**HOCHSCHULE RAVENSBURG-WEINGARTEN**

**UNIVERSITY OF APPLIED SCIENCES**

*A report submitted in partial fulfillment of the requirements for the Award of Degree of*

**Bachelor in E-Mobility and Green Energy**

# Calibration method between 2D LiDAR and Camera

May 30, 2024

BACHELOR THESIS

**Prepared by:**
Anupam Kshetri (33471)

**Supervised by:**
Prof. Dr. rer. nat. Stefan Elser, Ravensburg Weingarten Hochschule
Felix Berens M.Sc., Ravensburg Weingarten Hochschule

## Statutory Declaration

I, hereby declare that the content of this thesis named “Calibration method between the 2D LiDAR and Camera" is all my work except for the quoted literature and other sources to which the sources are provided.
I have clearly provided the reference and source to the part or image that I have taken while writing this thesis.
I understand that any violation of this declaration could lead to academic failure.

Weingarten, ...... 2024

........................................
Name and Signature

# Acknowledgement

I would like to express my sincere appreciation to Prof. Dr. rer. nat. Stefan Elser for being my supervisor and examiner and providing constant support and knowledge throughout the course of my thesis. Without his help, I am sure the project would not have been completed properly. It has been a truly valuable and enlightening experience and I feel privileged to have been a part of such an innovative topic.

I would also like to express my sincere gratitude to Felix Berens M.Sc for his guidance and recommendations for required improvements throughout my thesis. His insights and constructive feedback have helped me to develop new skills and knowledge that will be valuable in my future engineering career. I would like to acknowledge the efforts of the professors and academic staff at RWU for their guidance and support throughout my studies, which have helped me to make the most of my thesis experience.

Finally, I am grateful to my family, friends and my girlfriend for their unwavering support and encouragement, which has been a constant source of motivation and inspiration.

HOCHSCHULE RAVENSBURG-WEINGARTEN UNIVERSITY OF APPLIED SCIENCES

# Contents

# List of Abbreviations

| | |
|---|---|
| 2D | Two Dimensional |
| 3D | Three Dimensional |
| SLAM | Simultaneous Localization and Mapping |
| v2 | Version Two |
| MP | Mega Pixel |
| mAh | Milliampere Hour |
| Wh | Watt Hour |
| LiDAR | Light Detection and Ranging |
| FoV | Field of View |
| ICP | Iterative Closest Point |
| Hz | Hertz |
| USB | Universal Serial Bus |
| IOT | Internet Of Things |
| OpenCV | Open Computer Vision |
| TPE | Tree-structured Parzen Estimator |

# List of Figures

# Listings

# List of Tables

## Abstract

In this thesis, we investigate the method for the calibration of the Camera and 2D LiDAR using an ArUco marker as a reference point instead of a traditional Charuco marker or chessboard. Unlike other calibration methods where 3D LiDAR is used, this 2D LiDAR poses a unique challenge as it doesn‘t provide the depth information so it is quite difficult to achieve accurate alignment between these sensors. However, this thesis proposes an innovative calibration approach combining the feature extraction from camera images with corresponding point identification in 2D LiDAR scans to estimate the extrinsic parameters of both sensors. Several experiments were carried out placing the ArUco markers at different distances and orientations. The extrinsic parameters were at first manually provided and were later optimized automatically to find the precise parameters‘ values for proper calibration ensuring robust and accurate results.

# 1 Introduction

Cameras and LiDARs are very important in today's robotic perceiving systems. While camera provide perception data regarding the outlook, structure and color of the surrounding, LiDAR provides the information about the actual position and the precise distance of the object. Cameras has an advantage at identifying colors and other visual characteristics while has disadvantage to precisely provide the information of the surrounding during several environmental conditions like: fog, at low lighting condition (night time) and on the other hand LiDAR can still provide the accurate result even in the night time or in different environmental conditions but lacks the ability to distinguish colors and texture of the surrounding. Each sensor has their disadvantage which can be fulfilled by another sensor. So, by combining the abilities of these two sensors, the disability of the sensors can be nullified. This also means that the perception capabilities of robots can be optimized making the decisions of the robots more accurate even in complex real world scenarios. But in order to achieve the combined perception ability, these sensors need to be calibrated properly. Proper calibration of the Camera and LiDAR sensors ensures the robots to work better by aligning and synchronizing the data from these calibrated systems to reduce errors and disparities during tasks of perception. In addition to this, the Camera-LiDAR calibration lays a foundation for other advanced applications, such as: as navigation, object tracking, and other applications like SLAM.

## 1.1 Motivation

The previous researches [1], [2], [3], [4] has dealt with the calibration between the camera and 3D LiDAR, and it generally depends on the depth data from the 3D LiDAR using Checkerboard, ChArUco, or even Chessboard as a reference point. However, not many researches has been done regarding the calibration of 2D LiDAR and camera presenting a strong motivation for this study. Unlike the other method in which the calibration is done between 3D LiDAR and camera in combination with checkerboards, this thesis proposes a way to calibrate the 2D LiDAR and camera using ArUco markers as a reference point. This algorithm gives a strong framework for estimating the translation parameters (X, Y, Z) and rotation parameters (roll, pitch, yaw) of both of the sensors at different orientations to achieve lower calibration errors. In this thesis, we also present an automated method to obtain the suitable calibration parameters using Bayesian optimization to minimize the calibration errors in our collected dataset with images and LiDAR cloud points. Given the huge search space of the parameters, we used Bayesian optimization over other methods like grid search to obtain these parameters at reasonably time duration. The source code of this thesis is publicly available.[1]

[1]Source Code: `https://github.com/AnupamKshetri/Thesis.git`

## 1.2 Objectives

The objective of this thesis is to develop an algorithm to accurately estimate the extrinsic parameters for the calibration of the Camera and 2D LiDAR. We validate the accuracy of the transformed LiDaR points by visualizing them on the ArUco marker. Furthermore, we also mention the issues that arised while using the ArUco marker as the reference.

The calibration is carried out by using a square-shaped box (see Figure 6) with an ArUco marker attached at every side. To maximize the accuracy of the extrinsic parameters, the box is placed at different distances and orientations. The purpose of this thesis is also to present full documentation accompanied by a detailed analysis of how this calibration was done together with the step-by-step procedure followed for gathering the data, processing, and algorithms.

## 1.3 Outline

This section outlines the organization of the chapters which discusses various points concerning the calibration of the camera and 2D LiDAR with ArUco markers. Chapter 1 addresses the motivation, objectives and a brief overview concerning the thesis. Chapter 2 discusses various methods used for calibration and outlines the proposed method. Chapter 3 provides the information about the system with hardware and software used and the position of the camera and LiDAR in the PiCar. Chapter 4 covers the details of data collection and processing and it also include process involving the identification and segmentation of ArUco markers from camera images and from LiDAR point clouds. Chapter 5 describes the complete process of converting LiDAR points to camera pixels. Chapter 6 demonstrates the results of calibration and assess the calibration error and covers the performance analysis of the calibration method in terms of the time taken and with regard to scalability. Chapter 7 concludes the work in this thesis and suggests potential directions for future research.

# 2 Literature Review

## 2.1 Methods of Calibration

Till now, several researchers have used different methods for the calibration of LiDAR and cameras. These methods can be generally classified into Target based calibration and Targetless calibration.

### 2.1.1 Target based calibration

Target based calibration method uses a calibration pattern with known features such as: borders, edges and boundaries. It makes easy for both the camera and lidar to scan and detect the corresponding features of the calibration board like: Chessboard, ArUco or ChArUco allowing for better calibration. [4] focused on target based approach using targets of known dimension and accuracy to find the extrinsic calibration of a LiDAR camera. As a result, they were able to achieve a reduction in projection error by 50%. While [1], [2] and [5] used chessboard as their reference point, [6] used ArUco marker as a reference point. Basically, in target based calibration, the calibration patterns are aligned within the FoV of both sensors and the data is collected. The data from both the sensors is then analyzed to find the corresponding features to estimate the extrinsic parameters for the calibration. Based on the findings presented by [6], it was noted that their algorithm is able to predict the parameters conveniently using fewer correspondences compared to higher correspondences required for ICP (Iterative Closest Point).

However, there may be some disadvantage of using this technique. The method may be inapplicable when conducting the task in a room or an area with low lighting conditions. Furthermore, target quality is also important for proper estimation. If the features or corners are not clearly visible by both sensors, then it is difficult to find the correspondence between the sensors and leads to inaccurate results.

### 2.1.2 Targetless calibration

Targetless Calibration does not require specific calibration patterns or markers of any form for calibration. Extrinsic parameters requirements are all attained from the environment by establishing a correspondence between the data acquired through LiDAR and the camera. [7] has proposed four different methods for targetless calibration depending upon the usage of the potential information by the sensors to solve the calibration problem and they are: information theory based, feature based, ego-motion based, and learning based methods. [8] introduces a calibration method to align camera and LiDAR with small FoV for completing transferring and transforming tasks in geometric data. The technique developed is an adaptive voxelization method, in which the LiDAR feature extraction and matching process is faster with less computational time and with high accuracy. It also shows that the proposed algorithm has advantages over the ICP-based algorithms in the areas such as feature matching and optimization technique as well as computational time. Finally, the proposed method offers a comprehensive solution for the extrinsic calibration of multiple small FoV LiDAR units and cameras that is crucial to construct the entire autonomous car subsystem. [9] conducted a targetless and structureless approach in their spatiotemporal Camera-LiDAR calibration, with the objective of spatial and temporal alignment for LiDARs and visual cameras to improve its reliability and accuracy in robotic systems. The proposed approach is claimed to have the ability to provide an estimate of spatiotemporal parameters even when there is no view overlap between the camera and LiDAR and it is not affected by the time lag and can be calibrated as it runs without needing a specific motion of the sensors.

# 3 System Overview

The PiCar is the most important element of this thesis where all the necessary components such as: wheels, motors, motor control board. Raspberry Pi, LiDAR sensor, and the camera are incorporated into the robotic platform. It has variety of applications specifically it is used for better perception and navigation in robotics. During the calibration process the PiCar is in rest state, whereas the LiDAR and or camera is constantly collecting or capturing the data from the environment. Since, the PiCar doesn‘t require to move around for this research, the components like: Motor, Wheels , motor control board are barely used.

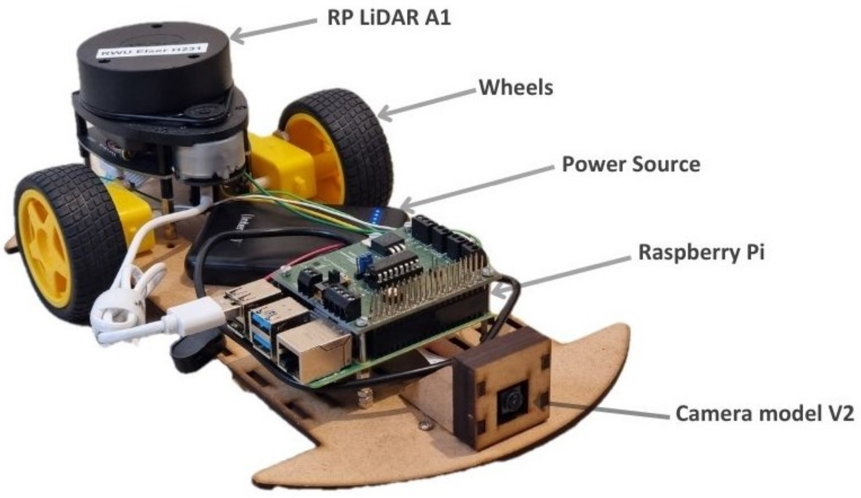


**Figure 1:** PiCar

As the method used in this thesis is Target-based calibration, the PiCar should be kept in such a way that LiDAR and camera can have a clear vision of the ArUco marker box as in (Figure 9). This arrangement allows the sensors to gather the information from the surrounding with ArUco marker and the data collected is then processed to find the correspondence to determine the possible extrinsic parameters. Some of these components include the motor, the wheels, and the motor control board; however, since the PiCar is kept stationary in this thesis, information about these components are out of scope of this thesis.

## 3.1 Hardware Components

### 3.1.1 RPLiDAR A1

The LiDAR used in this thesis is the RPLIDAR A1, developed by SLAMTEC. This is a 2D type LiDAR and can perform 360° scan within 12-meter range. The scanning frequency of this LiDAR can reach up to 5.5 Hz when sampling 1450 points each round. And it can be configured up to 10Hz maximum. With options for both indoor and outdoor use, it can be used for robotic applications, including mapping, localization, and obstacle detection. It contains a a state-of-the-art 2D laser scanning system and motor unit where the scanning is done by the first and the latter ensures the movement. As the LiDAR is fed with the power, it starts to spin in a clockwise direction and the users can access these data collected via Serial port or USB. It is based on laser triangulation ranging principle and uses high-speed vision acquisition and processing hardware. The system measures distance data in more than 8000 times' per second and with high resolution distance output (<1% of the distance) [10].

