data, the system was tested inside a construction area. this data has given us the necessary feedback to improve the calibration and fix failures in the system. Also, the data will be used to create a dataset for the RoBétArmé project.

The data has been divided into the different objects observed. The test started by collecting data about broken ceiling parts.

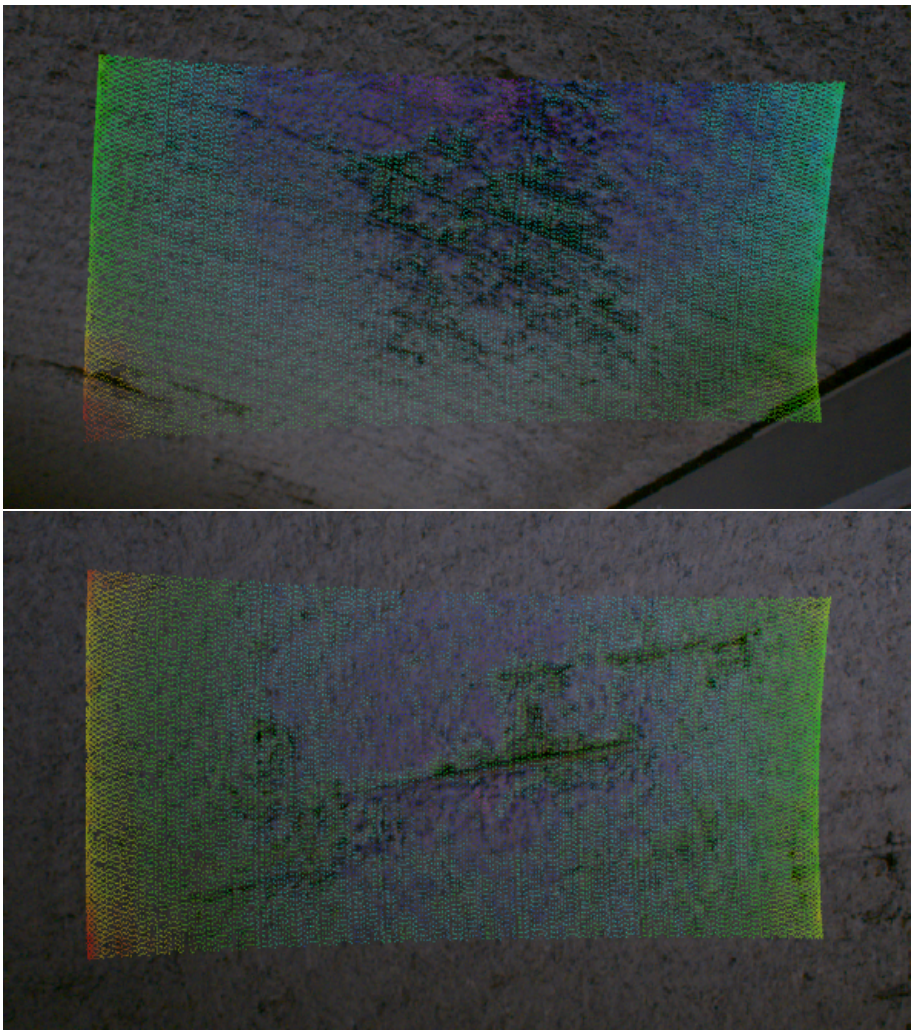


Figure 6.6: Degraded concrete ceiling showing the inner rebar

Figure 6.6 is observable in the construction pipeline. LiDAR data is in intensity mode to differentiate better the objects rather than the distance. Both data overlap each other with a minimum difference. It is observable that the shape of the pipeline structure is green, while the surrounding ceiling is blue. The image is unclear and does not provide enough information about the calibration. The next type of data collected is the structure of the construction.