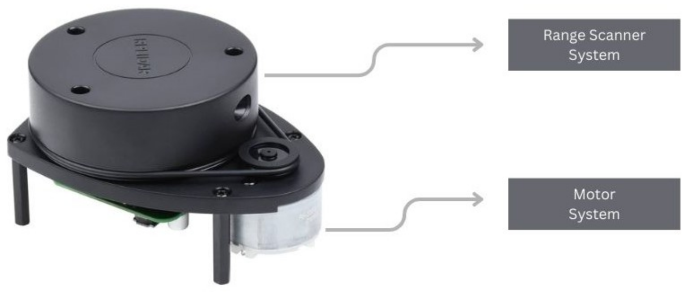


**Figure 2:** RPLiDAR A1
Source: [11]

### 3.1.2 Camera Module v2

Camera is the integral component of this research. For calibration, it is very important to detect the patterns of the reference points from the images. So, to capture the images, a high resolution camera should be used. To meet this requirement, the Camera Module v2 is used which was developed by the Raspberry Pi organization. It is very small in size and comparably cheap compared to other cameras. This camera was designed to be used with Raspberry Pi boards. The sensor has a very high resolution of Sony IMX219 with an 8-MP resolution [12]; it is capable of taking images and capturing video.

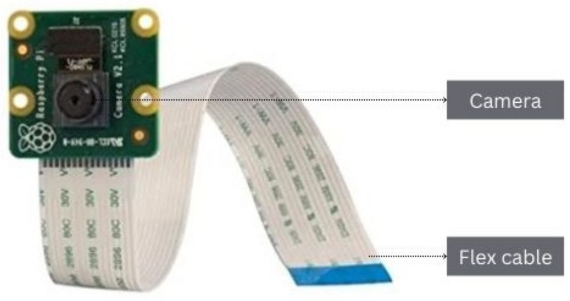


**Figure 3:** Camera Module v2
Source: [13]

To be able to use the camera, it is interfaced to the Raspberry pi board through the Flex cable. It is popularly used in robotics projects, OpenCV applications, and IoT (Internet of Things) projects because it can be accessed with different Raspberry Pi devices, and many software libraries are present to operate the camera.

### 3.1.3 Raspberry Pi

The Raspberry Pi 11 (bullseye) unit is a development board used to perform all the computational and controlling functions of this research. Operating on the Raspbian GNU/Linux 11 (bullseye), Raspberry Pi allows smooth and efficient execution of the algoithms and interacts with different peripherals as well as conduct operations on various types of sensors. The pins of the camera and LiDAR are directly connected to the Raspberry Pi. Raspberry Pi works as the central operational hub and manages the operation of camera and the LiDAR sensors. It allows for the smooth data collection and the RAM present in raspberry pi allows to store the data for further process. In addition, it has high processing powers and is compatible with the programming language, Python which is used in this thesis.

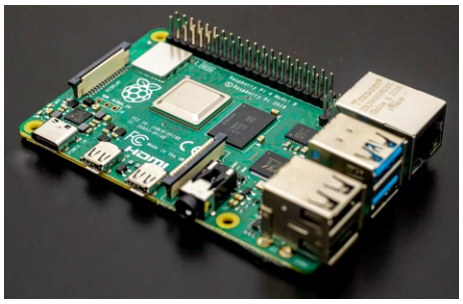


**Figure 4:** Raspberry Pi
Source: [14]

### 3.1.4 Power Supply

In this thesis, a rechargeable battery with the capacity of 5000mAh and 18.5 W is used to power both the LiDAR and camera systems. The 5000mAh massive battery power ensures that operations such as LiDAR and cameras will be able to persist for long duration, enabling the continuous collection and processing of data. This becomes crucial under calibration procedures where continuous operation has to be done to capture a lot of data from several orientations and positions. Through enhanced battery power, portable construction and convenient charging solutions, the devices help to ease for conducting the research. Another advantage is that it is a rechargeable battery which means it can be charged and reused unlike other batteries which have to replaced time and again. Also the battery is small in size due to which it can be mounted in the PiCar.

**Figure 5:** Battery

## 3.2 ArUco Markers

The ArUco markers are applicable in a vast majority of cases. They are mainly used in applications related to computer vision and robotic fields, object tracking and camera calibration, as well as augmented reality. ArUco marker is generally square in shape consisting of black border and the inner square of either black or white. It is easily detectable even under low light conditions. ArUco marker pattern differs according to the dictionary and the marker ID.

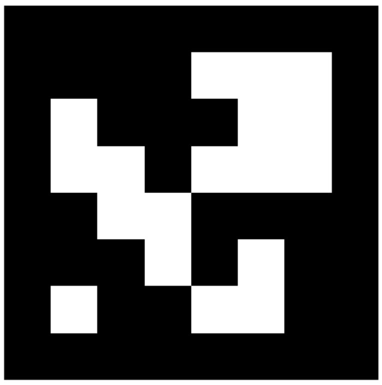


**Figure 6:** ArUco Marker
generated from `https://chev.me/arucogen/`

In this thesis, we used an ArUco marker of dictionary 6*6 (50, 100, 250, 1000) with different marker ID. Six different ArUco markers were printed and glued on each side of the cube so that it can be used as a reference for the LiDAR camera calibration. ArUco markers were used as the reference point because they have distinct features which can be easily detected by the camera and also there is already an existing OpenCV library that shall further make the calibration tasks easy. Moreover it is simple to find the correspondence between points in the LiDAR and camera information allowing determining the accurate extrinsic parameters between these two sensors.

## 3.3 Camera and LiDAR placement

The position of the sensor is very important for the calibration process. From the figure 5, it can be seen that the camera is mounted at the front while LiDAR is mounted at the rear of the PiCar. This cause a vertical offset of approx. 48 cm and a horizontal distance of approx. 260 cm between the two sensors.

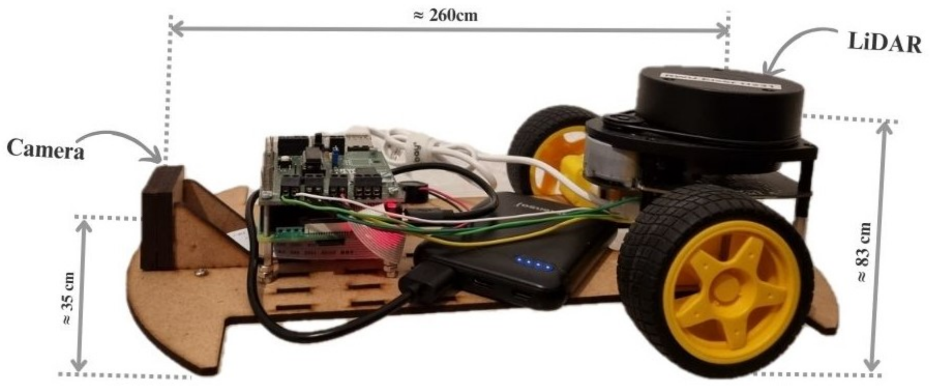


**Figure 7:** PiCar with offset

During the calibration, such offsets must be considered because they greatly affect the relative positioning and orientation of the sensors. If these differences are ignored, they will cause much inconsistent data measurements with the sensors and will minimize the accuracy of the results of the sensor fusion algorithms. In a simple sense, a failure to achieve proper synchronization between the sensors may result in any of the detected objects from one sensor to appear in different location within the another sensor coordinate frame misleading the information of the surrounding.

# 4 Methodology

## 4.1 Data Collection

### 4.1.1 Camera Image Collection

The initial step for camera calibration starts with capturing a series of images with the reference point located within the FoV of the camera. In our case, ArUco marker is used as a reference point. It is important for the ArUco marker to be placed at different distances and orientations. The variations in the distances and the angles will permit the calibration algorithm to adapt to different perspective and environment, which makes it applicable in all places in real life without losing accuracy. Finding the parameters utilizing such variation will assist in the creation of a calibration model which will have the possibility of estimating the intrinsic and extrinsic parameters of the camera accurately across a wider scope of operating conditions. Similarly, We tried to keep the environment and the lighting condition same while collecting the data. The ArUco marker in the image needs to be properly visible in order to further the calibration process. If the light condition is poor then the distinct feature of the marker may not be visible and can result in inaccurate calibration.

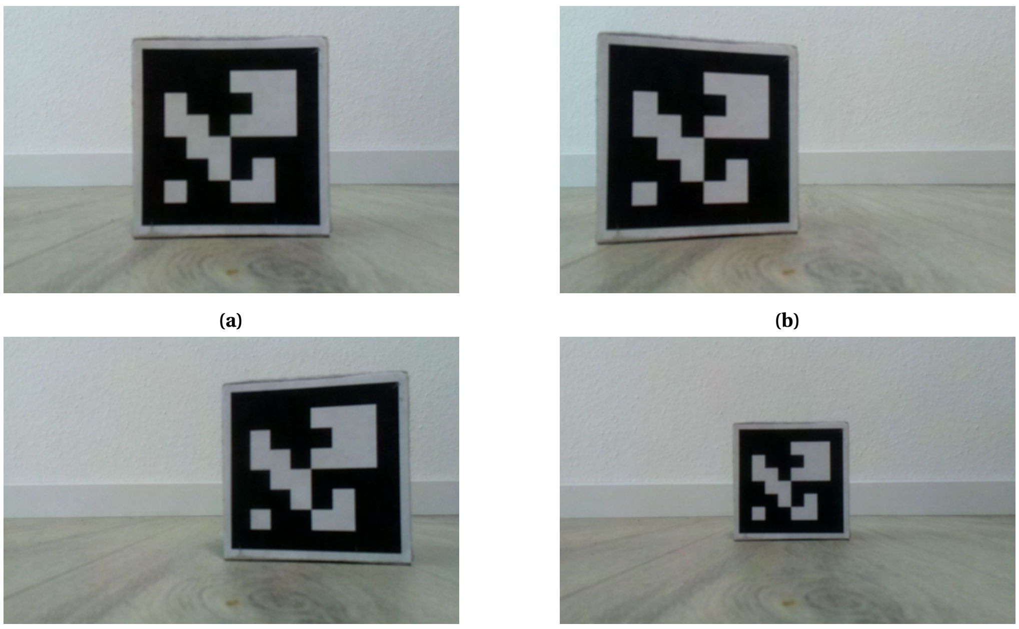


**Figure 8:** Sample of images

After all above mentioned criteria are fulfilled, the image are then captured and stored in a separate repository. We collected and used 20 images in this thesis. However, the number of images can be more or less depending upon the necessities.

### 4.1.2 2D LiDAR Point Collection

The LiDAR that I am using for my thesis is RPLiDAR A1 which is capable of collecting the point cloud data from 360°. I have written an algorithm which captures images from camera and collects point cloud data from LiDAR simultaneously when the 'I' key on the keyboard is pressed. This method made it possible to get LiDAR point cloud data and camera images at the same time, and therefore the point cloud data and the camera images could be timed perfectly, which was very necessary for the further analysis. The lidar data consists points clouds collected from 100 revolutions to make sure that the whole surrounding was scanned without leaving any blank spots or gaps in the point cloud and hence to create a point cloud with a higher density and the detail.

Throughout the process, PiCar was kept in stationary state and so the LiDAR and camera. However, the square box containing the ArUco marker was moved to different distances and orientations within the FoV of the sensors. Before manually changing the location of the box, the image captured and point cloud data collected were ensured to be stored properly in the designated folder so that it can be used later on for calibration.

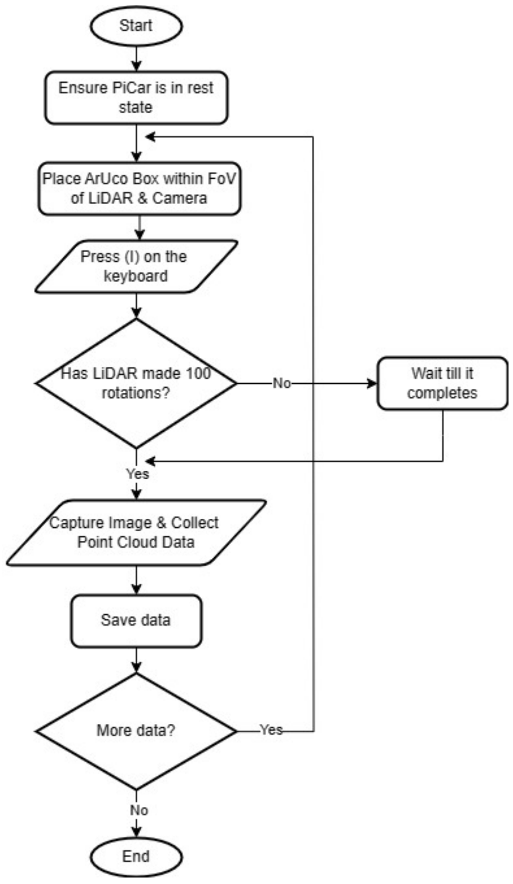

Start
Ensure PiCar is in rest state
Place ArUco Box within FoV of LiDAR & Camera
Press (I) on the keyboard
Has LiDAR made 100 rotations?
No
Wait till it completes
Yes
Capture Image & Collect Point Cloud Data
Save data
More data?
Yes
No
End

**Figure 9:** Flowchart for LiDAR data collection

## 4.2 Data Processing

### 4.2.1 Image Undistortion

Image undistortion is very crucial step for the calibration process. The image captured from the image can sometimes seem distorted either radially or tangentially [2] as in (Figure 8). If the estimation of the extrinsic parameters of the camera is carried out with such image then the found values will have a lot of inaccuracies. Using such inaccurate values will cause difficulty while plotting the point cloud data of the LiDAR onto the image.

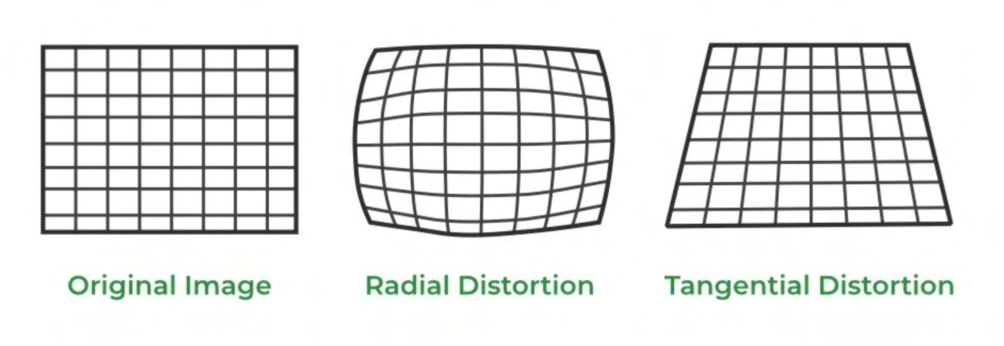


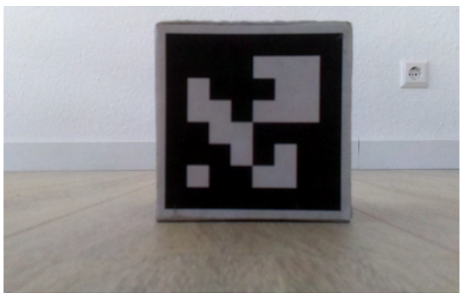
**Figure 10:** Distortion types
Source: [15]

The estimation of the distortion coefficients is done automatically using the OpenCV function i.e.
`cv2.calibrateCamera()`.

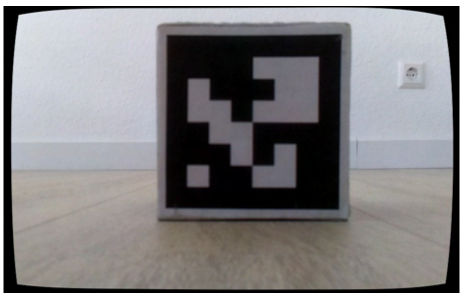
$$\text{Distortion coefficients} = [k_1, k_2, p_1, p_2, k_3]$$

Here $k_1, k_2, k_3$ are radial distortion coefficients and $p_1, p_2$ are tangential distortion coefficients.

The distortion coefficient found from above function is then used to undistort the image using again the OpenCV function i.e. `cv2.undistort()`. Thus obtained images are now applicable for the determination of the parameters with high accuracy.



**(a)** Distorted image



**(b)** Unistorted image

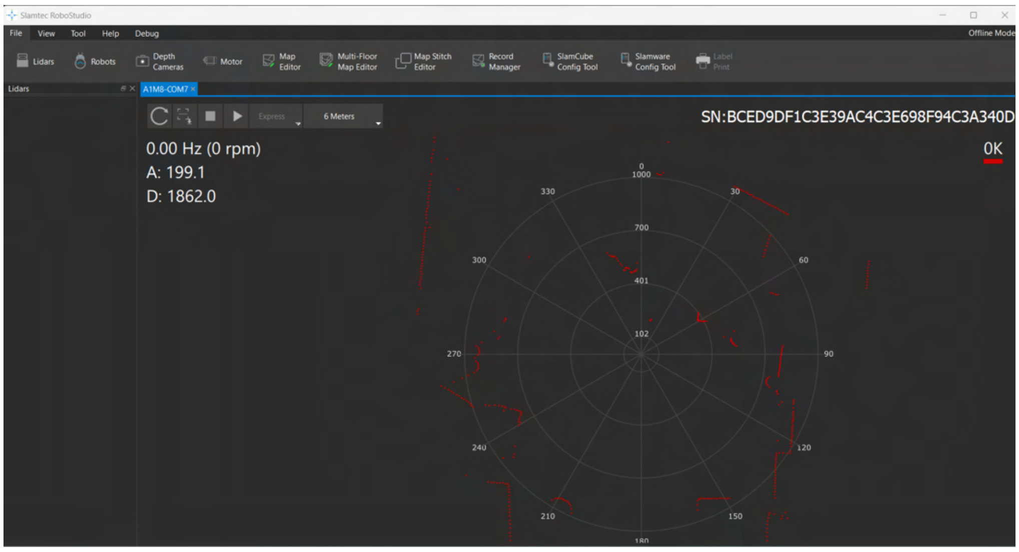
**Figure 11:** Image Undistortion

[2]For more information about Distortion: `https://en.wikipedia.org/wiki/Distortion`

### 4.2.2 LiDAR Data Filtering and Validation

Another critical need for the camera and LiDAR calibration of the robot is that the reference point (ArUco marker) needs to be in the FoV of both camera and LiDAR. While the FoV of camera is small in comparison to LiDAR since it captures data from an angle of 360 degree. So, as described earlier, the marker should be in front of the camera that is within the FoV of the camera. So it is not necessary that all the data obtained from the 360° is useful for the process rather only the data from the area of the region of interest i.e. the FoV of the camera is needed. This method to filter LiDAR data can allow to achieve results faster reducing the processing time and can also increase the precision of the LiDAR data processing.

We used the software "RoboStudio[3] " to determine teh FoV of the camera. For this, I run both the camera and LiDAR at the same time. Looking at the GUI of the camera, I placed two object at the edges of the camera‘s frame. After this, the LiDAR was connected to the PC. The RoboStudio software then opens the GUI which displays point cloud data of the object from the surrounding. This software offers the user to point the object using the mouse cursor and shows the angle and distance in the real time. So, facilitating from this feature, I pinpointed the two object and its angle from the LiDAR. This angle now corresponds to the FoV of the camera. So after the FoV of the camera is found, I then started collecting the data only from these regions instead of the whole 360 degree.



0.00 Hz (0 rpm)
A: 199.1
D: 1862.0

**Figure 12:** RoboStudio

[3]RoboStudio link: `https://www.slamtec.com/en/RoboStudio`

An analysis has been further conducted for this identified angle range to validate its accuracy. At first, I captured an image that contained the ArUco marker at the center position of the image and calculated its dimensions including height and width to find the Azimuth range. Then with the help of LiDAR, I calculated the distance at which the ArUco marker box was situated from the LiDAR. Now after all this information was gathered, I wrote an algorithm which will return the azimuth range. The angle which I found using RoboStudio was almost similar to the one which I found using the algorithm, however I added extra angles to the determined angle range for assurance.

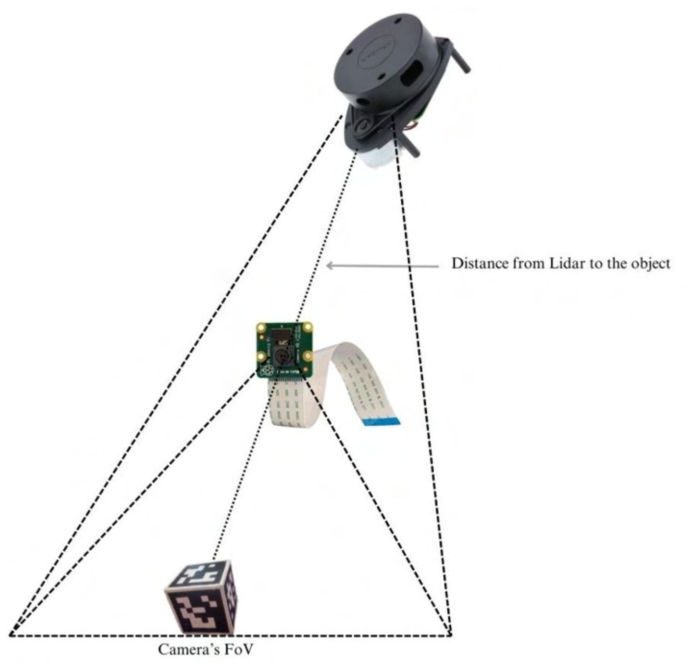


**Figure 13:** Finding FoV

**Mathematically:**

As in Figure 11, if the two lines are drawn from LiDAR with its FoV as the hypotenuse and the second line is drawn connecting these two hypotenuse lines from the region where the ArUco marker is placed, then an Isosceles triangle is formed. When this triangle is split into half then it forms two right angle triangles.

Let‘s suppose one of the formed right angle triangle. Here in this triangle, the distance between the LiDAR and the object serves as the perpendicular ($P$) and the half of the image width as the base ($B$) and one of the line drawn from the FoV to the base as the hypotenuse ($H$). Now from the tangent formula, the angle ($\theta$) can be calculated as:

$$\tan\Theta = \frac{P}{B}$$

$$\theta = \arctan\left(\frac{\mathrm{P}}{\mathrm{B}}\right) \tag{4.1}$$

After finding the angles of the corresponding right angle triangle from the above formula, the Azimuth range can then be calculated. The following algorithm is used to automatically calculate the Azimuth range:

```python
import numpy as np

image_width = 864
lidar_to_aruco_distance = 511
lidar_fov = 360.0
angular_size_rad = np.arctan2(image_width / 2, lidar_to_aruco_distance)
angular_size_deg = np.degrees(angular_size_rad)

azimuth_range_min = 180.0 - angular_size_deg / 2
azimuth_range_max = 180.0 + angular_size_deg / 2

print(f"Azimuth Range: {azimuth_range_min} to {azimuth_range_max}")
```

**Listing 1:** Algorithm to calculate azimuth range

## 4.3 Aruco Marker Detection

### 4.3.1 Camera Image

ArUco markers detection is necessary in order to identify the relationship between the camera and the real world. In the camera image, the ArUco marker can be easily detected using OpenCV function which is
`cv2.aruco.detectMarkers().`

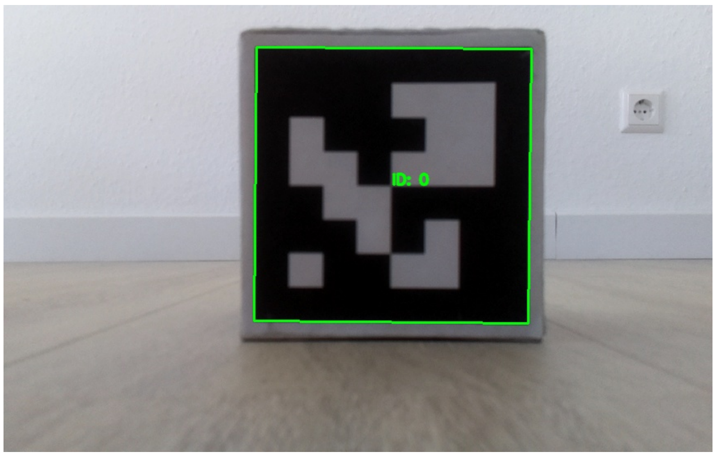


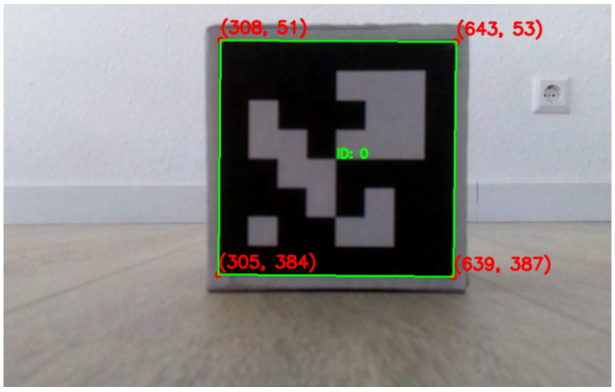
**Figure 14:** Aruco Marker Detection

```python
import cv2
import numpy as np

def get_aruco_marker_inner_corners(image):
    marker_id = 0
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    dictionary = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250)
    parameters = cv2.aruco.DetectorParameters_create()
    corners, ids, _ = cv2.aruco.detectMarkers(gray, dictionary, parameters=
    parameters)
    if ids is not None and marker_id in ids:
        marker_index = np.where(ids == marker_id)[0][0]
        marker_corners = corners[marker_index][0].astype(int)
        return marker_corners
    return []
```

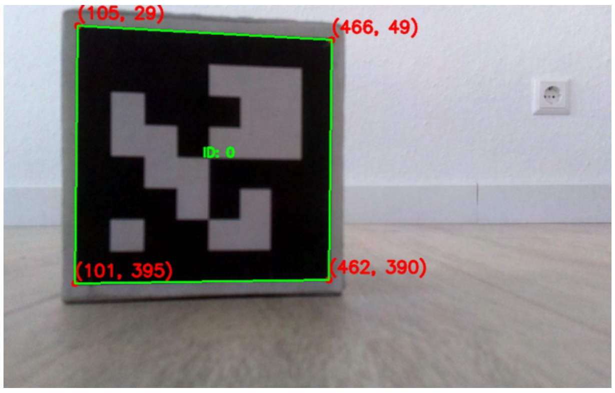
**Listing 2:** Algorithm to detect ArUco Marker

The identification of the ArUco marker in the image allows to estimate the intrinsic parameters of the camera such as: focal length or principal distance, optical center or principal point and distortion coefficient. The camera is simply a device that records the 3D existence of the surrounding and exports it to 2D picture. Therefore, if the ArUco marker in the images are detected in variety of orientation or distances and positions, then it becomes easy for locating the position of the camera with regards to the 3D coordinates of the marker position. This information can then be used for the refining of the intrinsic parameter of the camera for minimizing the reprojection error. In addition, ArUco markers detection could be used to estimate the camera extrinsic parameters which will further help in the process of fusing a LiDAR sensor coordinate system with the camera system for a more accurate estimation.