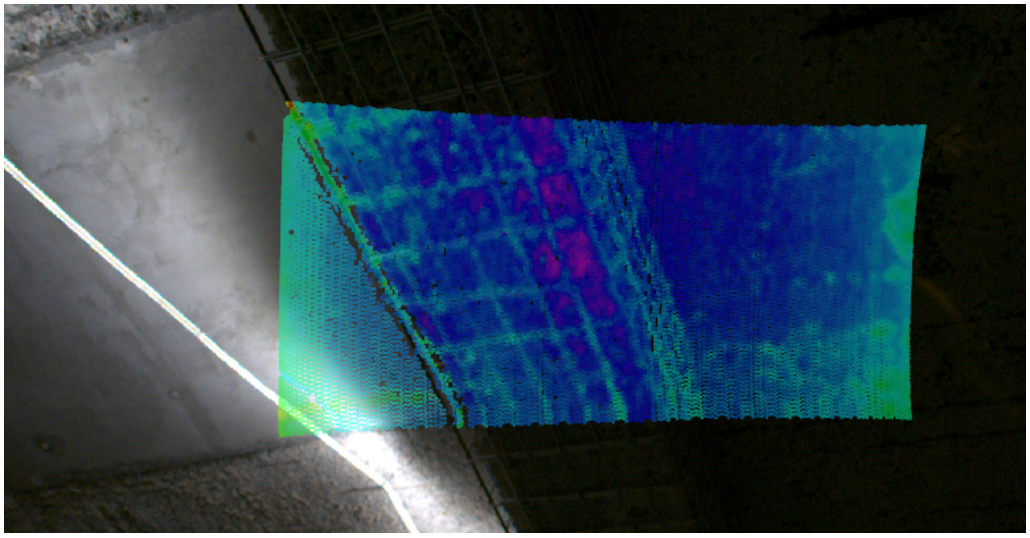


Figure 6.7: Steel rebar surrounding a beam

Figure 6.7 shows a better calibration resolution. The structure is clearly differentiable, and the shape combines with the camera's image. The calibration results seem to be good, providing a correct data fusion. Finally, another type of data collected is the surroundings of the construction zone, the material and machinery used and the obstacles a robot could face.

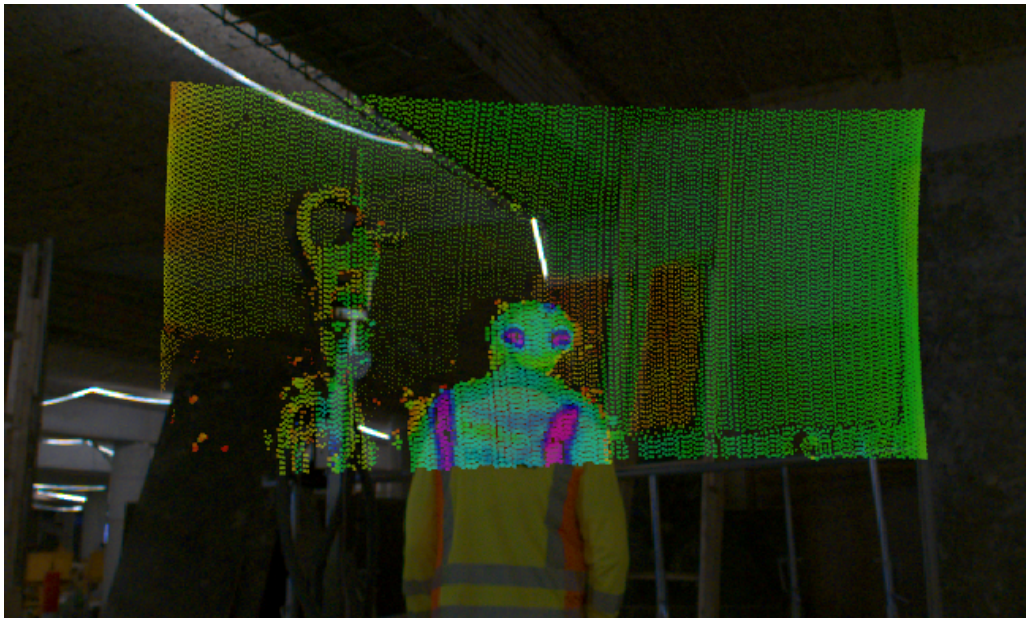


Figure 6.8: People and scaffolding

Figure 6.8 shows the possible situation of robots in construction areas, where they interact with people and scaffold structures. The robot must be able to detect them precisely to localize and avoid them. At this point of the testing, the first problem of the system was detected. The LiDAR has moved from the original position inside the case due to not being fixed correctly to the attachment. The image shows the point-cloud of the person with some variations in the fusion.