One of the key processes that are involved in the camera calibration which basically require the determination of the corresponding points in the actual physical world and on the image captured by the camera are the Object Point Detection phase and Image Point Extraction phase. ArUco markers are made in such a way that their corners and patterns can be easily detected with the OpenCV functions. These corner act as the object points and these are defined by 3D coordinates in the real world coordinates. These co-ordinates are then converted into the pixel co-ordinates using OpenCV functions, which in these cases will act as image point. The algorithm that is employed in order to calculate the corners of the ArUco marker can be observed in the (Listing 2).
Some of the examples of the inner corner points found from the samples of image taken for the research are:



**(a)**



**(b)**

**Figure 15:** Inner corner points of the respective images

### 4.3.2 LiDAR scans

Unlike 3D LiDAR, the 2d LiDAR used in this thesis only provides the distance and angular measurements and lacks the depth information. Due to this, LiDAR provides the data only from its scanning horizontal plane. With this info only it is difficult to determine the exact position of ArUco marker in the 3D space.

Due to noisy data in longer distances, we placed the ArUco markers at closer distances and collected the data as shown in (Figure 6). It is worth noting that the LiDAR cannot distinguish the ArUco marker by itself as camera. Hence it was crucial during data collection to ensure no other objects were present near the scanning area of the LiDAR except for the ArUco marker box. Presence of other objects near Aruco marker would make it difficult to determine the lidar readings of the marker in the collected data. Filtering the LiDAR data as I have done before (see 3.2.2) was to filter out the extra data of other region outside the LiDAR's FoV corresponding to the camera so that the data collected was only from the region where solely the ArUco marker box was located.This method streamlined our analysis efforts and directed them towards our area of interest.

After the data is collected by the LiDAR, the output was analyzed for the purpose of identifying points from the ArUco markers in the point cloud dataset. Firstly, the process involved segmenting point clouds and finding cluster of points. The main aim was to look for some sharp changes or discontinuities in the LiDAR data that could indicate the edges expected shape and position of the ArUco marker. There was still one issue with the ArUco marker. Only the ArUco markers corners were obtained when the corners of the marker were extracted from the image as shown in (Listing: 1) and not the corners from the the entire box. If we look at the design of the ArUco marker box, the edge of the ArUco marker isn't the edge of the whole box. Its because the size of the marker is 10*10 while the size of the box is 11 cm on each side. But for the proper alignment of the data between LiDAR and camera, this also should be taken in consideration.

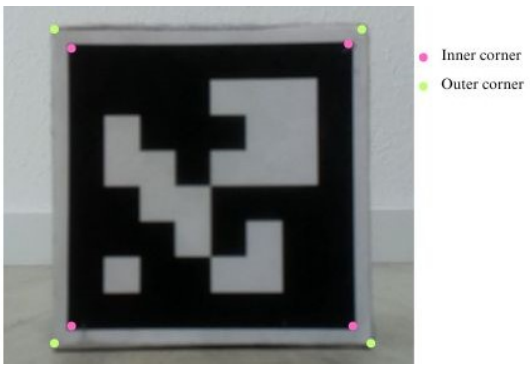


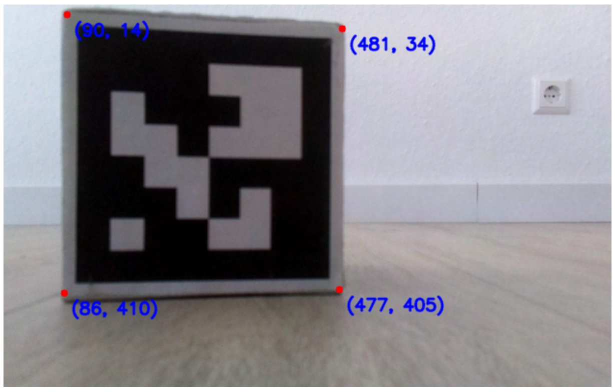
**Figure 16:** ArUco marker with inner and outer corners

In order to find the corners of the whole box, I then modified the existing algorithm by adding an extra margin to the detected corners. The new algorithm as shown below is now able to extract the corner of the whole box. After finding the corners, I compared the corner found from image and LiDAR to get the better position of the ArUco marker in the LiDAR scans.

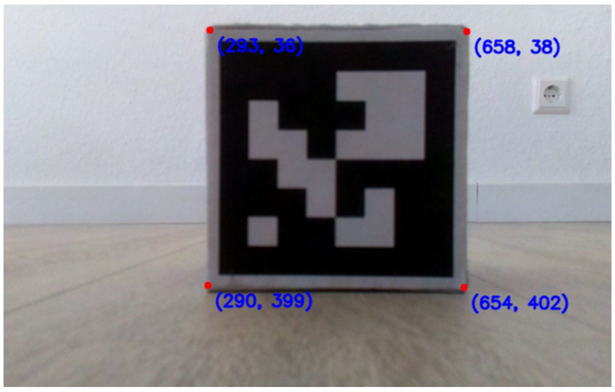
```python
import cv2
import numpy as np

def get_aruco_marker_corners(image):
    marker_id = 0
    margin_mm = 15
    margin_rect = margin_mm * np.array([
        [-1, -1],
        [1, -1],
        [1, 1],
        [-1, 1],
    ])

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    dictionary = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250)
    parameters = cv2.aruco.DetectorParameters_create()
    corners, ids, _ = cv2.aruco.detectMarkers(gray, dictionary, parameters=
    parameters)
    if ids is not None and marker_id in ids:
        marker_index = np.where(ids == marker_id)[0][0]
        marker_corners = corners[marker_index][0].astype(int)
        outer_corners = marker_corners + margin_rect
        return outer_corners
    return []
```

**Listing 3:** Python function to get ArUco marker corners

Some of the examples of the outer corner points found from the samples of image taken for the research are:



**(a)**



**(b)**

**Figure 17:** Outer corner points of the respective images

## 4.4 Obtaining Camera Parameters

Camera parameters are very necessary for the calibration process. The camera parameters comprises of intrinsic and extrinsic parameters. These parameters defines how the lens of the camera view the surrounding around it and also helps to understand camera‘s behavior and their spatial relationship.

Intrinsic parameters defines the internal optical behavior of the camera and helps to understand how camera captures images. The intrinsic parameters are: Focal length, Principal point, Distortion Coefficient. Focal length describes the field of view of the camera and it determines how much scene of the surrounding is captured within the image. Short focal length camera captures a wider view while long focal length camera captures less view but the object in the image is magnified as compared to latter [16]. Principal point is the central reference of the image and describes the location of other points within the image. If the points are deviated from this point then distortions in the image occurs. Distortion Coefficient refers to the extent to which light bends within the lens, which may lead to radial or the tangential distortion. Radial distortion make the straight lines to appear as curve while the tangential distortion make the image tilted or stretched [17].

Extrinsic parameters [4] explain the positioning of the camera relative to the world coordinates system. The extrinsic parameters are: translation vector and rotation matrix. Translation vector and Rotation matrix both represent the whereabouts of the camera in the real world by a vector and a matrix respectively which is important for tasks such as target tracking, robotics, and augmented reality where high robot accuracy is important.

Following the extraction of the object points and image points, the estimation of intrinsic parameters is done. There is an OpenCV function, `cv2.calibrateCamera()` to determine the intrinsic parameter.

```
ret, camera_matrix, dist_coeffs, _, _ = cv2.calibrateCamera(obj_points,
    img_points, gray.shape[::-1], None, None)
```

**Listing 4:** Code to find Intrinsic Parameter

After the above code is run, it returns the values of intrinsic parameters consisting optical center, focal length, and distortion coefficient. Similarly, the extrinsic parameters are determined using the OpenCV function i.e. `cv2.aruco.estimatePoseSingleMarkers()`.

```
ret, rotation_vector, translation_vector = cv2.aruco.estimatePoseSingleMarkers(
    corners[i], marker_size, camera_matrix, dist_coeffs)
```

**Listing 5:** Code to find Extrinsic Parameter

This algorithm returns the values of the extrinsic parameters including rotation vector, translation matrix, and euler angles (roll, pitch, yaw). Now using these parameters, the camera is calibrated and thus captures the distortion free images.

[4]For more information about Intrinsic and Extrinsic parameters: `https://towardsdatascience.com/what-are-intrinsic-and-extrinsic-camera-parameters-in-computer-vision-7071b72fb8ec`

# 5 Implementation

## 5.1 Coordinate Transformation

Initially, when the point cloud data are collected using LiDAR, the data are in the LiDAR's coordinate system but for the calibration between the LiDAR and the camera, these LiDAR coordinates must be transformed into the camera coordinate system and also extrinsic parameter (translation and rotation) must be considered during the transformation since there is an offset in the position between these two sensors.

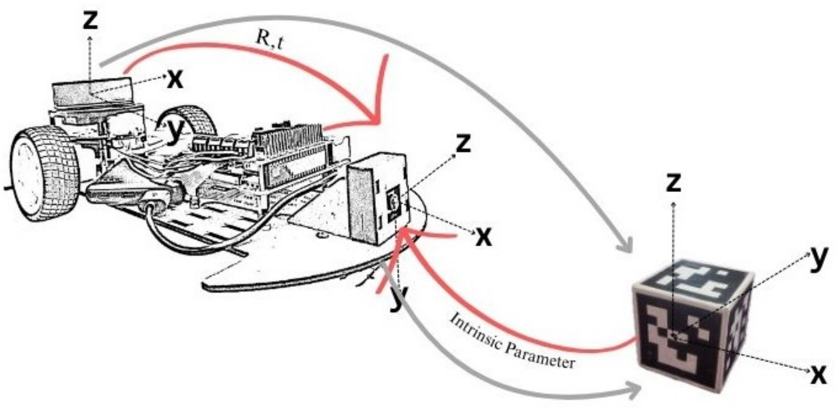


**Figure 18:** Relationship Between Intrinsic and Extrinsic Parameters

### 5.1.1 Translation

The translation vector gives the relative position x, y, and z within which the LiDAR is located with respect to the camera. This vector also gives the offset between the positions of the LiDAR and camera and shifts the origin of the LiDAR to the origin of the camera coordinates.

Let,

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{5.1}$$

Where:

- $t_x$ = translation in x axes
- $t_y$ = translation in y axes
- $t_z$ = translation in z axes

### 5.1.2 Rotation

Here, the Rotation means the position of the LiDAR with respect to the camera. Looking at the (figure 5), it can be seen that there is some offset of LiDAR in regard to the camera. The rotation matrix is derived from the Euler angles (roll, pitch, and yaw). These angles are used to rotate the LiDAR coordinate system to align with the camera's coordinate system. These angles are generally given in degrees and need to be converted to radians using the ZYX convention.

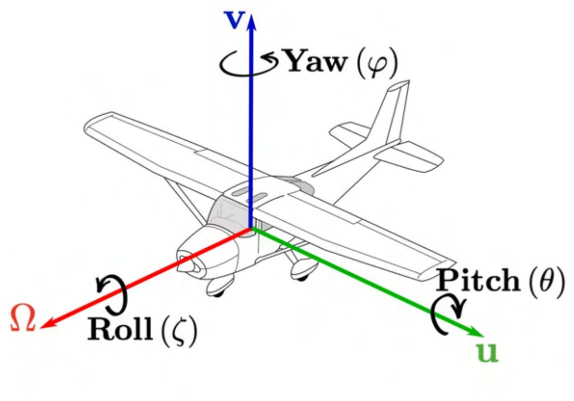


**Figure 19:** Roll, Pitch, and Yaw
Source: [18]

#### Rotation Matrix Construction

The rotation matrix is made by combining the Euler angles. The rotation matrices for each axis are:

**Roll (around the x-axis):**

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

**Pitch (around the y-axis):**

$$R_y = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

**Yaw (around the z-axis):**

$$R_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now, the rotation matrix can be determined as:

$$R = R_z \cdot R_y \cdot R_x \tag{5.2}$$

By applying this combined rotation matrix, the LiDAR coordinate system is aligned with the camera's coordinate system in terms of orientation.

## 5.2 Projection onto the Image Plane

### 5.2.1 Polar to Cartesian Conversion

The 2D LiDAR gives the 2D data in the form of the distance and angle. Initially, the data collected is in Polar coordinates. However, in order to effectively project these point cloud data onto the camera image, these polar coordinates must be converted into Cartesian coordinates because they are more easy to project as they define the point in two dimensional plane, x coordinate for horizontal plane and y coordinate for vertical plane.

This conversion from Polar to Cartesian coordinates can be done using trignometric functions:

$$x_{\text{LiDAR}} = d \cdot \cos(\theta)$$

$$y_{\text{LiDAR}} = d \cdot \sin(\theta)$$

$$z_{\text{LiDAR}} = 0$$

where

- $d$ = distance measured by the LiDAR
- $\theta$ = angle of the LiDAR measurement

### 5.2.2 Transform to Camera Coordinate Frame

Combination of the rotation matrix $R$ from ( 5.2) and the translation vector $t$ from ( 5.1) into a single transformation matrix will enable to transform any LiDAR point to the coordinate system of the camera. This can be expressed in mathematical form, using a $4 \times 4$ extrinsic matrix, $E$:

$$E = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \tag{5.3}$$

Here, $R$ is the 3x3 rotation matrix, $T$ is the 3x1 translation vector, and $0^T$ is a row vector of zeros to maintain the matrix dimensions.

To transform a LiDAR point

$$P_{\text{LiDAR}} = \begin{bmatrix} x_{\text{LiDAR}} \\ y_{\text{LiDAR}} \\ z_{\text{LiDAR}} \\ 1 \end{bmatrix}$$

into camera coordinate,

$$P_{\text{camera}} = \begin{bmatrix} x_{\text{camera}} \\ y_{\text{camera}} \\ z_{\text{camera}} \end{bmatrix}$$

it is multiplied by the extrinsic matrix $E$ from equation ( 5.3):

$$P_{\text{camera}} = E \cdot P_{\text{LiDAR}}$$

$$\begin{bmatrix} x_{\text{camera}} \\ y_{\text{camera}} \\ z_{\text{camera}} \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x_{\text{LiDAR}} \\ y_{\text{LiDAR}} \\ z_{\text{LiDAR}} \\ 1 \end{bmatrix} \tag{5.4}$$

Where:

- $R = 3 \times 3$ rotation matrix.
- $\mathbf{t} = 3 \times 1$ translation vector.
- $(x_{\text{camera}}, y_{\text{camera}}, z_{\text{camera}})$ = 3D coordinates of the lidar points in the camera coordinate frame.
- $(x_{\text{LiDAR}}, y_{\text{LiDAR}}, z_{\text{LiDAR}}, 1)$ = homogeneous coordinates of the LiDAR point.

By applying this extrinsic matrix to all LiDAR points, they are rotated and translated in a way that they lie within the camera's coordinate system. This transformation is crucial to help project LiDAR points on the 2D image plane which will be further explained in the following part of the process.

### 5.2.3 Project onto Camera Image

Up until now, the coordinates are still in camera coordinates. Now, in order to project it into the image, these coordinates must be transformed into pixel coordinates. In order to do so, the intrinsic parameter of the camera is very important. Intrinsic parameters is composed of the focal length, principal point and distortion coefficient. It combines the focal length and the optical center into 3×3 matrix which is further used to transform 3D camera coordinates into pixel in the image plane. The matrix is given by:

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

Here,

- $K$ is Intrinsic matrix
- $f_x$ and $f_y$ are the focal lengths.
- $c_x$ and $c_y$ are the coordinates of the optical center.

The 2D LiDAR points are projected to the image plane coordinate using intrinsic matrix $K$. The 3D coordinate of the camera are transformed to the 2D pixel coordinates by using the following equation:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x_{\text{camera}} \\ y_{\text{camera}} \\ z_{\text{camera}} \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{camera}} \\ y_{\text{camera}} \\ z_{\text{camera}} \end{bmatrix}$$

This results in:

$$u = \frac{f_x x_{\text{camera}}}{z_{\text{camera}}} + c_x \tag{5.5}$$

$$v = \frac{f_y y_{\text{camera}}}{z_{\text{camera}}} + c_y \tag{5.6}$$

Where:

- $(u, v)$ = the pixel coordinates in the camera image.
- $(c_x, c_y)$ = the principal point (optical center).
- $(f_x, f_y)$ = the focal length of the camera.

### 5.2.4 Scaling and Translation for Image Dimensions

After the obtained 2D LiDAR points are projected on the image plane, it is sometimes required to scale these coordinates so that it fits within the image properly. This often includes scaling and translating the pixel coordinates for better positioning.

#### Scaling Factors

Cameras and LiDAR sensors have different resolutions that may vary from each other. The camera can take high resolution images while LiDAR has different spatial resolution. Since, the LiDAR points have to be projected onto the camera image, it is mandatory to scale its coordinates with the resolution of the camera.

#### Horizontal and Vertical Scaling

Scaling is done separately to the horizontal and vertical axes, x and y axes respectively. Two scaling factors ($s_x$ and $s_y$) are used to specify how much they should be scaled in the x-axis and y-axis direction.

The scaling transformation for a point $(u, v)$ can be represented as:

$$u' = s_x \cdot u$$
$$v' = s_y \cdot v$$

Here,

- $u'$ = scaled pixel coordinate in the x-axis
- $v'$ = scaled pixel coordinate in the y-axis
- $s_x$ = scaling factor in the x-axis
- $s_y$ = scaling factor in the y-axis

Since, I am using 2D LiDAR in this research, the LiDAR scans only in the horizontal direction. Therefore, I utilized only the horizontal scaling.

### Translation Offsets

Similarly, during the calibration, sometimes deviations can be found such as- the optical centre of the camera not aligned with the image sensor centre or if there are shifts between the two systems. In such condition, the projected points are translated to help the LiDAR points to correspond with the image plane perfectly.

### Horizontal and Vertical Translation

The horizontal and vertical translation of the previously scaled coordinates $(u', v')$ can be showed as:

$$u'' = u' + t_x$$
$$v'' = v' + t_y$$

Here,

- $u''$ = translated pixel coordinate in the x-axis
- $v''$ = translated pixel coordinate in the y-axis
- $t_x$ = translation factor in the x-axis
- $t_y$ = translation factor in the y-axis

The equation combining both the scaling and translation offset can be written as:

$$u'' = s_x \cdot u + t_x \tag{5.7}$$

$$v'' = s_y \cdot v + t_y \tag{5.8}$$

This equation ( 5.7) and ( 5.8) can be used to adjust the pixel coordinates $(u, v)$ in the image after scaling and translating. Since, I am using 2D LiDAR in this research, I again used only the horizontal translation.

### 5.2.5 Removing Out-of-Frame Points

Depending on how the camera is oriented, some of the points which need to be projected onto the camera image may lie outside the frame of the camera image. It would be better for these out-of-frame points to be removed to include only visible points.

The x-coordinate of a pixel coordinates should be in the range of the image width, while the y-coordinate of a pixel coordinate $(v)$ should be within the height of the image.

This can be expressed as:

$$0 \leq u < \text{image width}$$

$$0 \leq v < \text{image height}$$

Now, the filtering is done in order to remove the points that have failed to remain within the desired margin of the image. For this, each projected point is iterated and checked whether the corresponding point $(u, v)$ lies inside the image border.

## 5.3 Calibration Method

### 5.3.1 Calibration Dataset

#### Image and LiDAR Data Collection

For the calibration process, as previously explained in (see 4.1.1) and (see 4.1.2), I took about 20 images from camera and at the same time same number of point cloud data was collected respectively from the LiDAR. These data were collected and were later used for calibration. This dataset serves an essential role for the further calibration of LiDAR and Camera. Similarly, it was important to have the variety of orientation and distances of the reference point both in images and LiDAR data to facilitate the calibration process.

The image captured for the calibration are:

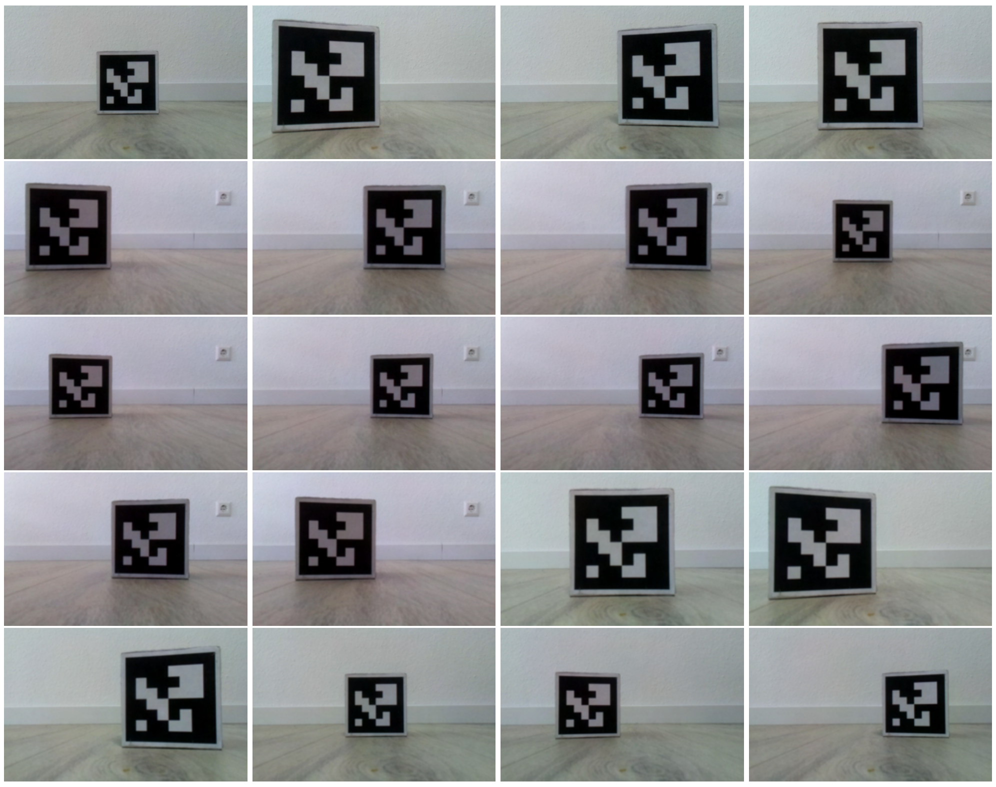
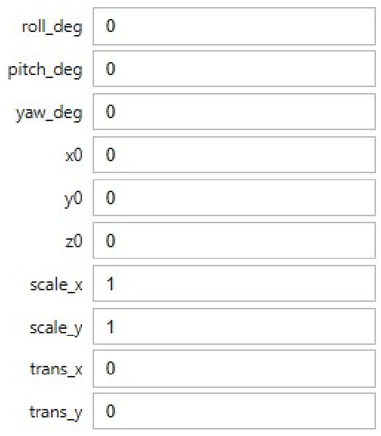


**Figure 20:** Images from the calibration dataset

### 5.3.2 Parameters

Several parameters are required in order to calibrate the LiDAR and Camera, such as :

- roll_deg: Roll
- pitch_deg: Pitch
- yaw_deg: Yaw
- $x_0, y_0, z_0$: Translation vector
- scale_x, scale_y: Scaling (x and y axis)
- trans_x, trans_y: Translation (x and y axis)



| | |
|---|---|
| roll_deg | 0 |
| pitch_deg | 0 |
| yaw_deg | 0 |
| x0 | 0 |
| y0 | 0 |
| z0 | 0 |
| scale_x | 1 |
| scale_y | 1 |
| trans_x | 0 |
| trans_y | 0 |

**Figure 21:** Parameters

Combining the parameters roll_deg, pitch_deg, yaw_deg, the rotation matrix ($R$) is created as mentioned in (5.1.2) and using the parameters $(x_0, y_0, z_0)$, the translation vector $(t)$ is created as in (5.1.2). Additionally, scale_x, scale_y are used for scaling as mentioned in (5.2.4.1) and trans_x, trans_y are used for translation as in (5.2.4.2).

### 5.3.3 Evaluation

**LiDAR Terminal Points ($Q$) and Reference Points ($P$)**

In this research, I am using ArUco marker as a reference for the calibration between LiDAR and camera. When the LiDAR scans the surrounding, it collects a large number of point cloud data including the point cloud data from the ArUco marker. Lets consider the point cloud data only from the reference (ArUco marker box). Now, if the end points of these point cloud which represents the edges of the box is extracted, these points are called terminal points. Reference points are the points on the edge of the ArUco marker box where the terminal points are expected to be. The goal of the calibration is to have the teminal points as close as possible to the reference points in all images of the calibration dataset.

These points play a huge role to determine boundaries of the object to which the LiDAR sensor is scanning (in this case, ArUco marker). This boundary information is important for the appropriate matching and calibration with the camera. Another reason of using terminal points is for measuring the calibration error. By noting the terminal points and comparing them with a known reference point, we can measure the deviation and therefore be able to determine the degree of calibration of the tools.

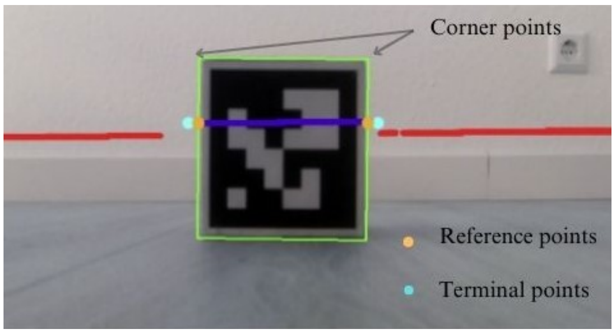


**Figure 22:** Terminal and Reference Points

```python
def get_calibration_points(marker_corners, lidar_points):
    lidar_left_point, lidar_right_point = lidar_points
    top_left, top_right, bot_right, bot_left = marker_corners
    paired_points = [
        (top_left, bot_left, lidar_left_point),
        (top_right, bot_right, lidar_right_point),
    ]
    cal_points = []
    for corner_point_a, corner_point_b, lidar_point in paired_points:
        lidar_x, lidar_y = lidar_point
        line_x = find_x_coordinate(corner_point_a, corner_point_b, lidar_y)
        # y-coord of line is limited by the upper corner
        line_y = min(lidar_y, max(corner_point_a[1], corner_point_b[1]))
        cal_point = [line_x, line_y]
        cal_points.append(cal_point)
    return np.array(cal_points)
```

**Listing 6:** Algorithm to find reference point

**Calibration error**

The calibration between the LiDAR and camera mean that the point cloud data of LiDAR should be properly aligned with the camera image. In order to validate whether the calibration is successful or not, the position of the reference point and the terminal point is checked. Hence, the calibration error is a measure of difference between the position of terminal points of LiDAR in relationship to the reference points on the ArUco marker in the camera image.