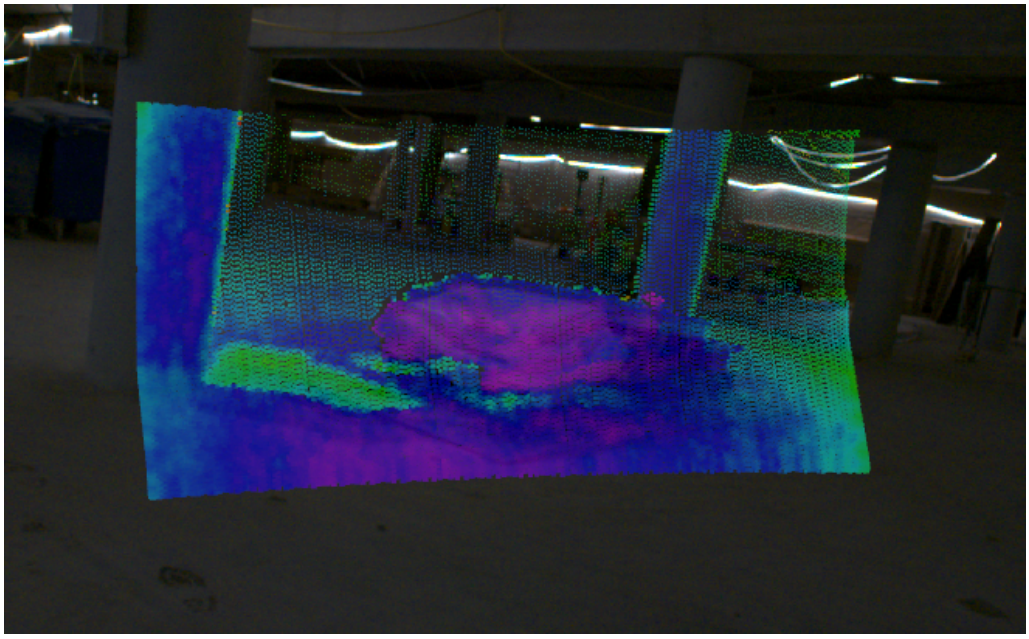


Figure 6.9: Obstacles

After moving the LiDAR to the original position, the problem was slightly solved, as moving it to the exact location was impossible. Figure 6.9 shows more obstacles with better fusion than the previous image but still with some errors.

Thanks to the data collected, it has been possible to receive some feedback on the developed system, taking notes on improvements that can be made to the design and the calibration procedures.

All the codes used in this project are uploaded to a GitHub repository.

7 Future work

This chapter discuss some ideas for the improvement of the results and data gathered that can be helpful for other future projects.

7.1 Improvement of the calibration

There are several factors that can alter the extrinsic calibration of the sensors and directly affect the quality of the data gathered.

7.1.1 Rigid fixing of the components

Initially when designing the case, slotted holes in the Z direction were made to move the LiDAR in order to find it's best position. However this had the downside that if the LiDAR is not correctly fastened to the holes it can move around in the Z direction as well as in the yaw axis due to some tolerances. This movement makes for a big error in the transform from LiDAR to camera which is not good when joining the data. The fix to this problem would be to redesign the base and make normal holes. Another option would be using self locking nuts to avoid the bolts coming lose.

7.1.2 Changing the checkerboard paper

Due to the reflectivity differences of black and white, in the point-cloud data there were big disparities in the Z values of the checkerboard plane. See Figure 7.1. This irregularities can be altering the results of the calibration and make it hard for the program to automatically find the checkerboards. Our solution was to modify the region of interest to include

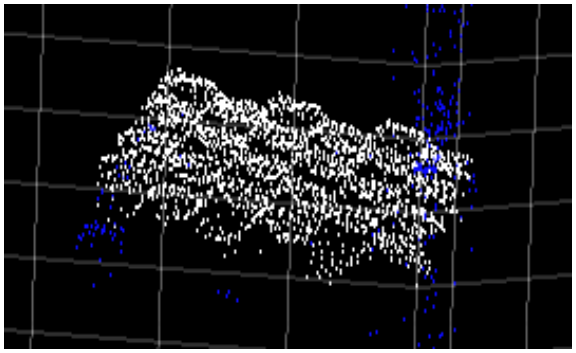


Figure 7.1: Checkerboard Z values disparities between black and white checkers

only the checkerboard points. As well as lowering the cluster threshold and incrementing the dimension tolerance in order to find some of the checkerboards. However this last two things can be compromising the quality of the final calibration result.

Then, if better data of the checkerboard was obtained the calibration process would be easier and more reliable. There is a physical way of improving the measurements of the LiDAR from the checkerboard and that is to use photographic paper instead of normal one as it will make the checkerboard more reflective and therefor the LiDAR will get better measurements. However there is a trade-off with the camera data, as a more reflective paper can make for overexposed parts of the checkerboard due to some sources of light making it hard for the program to detect the checkerboard.