## Calculation

In order to calculate the calibration error, the following steps are undertaken: First of all, the pixel coordinates of the reference point ($Q$) and the terminal point ($P$) are obtained. There are two terminal points and reference points when the LiDAR pixel coordinates are projected onto the image. Let's consider the left edge of the Aruco marker in the image. Here the $L2$ distance is calculated by using the coordinates of the left terminal point and left reference point. Now using the $L2$ norm error formula, the error is calculated which gives the calibration error between projected LiDAR point and reference point on the basis of Euclidean distance.

The formula to calculate the L2 norm error on the left side can be written as:

$$E_{\text{left}} = \sqrt{(P_{1x} - Q_{1x})^2 + (P_{1y} - Q_{1y})^2}$$

The formula to calculate the L2 norm error on the left side

$$E_{\text{right}} = \sqrt{(P_{2x} - Q_{2x})^2 + (P_{2y} - Q_{2y})^2}$$

Total Calibration Error for One Image

$$E_{\text{total}} = E_{\text{left}} + E_{\text{right}}$$

$$E_{\text{total}} = \sqrt{(P_{1x} - Q_{1x})^2 + (P_{1y} - Q_{1y})^2} + \sqrt{(P_{2x} - Q_{2x})^2 + (P_{2y} - Q_{2y})^2}$$

Mean Calibration Error for 20 Images

$$\text{Mean Calibration Error} = \frac{1}{20} \sum_{i=1}^{20} E_{\text{total},i}$$

where,

- $P_{1x}, P_{1y}$ = coordinates of the left terminal points
- $P_{2x}, P_{2y}$ = coordinates of the right terminal points
- $Q_{1x}, Q_{1y}$ = coordinates of the left reference points
- $Q_{2x}, Q_{2y}$ = coordinates of the right reference points

### 5.3.4 Automatic Calibration

In order to optimize the parameters used for calibration automatically, I used Bayesian Optimization [5] in this thesis research. It repeatedly search for the best values of the parameters to align the LiDAR coordinate to the image coordinate. It uses the experiences from the already searched parameter spaces to estimate the next parameters which have a higher confidence of having a lower calibration error.

The parameters that I optimized using Bayesian Optimization are:

1. `roll_deg`: Roll
2. `pitch_deg`: Pitch
3. `yaw_deg`: Yaw
4. `x0`, `y0`, `z0`: Translation vector
5. `scale_x`: Scaling (x axis)
6. `trans_x`: Translation (x axis)

```python
study = optuna.create_study(
    study_name="lidar_param_study",
    storage="sqlite:///lidar_param_1.db",
    load_if_exists=True,
    direction="minimize",
    sampler=optuna.samplers.TPESampler(n_startup_trials=100, n_ei_candidates
    =200),
)
for _ in range(8):
    study.optimize(obj_function, n_trials=5000, show_progress_bar=True)

    trial = study.best_trial
    print("Best Score: ", trial.value)
    print("Best Params: ")
    for key, value in trial.params.items():
        print("  {}: {}".format(key, value))
```

**Listing 7:** Code snippet of using Bayesian Optimization

[5] Bayesian optimization is a sequential design strategy for global optimization of black-box functions that does not assume any functional forms. Source: `https://en.wikipedia.org/wiki/Bayesian_optimization`.

In this research, Bayesian optimization utilized TPE [6] (Tree-structured Parzen Estimator) sampler which was applied to minimize the total calibration error while working on the calibration of parameters. The TPE sampler provides the refined parameters' values in each iterations within the predefined ranges.

Before the use of Bayesian optimization, I tried to manually select the values to the parameters for the calibration. After several trial and errors, I found some values which showed better results. And when I implemented the Bayesian optimization with TPE sampler, I considered those values found from manual selection and created a range for each parameter which served as a parameter space for the TPE sampler. And I had almost 40k iterations so, the TPE sampler explored to find the better and optimized values for the parameters within this range in each iteration. As mentioned in (5.3.3), the mean calibration error was determined. The TPE sampler iterates through each parameter and find the best values in order to minimize this error as much as possible.

```python
def obj_function(trial):
    # Optimized parameters
    roll_deg = trial.suggest_int("roll_deg", 70, 100)
    pitch_deg = trial.suggest_int("pitch_deg", 260, 280)
    yaw_deg = trial.suggest_int("yaw_deg", 170, 200)
    x0 = trial.suggest_int("x0", -15, 50)
    y0 = trial.suggest_int("y0", -15, 50)
    z0 = trial.suggest_int("z0", -15, 50)
    scale_x = trial.suggest_float("scale_x", 1.0, 3.0)
    trans_x = trial.suggest_int("trans_x", -300, 0)
    scale_y = 1.0
    trans_y = -50
    return calibrate(roll_deg, pitch_deg, yaw_deg, x0, y0, z0, scale_x, scale_y
    , trans_x, trans_y)
```

**Listing 8:** Parameter space for TPE Sampler

The database is also created which will store the searched parameters and their respective error values. This ensures that the already explored parameter spaces are not explored again when we change some parameter spaces and start the parameter search again.

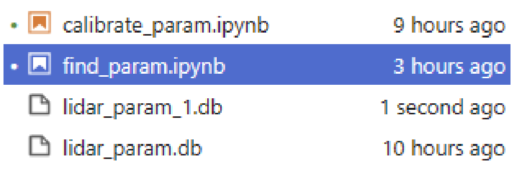

calibrate_param.ipynb 9 hours ago
find_param.ipynb 3 hours ago
lidar_param_1.db 1 second ago
lidar_param.db 10 hours ago

**Figure 23:** Database

[6]For more information:
https://optuna.readthedocs.io/en/stable/reference/samplers/generated/optuna.samplers.TPESampler.html
https://tech.preferred.jp/en/blog/multivariate-tpe-makes-optuna-even-more-powerful/.

# 6 Results and Discussion

## 6.1 Evaluation of Optimized Calibration Parameters

The values of the calibration parameters that were obtained applying the Bayesian Optimization based on the TPE Sampler is evaluated in this section. Several key parameters were optimized and the following table compares the initial parameter values with the optimized values found through the Bayesian optimization process:

| Parameter | Initial Search space | Optimized Value |
|---|---|---|
| roll_deg | 90-120 | 98 |
| pitch_deg | 260-300 | 280 |
| yaw_deg | 170-200 | 171 |
| x0 | -15 to 20 | -5 |
| y0 | 10 to 40 | 24 |
| z0 | 10 to 50 | 17 |
| scale_x | 1.0 to 3.0 | 1.8822123 |
| trans_x | -300 to 0 | -153 |

**Table 1:** Comparison of Initial and Optimized Calibration Parameters

Here, the Bayesian optimization was able to efficiently determine the set of parameters that yielded the least calibration error. The optimized values made huge changes in calibration showing the necessity of optimization for each parameter.

## 6.2 Comparison of Calibration Error Before and After Optimization

The table below summarizes the calibration errors for 20 images, both before and after optimization:

| Image Number | Calibration Error (Before) | Calibration Error (After) |
|---|---|---|
| 1 | 467.09 | 33.53 |
| 2 | 493.92 | 44.60 |
| 3 | 446.61 | 48.54 |
| 4 | 478.30 | 8.52 |
| 5 | 457.66 | 22.79 |
| 6 | 492.85 | 4.74 |
| 7 | 491.07 | 16.73 |
| 8 | 484.08 | 26.32 |
| 9 | 434.06 | 670.20 |
| 10 | 526.47 | 25.44 |
| 11 | 550.46 | 31.45 |
| 12 | 508.46 | 13.17 |
| 13 | 495.44 | 6.63 |
| 14 | 460.83 | 24.22 |
| 15 | 433.38 | 27.17 |
| 16 | 455.20 | 13.61 |
| 17 | 433.68 | 56.05 |
| 18 | 452.02 | 491.91 |
| 19 | 402.48 | 643.76 |
| 20 | 501.47 | 7.78 |

**Table 2:** Calibration Errors Before and After Optimization for 20 Images

## 6.3 Visual Representation of Calibration Improvements

In this section, images are presented to show the effectiveness of optimizing calibration parameters for the alignment of LiDAR and image data.

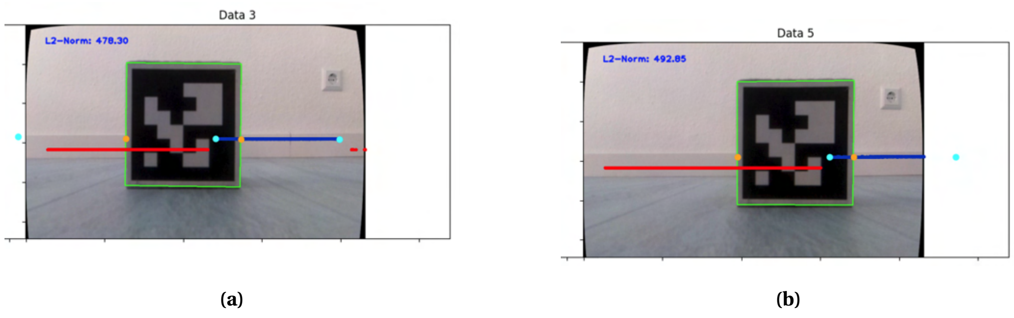


**Figure 24:** Manual Calibration

These images are taken before the optimization using Bayesian optimization. It can be clearly seen that the LiDAR points are not properly projected over the image and also the calibration error is high.

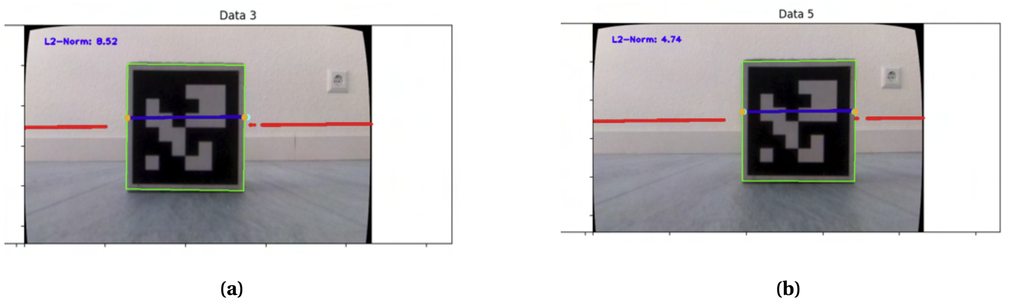


**Figure 25:** Automatic optimized Calibration

These images are taken after using the optimized parameters. It can be clearly seen that the projection of LiDAR point cloud data is much better than the previous one. Similarly, the calibration error is comparatively lower too.

## 6.4 Discussion

### 6.4.1 Evaluation

As mentioned in the (Table 2), it can be seen that some images still had more calibration error even after the optimization. Some of such cases can be seen via images below:

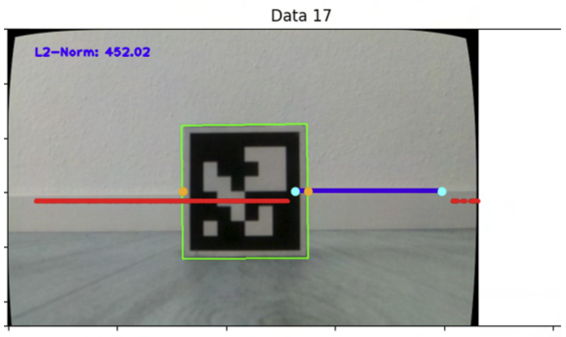


**Figure 26:** Error Before

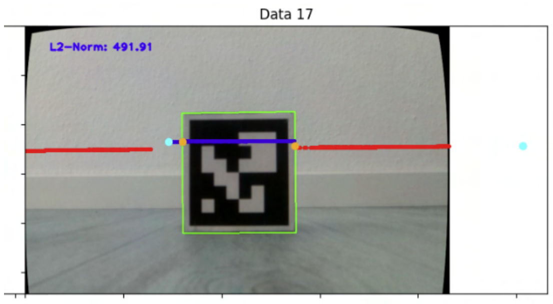


**Figure 27:** Error After

While collecting the LiDAR data, the ArUco marker was placed at a close distance to facilitate easy correspondence. Since 2D LiDAR was used, it provided information about both distance and angle. When analyzing the point cloud data, the nearer distance typically corresponds to the data gathered from the ArUco marker due to its proximity to the LiDAR setup.

During the extraction of terminal points, a measure of $x_{\text{min}}$ was obtained to represent the minimum distance from each LiDAR data point. Subsequently, a range around $x_{\text{min}}$, often defined as $x \leq x_{\text{min}} + 10$, was createdto filter out the lidar points (highlighted as blue points in Figure 27) corresponding to the aruco marker. Within this range, the minimum and maximum values of $x$ were determined to facilitate the identification of terminal points for LiDAR.

However, this approach had some limitations. In cases, such as the one depicted in Figure 27, outliers (red points) were found within the predefined range. Consequently, the terminal points were not properly determined, resulting in a high calibration error despite accurate calibration compared to the other scenario.

### 6.4.2 Performance Analysis

#### Processing Time

When measuring the time taken for calibration process, several attributes were taken into account, including transformations of LiDAR points, projecting them onto the camera image, time taken for optimizing the parameters and so on. Among all, a huge amount of the time was taken to optimize the parameter.

Bayesian optimization was used to optimize the eight parameters. The TPE sampler iterated for almost 40k iteration which is quite a massive number in order to find the refined parameters‘ values. However, this number can be brought down to lower number but having a large number of iterations generally mean that it will determine more accurate parameters to align the LiDAR data onto the camera image plane. Also, the images that were used were twenty in number. The number of images can also be made less but I opted for large number of images. The reason for that is I took the images keeping the reference i.e. ArUco marker at variety of orientation and distances. Having variations in the positions of the reference helps to pinpoint the position in both the camera image and LiDAR data and hence more accurate correspondence can be determined for better Calibration.

Similarly, other task like loading data of both camera and LiDAR was faster and the task involving LiDAR point conversion from polar to Cartesian coordinates and finally to pixel coordinates, where rotation matrix, translation vectors, scaling, translation and so on were implemented also didn‘t take much time.

# 7 Conclusion

## 7.1 Summary of Key Findings

This research has provided various insights related to the calibration of camera and 2D LiDAR. First of all, I setup a robust calibration approach for the alignment of camera images with LiDAR point clouds by the use of ArUco markers. For this, multiple number of images were captured from camera and the same amount of LiDAR data were collected. Then after the point cloud data were collected in polar coordinates which were then converted into Cartesian coordinates for easy transformation and projections. This conversion was very helpful to map LiDAR data in relation to coordinate system of camera. Again these Cartesian coordinates of LiDAR were transformed into the camera coordinate frame with the help of extrinsic parameters. This transformation was an important process for accurately projecting LiDAR points onto the image plane of the camera.

Later on the process, the LiDAR data once again were transformed into the pixel coordinates using the intrinsic parameters of the camera. The combined 3*3 matrix of focal length and optical center were multiplied to the LiDAR coordinates and thus generated pixel coordinates were projected onto the image. Scaling and translation were performed in the pixel coordinates for better alignment with the image. Filtering was also used to exclude the LiDAR coordinates which were beyond the camera FoV or behind the camera plane. In order to optimize calibration parameters, Bayesian optimization method in combination with the Tree-structured Parzen Estimator (TPE) sampler was used. This approach enabled a search for a set of best calibration parameters by minimizing the calibration error within a parameter space. Bayesian optimization with the TPE sampler provided a better parameter values in each iterations based on the previous values. Each iterations gave better values than the prior parameter resulting increased accuracy and reliability in the calibration process of parameters such as roll, pitch, yaw, translation and so on.

After the projection, the calibration accuracy was estimated by calculating the L2 norm error, which represents the difference between the terminal points of the LiDAR points and the reference points on the image. It was found that after the Bayesian optimization was used, the calibration error became comparatively lower than before.

## 7.2 Future Works

In the future, there are lots of possibilities to research, develop and make this calibration process more effective. First and foremost, the reference point is very crucial for the accurate calibration. In this thesis, I used a simple handmade ArUco marker box made out of paperboard and because of that it didn‘t have proper dimension. This box was fine to perform the calibration with but for more accuracy, it is suggested to use more standard and robust box. Future researches may use a more stable framework, like a 3D-printed cube of known dimensions and later on the ArUco marker can be glued in each side of the box.

Similarly, the type of sensors employed in this research may be inadequate despite serving the necessary requirements for carrying out this type of research for initial experiments. In the future, it is advisable to use the sensor of high quality and efficiency. For example: The LiDAR that I am using is capable to gather data only from limited small distance. Also when I collected the point cloud data, I had to create an algorithm to take 100 LiDAR rotations to get more dense point cloud data with high accuracy without leaving any spots. It was quite time consuming so, if possible, use high performance cameras and LiDAR sensors with enhanced characteristics of calibration in further researches in order to achieve more accurate and reliable results of calibration in shorter time.

Moreover, the approach followed to extract terminal points directly from the LiDAR data in my research was prone to some level of error or inaccuracy. In further research, it is better to investigate other methods of terminal point extraction which are more sophisticated than the conventional ones in order to increase precision of the calibration phase. There is also more room for improvement associated with determining the calibration parameters. Future research can use more complex optimization algorithms or approaches for more precise parameter. Also currently I captured 20 images and same amount of LiDAR data which is quite high. In the future, a technique can be determined to find the accurate results even with less number of data.

# A Appendix

## A.1 Extrinsic Parameters between Camera and LiDAR

| Parameter | Symbol | Value |
|---|---|---|
| **Euler Angles (Roll, Pitch, Yaw)** | | |
| Roll | $roll\_deg$ | 98° |
| Pitch | $pitch\_deg$ | 280° |
| Yaw | $yaw\_deg$ | 171° |
| **Translation Vector** | | |
| x-axis | $x_0$ | -5 |
| y-axis | $y_0$ | 24 |
| z-axis | $z_0$ | 17 |
| **Scaling** | | |
| x-axis | $scale\_x$ | 1.8822123 |
| y-axis | $scale\_y$ | 1 |
| **Translation** | | |
| x-axis | $trans\_x$ | -153 |
| y-axis | $trans\_y$ | -50 |

**Table 3:** Parameters with their Symbols and Values

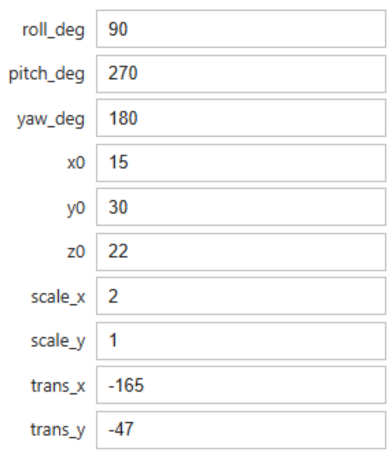
## A.2 Intrinsic Camera Parameters

| Camera Matrix (Intrinsic Parameters) | |
|---|---|
| **Parameter** | **Value** |
| Focal Lengths $(f_x, f_y)$ | $[713.21467978, 715.01592731]$ |
| Optical Center $(c_x, c_y)$ | $[419.63045252, 292.91949813]$ |
| **Distortion Coefficients** | |
| **Coefficient** | **Value** |
| $k_1$ | $0.3072799$ |
| $k_2$ | $-1.55346432$ |
| $p_1$ | $0.00711509$ |
| $p_2$ | $-0.00481935$ |
| $k_3$ | $2.69346288$ |

**Table 4:** Camera Intrinsic Parameters and Distortion Coefficients

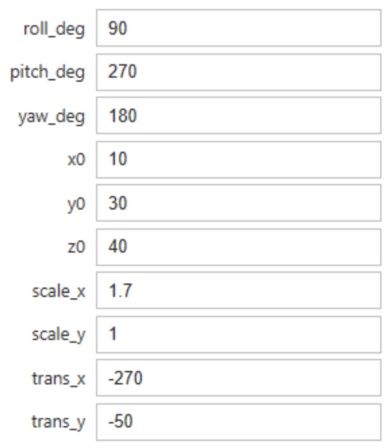
## A.3 Manually determined parameters

### Some early projections



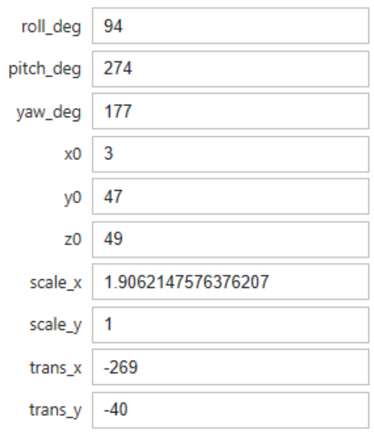
| | |
|---|---|
| roll_deg | 90 |
| pitch_deg | 270 |
| yaw_deg | 180 |
| x0 | 15 |
| y0 | 30 |
| z0 | 22 |
| scale_x | 2 |
| scale_y | 1 |
| trans_x | -165 |
| trans_y | -47 |

(a)



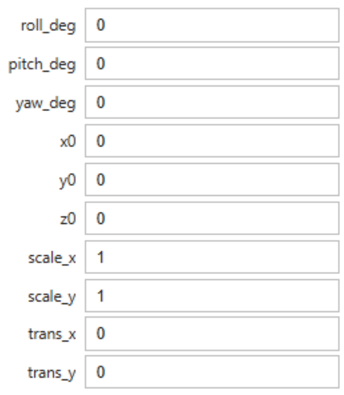
| | |
|---|---|
| roll_deg | 90 |
| pitch_deg | 270 |
| yaw_deg | 180 |
| x0 | 10 |
| y0 | 30 |
| z0 | 40 |
| scale_x | 1.7 |
| scale_y | 1 |
| trans_x | -270 |
| trans_y | -50 |

(b)



| | |
|---|---|
| roll_deg | 94 |
| pitch_deg | 274 |
| yaw_deg | 177 |
| x0 | 3 |
| y0 | 47 |
| z0 | 49 |
| scale_x | 1.9062147576376207 |
| scale_y | 1 |
| trans_x | -269 |
| trans_y | -40 |

(c)



| | |
|---|---|
| roll_deg | 0 |
| pitch_deg | 0 |
| yaw_deg | 0 |
| x0 | 0 |
| y0 | 0 |
| z0 | 0 |
| scale_x | 1 |
| scale_y | 1 |
| trans_x | 0 |
| trans_y | 0 |

(d)

**Figure 28:** Some early projections with errors

## A.4 Code Snippets

1.

```python
import cv2
import numpy as np

aruco_dict = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250)
parameters = cv2.aruco.DetectorParameters_create()

image = cv2.imread('C:/Users/chhet/Desktop/data_1.jpg')

if image is None:
    print("Error: Could not open or find the image.")
    exit()

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
corners, ids, rejected_img_points = cv2.aruco.detectMarkers(gray, aruco_dict,
    parameters=parameters)

if ids is not None:
    for i, corner in enumerate(corners):
        corner = corner.reshape((4, 2))
        for point in corner:
            point = tuple(map(int, point))
            cv2.circle(image, point, 5, (0, 0, 255), -1)
            text_position = (point[0], point[1] - 10)
            cv2.putText(image, f"{point}", text_position, cv2.
    FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 1, cv2.LINE_AA)

        int_corners = np.int32(corner)
        cv2.polylines(image, [int_corners], isClosed=True, color=(0, 255, 0),
    thickness=2)

        centroid = np.mean(corner, axis=0)
        centroid = tuple(map(int, centroid))
        cv2.putText(image, f"ID: {ids[i][0]}", centroid, cv2.
    FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2, cv2.LINE_AA)

cv2.imshow('Image with ArUco Marker Corners', image)
cv2.imwrite('C:/Users/chhet/Desktop/de.jpg', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

**Listing 9:** Algorithm to detect ArUco Marker

2.

```
import cv2 as cv
import numpy as np
import glob
import os

intrinsic_params_file = '/home/pi/Desktop/Thesis/PiCarProject/PiCar/Camera/
    Intrinsic/intrinsic_params_1.npz'
intrinsic_params = np.load(intrinsic_params_file)
camera_matrix = intrinsic_params['camera_matrix']
dist_coeffs = intrinsic_params['dist_coeffs']

images = glob.glob('C:/Users/chhet/Desktop/data_1.jpg')
objpoints = []
imgpoints = []

rvecs_array = []
tvecs_array = []

save_directory = '/home/pi/Desktop/Thesis/PiCarProject/PiCar/Camera'
os.makedirs(save_directory, exist_ok=True)

for fname in images:
    img = cv.imread(fname)
    h, w = img.shape[:2]
    undistorted_img = cv.undistort(img, camera_matrix, dist_coeffs, None,
    camera_matrix)
    mapx, mapy = cv.initUndistortRectifyMap(camera_matrix, dist_coeffs, None,
    camera_matrix, (w, h), 5)
    remapped_img = cv.remap(img, mapx, mapy, cv.INTER_LINEAR)
    undistorted_save_path = os.path.join(save_directory, 'undistorted_image.jpg
    ')

    cv.imwrite(undistorted_save_path, undistorted_img)

    cv.imshow("Undistorted Image", undistorted_img)
    cv.waitKey(1000)
    cv.destroyAllWindows()