Another way of improving the Z measurements of the checkerboard could be to increase the power of the LiDAR which will make for more accurate measurements in the darker parts.

7.1.3 Switching the lenses of the camera to ones with a greater focal length

Most of the LiDARs in the market are 360 degrees and low resolution, this means that a wide angle lenses is a great complement for this setup as we will focus on the information that is on a close range and therefore we want a great field of view.

However the Neuvition titan M1-A is a high resolution solid state LiDAR, that is made for gathering information at long distances but at a cost, only a 45 degrees horizontal field of view.

The problem comes with the wide angle lens used for the camera. Capturing such a great field of view means that there will not be a lot of information in pixels of the 45 degrees field of view that we are interested in, therefor we are wasting a lot of resolution of the sensor in parts of the image that we are not using to complement the point-cloud. The most evident solution then would be to change the lenses. To calculate the focal length of the new lens for the camera we would need to use the following formula:

$$FOV = 2 \cdot arctan\left(\frac{sensors\ size}{2 \cdot focal\ length}\right)$$

Knowing that our sensor horizontal size is 6.6mm and the wanted horizontal FOV is 45 degrees an 8mm focal length lenses would be the best replacement for the camera.

7.2 Add new sensors

The addition of sensors to the data collection platform would allow for more data to be collected in future deployments.

7.2.1 Stereo camera

An stereo camera consists on two different cameras separated by a distance (baseline) that works in the same way as the human eyes, with the processing of the images by software we can obtain depth of the points in the image. Then it can be useful for generating point-cloud with RGB values for feature detection. A case was designed in order to include more sensors in the future. See Figure 7.2

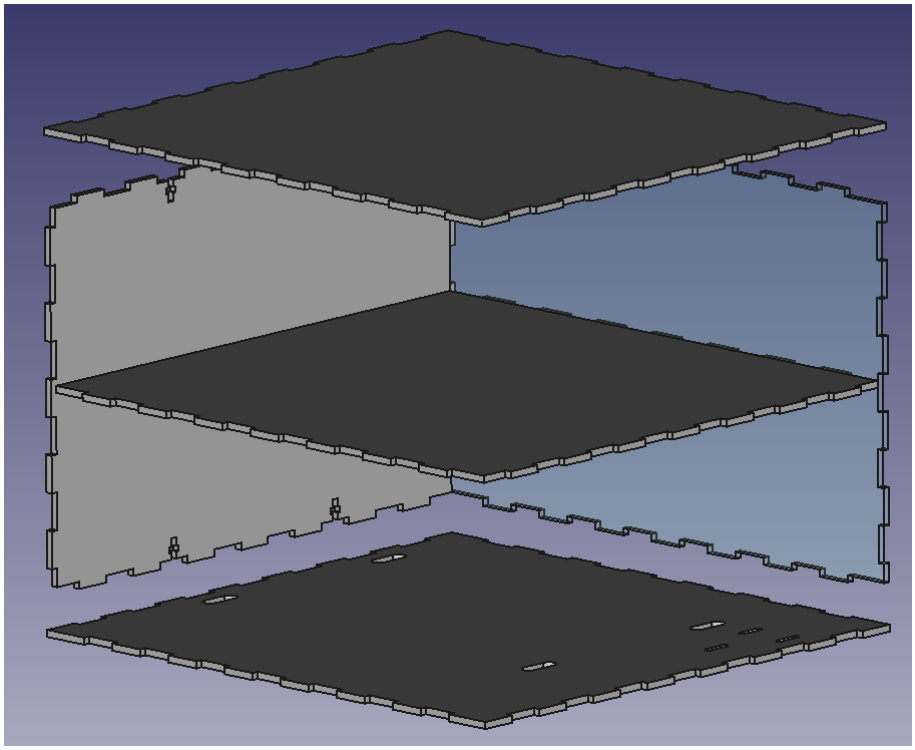


Figure 7.2: Case with two layers to allocate more sensors. The top and the bottom parts are bolted to the 3D H structure making it easy to mount the sensors.

8 Summary

This thesis project aims to develop a perception system for a robot that collects data using a solid-state LiDAR and a monocular camera. A protective housing was designed using FreeCAD and fabricated with a laser cutter to ensure the sensors' safety in a construction area, where the robot will be deployed as part of the RoBétArmé project. The housing also maintains a fixed relative position between the sensors to enable synchronized movement.