    # Print save confirmation
    print(f"Undistorted image saved to {undistorted_save_path}")
```

**Listing 10:** Algorithm to undistort the image

## A.5 Additional Figures

1. **Some early projections**

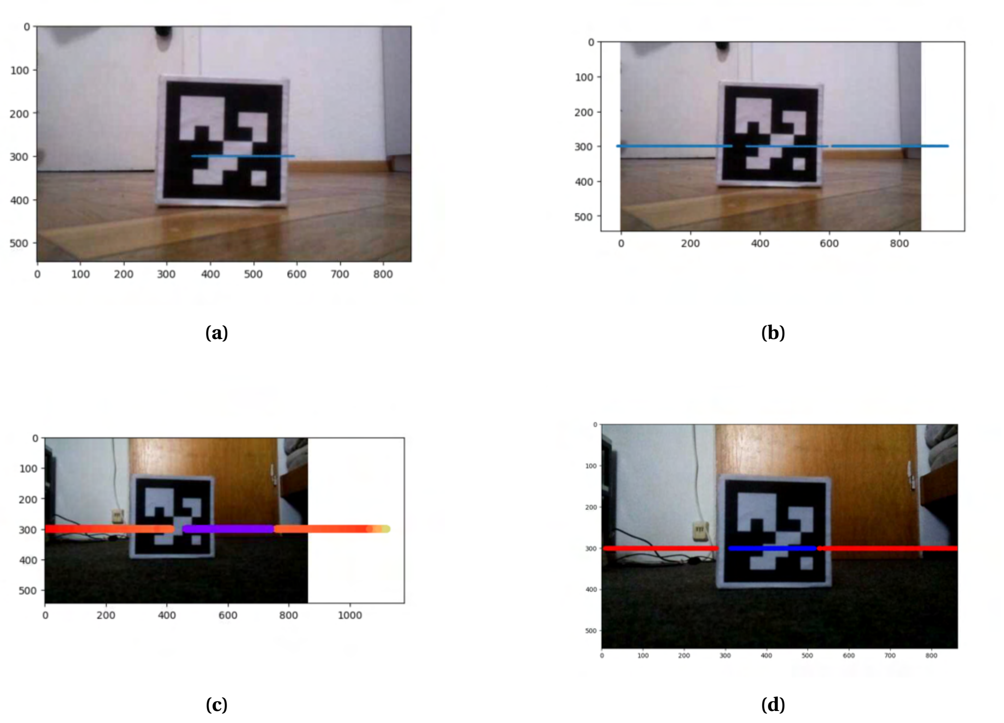


**Figure 29:** Some early projections with errors

## 2.Calibration before optimization

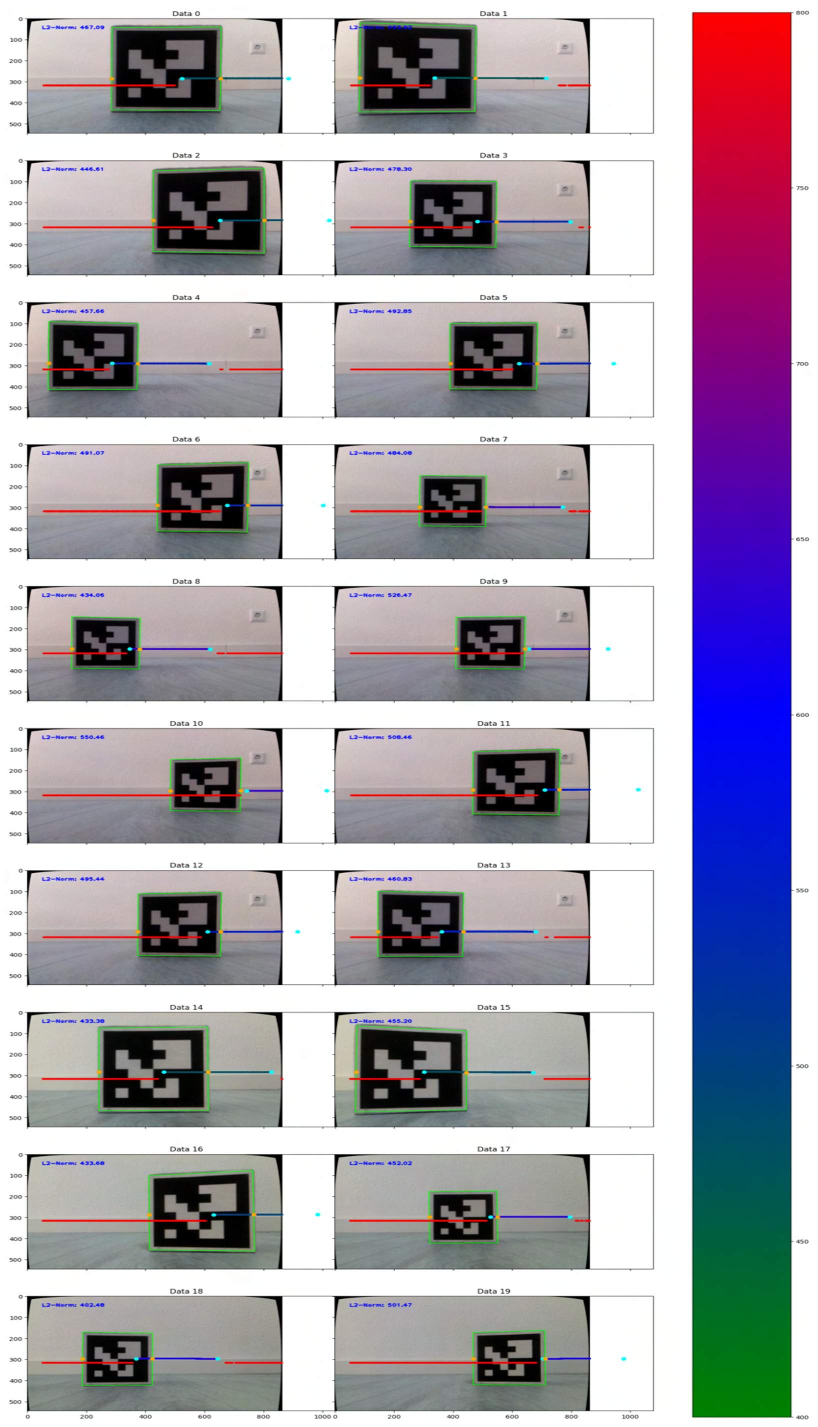


**Figure 30:** Calibration before optimization

2.Calibration after optimization

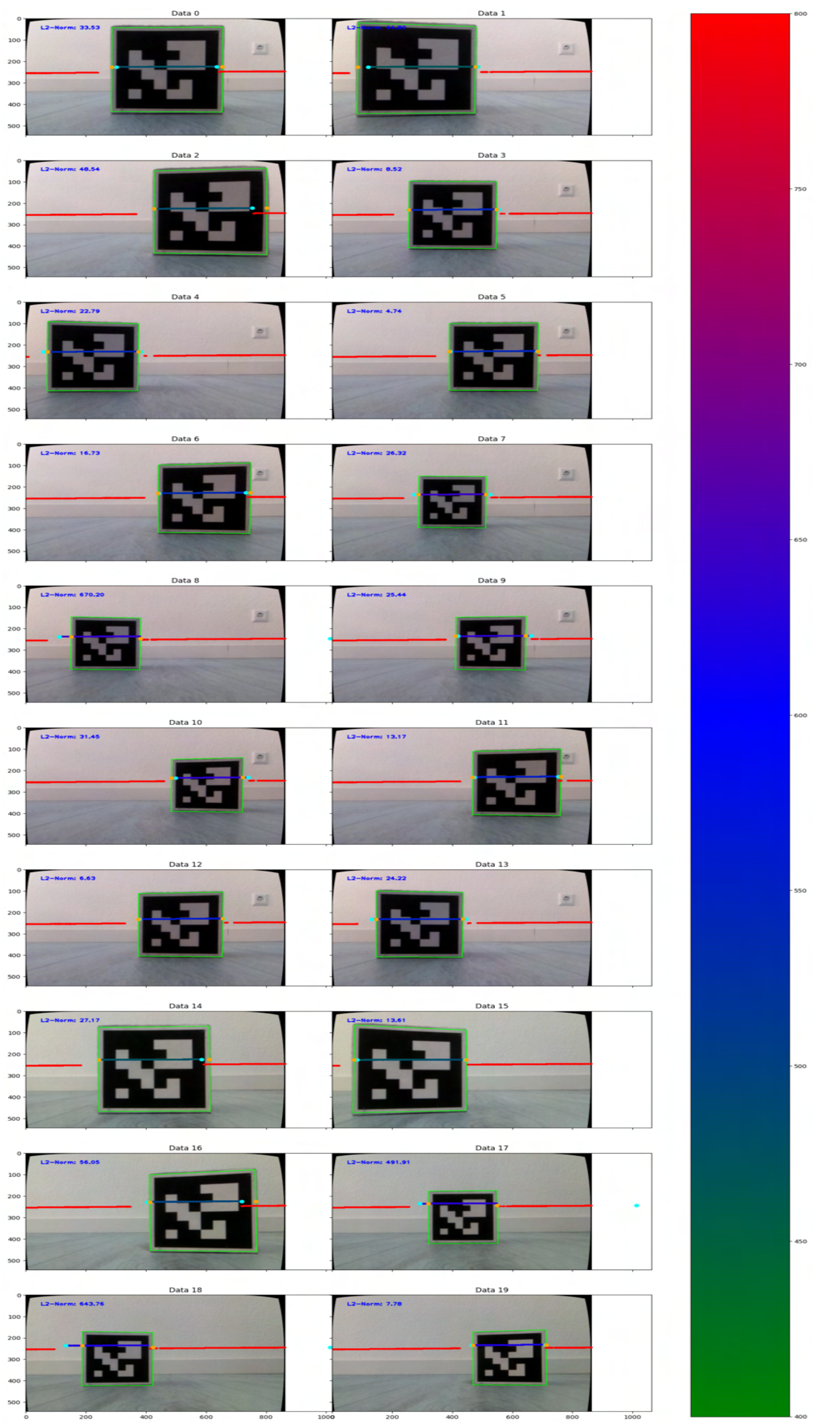


**Figure 31:** Calibration after optimization

## References

[1] Eung-Su Kim and Soon-Yong Park. Extrinsic calibration between camera and lidar sensors by matching multiple 3d planes. *Sensors*, 20(1):52, 2019. doi: 10.3390/s20010052. URL https://www.mdpi.com/1424-8220/20/1/52.

[2] Pei An, Tao Ma, Kun Yu, Bin Fang, Jun Zhang, Wenxing Fu, and Jie Ma. Geometric calibration for lidar-camera system fusing 3d-2d and 3d-3d point correspondences. *Opt. Express*, 28(2):2122–2141, 2020. doi: 10.1364/OE.381176. URL https://doi.org/10.1364/OE.381176.

[3] Wenzhong Wang, Ken Sakurada, and Nobuo Kawaguchi. Reflectance intensity assisted automatic and accurate extrinsic calibration of 3d lidar and panoramic camera using a printed chessboard. *Remote Sensing*, 9(3):851, 2017. doi: 10.3390/rs9080851. URL https://doi.org/10.3390/rs9080851.

[4] Jiunn-Kai Huang and Jessy W. Grizzle. Improvements to target-based 3d lidar to camera calibration. *IEEE Access*, 8(4):134101–134110, 2020. doi: 10.1109/ACCESS.2020.3010734. URL https://doi.org/10.48550/arXiv.1910.03126.

[5] Jae-Yeul Kim and Jong-Eun Ha. Automatic extrinsic calibration of a camera and a 2d lidar with point-line correspondences. *IEEE Access*, 11(5):76904–76912, 2023. doi: 10.1109/ACCESS.2023.3298055. URL https://ieeexplore.ieee.org/document/10190560.

[6] Ankit Dhall, Kunal Chelani, Vishnu Radhakrishnan, and K. Krishna. Lidar-camera calibration using 3d-3d point correspondences. (6), 05 2017. URL https://arxiv.org/abs/1705.09785.

[7] Xingchen Li, Yuxuan Xiao, Beibei Wang, et al. Automatic targetless lidar-camera calibration: A survey. (7), September 2022. PREPRINT (Version 1) available at Research Square https://doi.org/10.21203/rs.3.rs-2018540/v1.

[8] Xiyuan Liu, Chongjian Yuan, and Fu Zhang. Targetless extrinsic calibration of multiple small fov lidars and cameras using adaptive voxelization. *IEEE Transactions on Instrumentation and Measurement*, 71(8):1–12, 2022. doi: 10.1109/TIM.2022.3176889. URL https://arxiv.org/abs/2109.06550.

[9] Chanoh Park, Peyman Moghadam, Soohwan Kim, Sridha Sridharan, and Clinton Fookes. Spatiotemporal camera-lidar calibration: A targetless and structureless approach. *IEEE Robotics and Automation Letters*, 5(9):1556–1563, 2020. doi: 10.1109/LRA.2020.2969164. Available at https://doi.org/10.48550/arXiv.2001.06175.

[10] Rplidar a1, 2023. Datasheet available at https://www.slamtec.ai/wp-content/uploads/2023/11/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf.

[11] Waveshare rplidar a1 omnidirectional acquisition, July 2022. URL https://www.amazon.de/Waveshare-RPLIDAR-A1-Omnidirectional-Acquisition/dp/B0B6B5MWSJ. Accessed: 2024-05-08.

[12] Raspberry Pi. Camera. https://www.raspberrypi.com/documentation/accessories/camera.html, 2024.

[13] Raspberry pi camera module 8mp v2. URL `https://www.berrybase.de/raspberry-pi-camera-module-8mp-v2`. Accessed: 2024-05-08.

[14] Christian Cawley. How upgrading your raspberry pi 4 to bullseye will make it run faster, 2024. URL `https://www.makeuseof.com/how-upgrading-your-raspberry-pi-4-to-bullseye-will-make-it-run-faster/`. Accessed: 2024-05-30.

[15] amishakirti6410. Distortion, 2023. URL `https://www.geeksforgeeks.org/calibratecamera-opencv-in-python/`. Accessed: 2024-05-10.

[16] SmugMug. Understanding camera focal length and why it matters, March 2021. URL `https://news.smugmug.com/understanding-camera-focal-length-and-why-it-matters-2c47863af186`.

[17] Kaustubh Sadekar. Understanding lens distortion, October 2020. URL `https://learnopencv.com/understanding-lens-distortion/`.

[18] Pierre Degond, Antoine Diez, and Mingye Na. Bulk topological states in a new collective dynamics model. (17), January 2021.