The monocular camera suffers from lens distortion, which can be corrected by calculating its intrinsic parameters. Various methods were considered, including Python with OpenCV or MATLAB. However, the ROS package provided an intrinsic calibration solution that generated a file usable in the camera drivers.

For accurate data fusion, it is crucial to determine the extrinsic parameters that define the transformation between sensor frames. Although some existing methods were incompatible with the solid-state LiDAR used, two successful approaches were implemented and tested: one in MATLAB and the other in ROS. The ROS implementation yielded better translation results, while the MATLAB method exhibited errors in closer objects data fusion mostly due to this translation error. However the Matlab method seems to obtain a more accurate rotation matrix between frames with a combined mean rotation error of about a 2 percent, meaning that it will provide good results in further distances were the rotation errors become more noticeable than the translation ones.

During testing in a construction area, several observations were made regarding the project's strengths and weaknesses. The housing structure needs improvement to ensure rigidity and prevent sensor movement. The intrinsic calibration effectively addressed distortion, while the extrinsic calibration, although satisfactory for data fusion, it can be improved. Using better calibration target, changing the material of the paper of the checkerboard or switching the lens of the camera.

Finally it was impossible to fulfill the mounting of the sensors unit into the robot due to the lack of availability of a pan-tilt unit.

In conclusion, the project achieved positive results, providing an easily deployable system for data collection. The knowledge gained will contribute to future enhancements, including better results and the incorporation of additional sensors.

Bibliography

[Alm+16] Pedro Almeida et al. “Shaping the Future of Construction: A Breakthrough in Mindset and Technology”. In: (May 2016). DOI: 10.13140/RG.2.2.21381.37605.

[ROS] ROS. *Robot operating system website*. URL: https://www.ros.org/.

[Bas] Basler. *Camera website*. URL: https://docs.baslerweb.com/a2a1920-160ucbas.

[Neu] Neuvition. *LiDAR website*. URL: https://www.neuvition.com/products/titan-m1-a.html.

[Mata] Mathworks. *What Is Camera Calibration?* URL: https://es.mathworks.com/help/vision/ug/camera-calibration.html.

[Zha00] Z. Zhang. “A flexible new technique for camera calibration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334. DOI: 10.1109/34.888718.

[Ayy] Vasista Ayyagari. *Camera Calibration — Theory and Implementation*. URL: https://medium.com/analytics-vidhya/camera-calibration-theory-and-implementation-b253dad449fb.

[ZLK18] Lipu Zhou, Zimo Li, and Michael Kaess. “Automatic Extrinsic Calibration of a Camera and a 3D LiDAR Using Line and Plane Correspondences”. In: (2018), pp. 5562–5569. DOI: 10.1109/IROS.2018.8593660.

[CPl] A& CPlastics. *Polycarbonate vs Acrylic - Learn about the Difference Between Acrylic & Polycarbonate Material | A& C Plastics*. URL: https://www.acplasticsinc.com/informationcenter/r/acrylic-vs-polycarbonate.

[Fre] FreeCAD. *FreeCAD: Your own 3D parametric modeler*. URL: https://www.freecad.org/.

[Ope] OpenCV. *OpenCV - Open Computer Vision Library*. URL: https://opencv.org/.

[Jam] Patrick Mihelich James Bowman. *camera_calibration - ROS Wiki*. URL: http://wiki.ros.org/camera_calibration.

[Matb] Mathworks. *Lidar and Camera Calibration*. URL: https://es.mathworks.com/help/lidar/ug/lidar-and-camera-calibration.html.

[Matc] Mathworks. *Calibration Guidelines*. URL: https://es.mathworks.com/help/lidar/ug/lidar-camera-calibration-guidelines.html.

[Matd] Mathworks. *Estimate world frame coordinates of checkerboard corner points in image - MATLAB estimateCheckerboardCorners3d - MathWorks España*. URL: https://es.mathworks.com/help/lidar/ref/estimatecheckerboardcorners3d.html.

[Mate] Mathworks. *Get Started with Lidar Camera Calibrator*. URL: https://es.mathworks.com/help/lidar/ug/get-started-lidar-camera-calibrator.html.

[Bel+] Jorge Beltrán et al. *velo2cam_calib repository with the package of the Automatic Extrinsic Calibration Method for LiDAR and Camera Sensor Setups*. URL: https://github.com/beltransen/velo2cam_calibration.