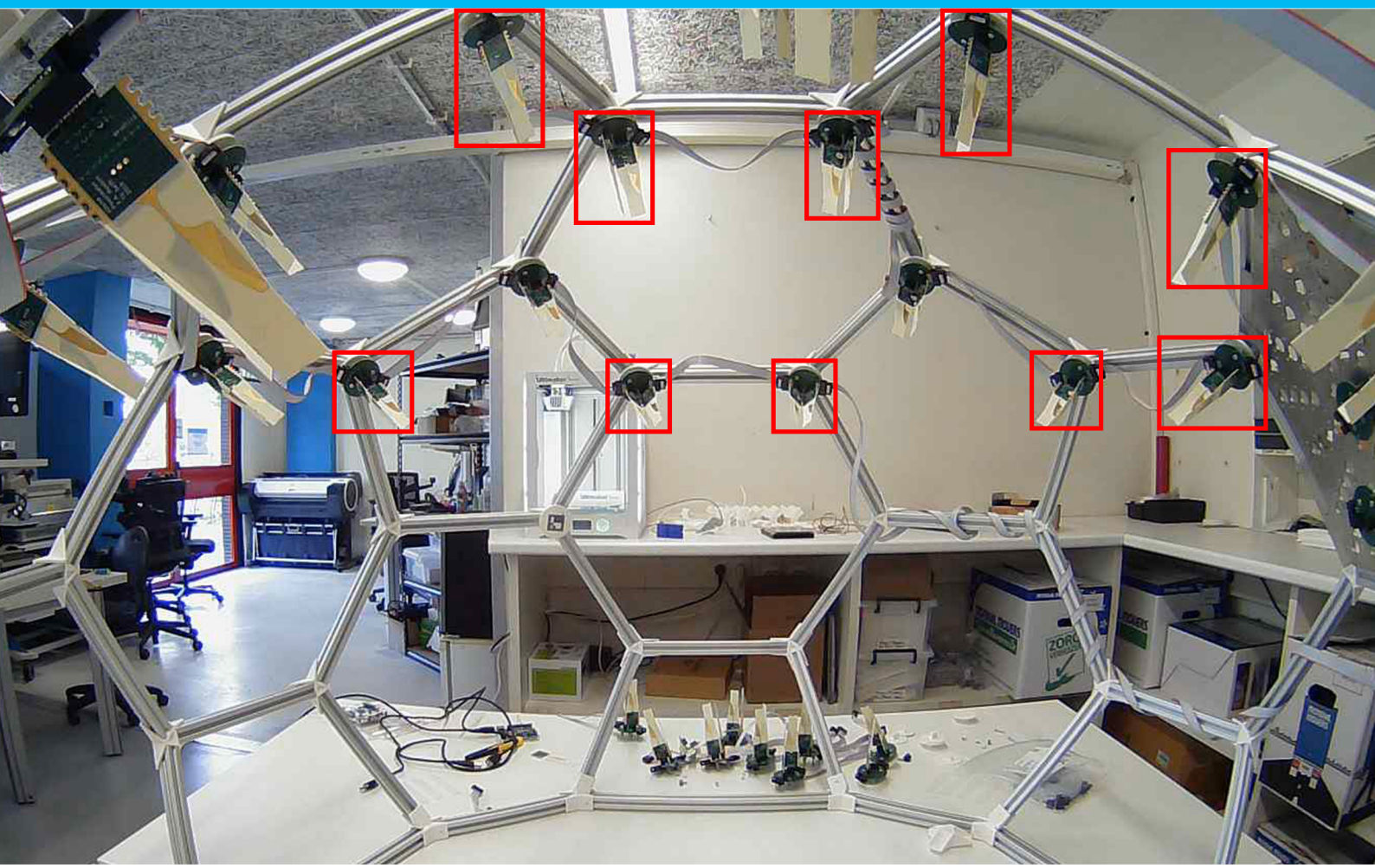
Optical Calibration System

Subproject within the ADome project



TU Delft

Daan Roos
Tycho van Velden

Optical Calibration System

Subproject within the ADome project

by

D. Roos & T. van Velden

to obtain the degree of Bachelor of Science in Electrical Engineering
at the Delft University of Technology.

Project duration:	April 19, 2021 – July 2, 2021
Thesis committee:	Prof.Dr. L.C.N. de Vreede (chair)
	Dr. M. Spirito
	Dr. M. Alonso del Pino
	MSc F.A. Musters
Thesis supervisors:	Dr. M. Spirito
	MSc F.A. Musters
	R.A. Coesoij

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

In the ADome project, a truncated icosahedron is used as a frame for multiple antenna nodes. These nodes are used to measure the radiation pattern of an antenna-under-test located in the center of this dome. Since these antennas can be placed anywhere in the ADome by the user, a calibration system is necessary to find the spherical coordinates of each of these antennas.

In this thesis, a method is proposed that uses computer vision algorithms, written using the OpenCV library in C++, to locate each of these antennas by detecting controllable LEDs attached to the PCBs of the antenna nodes. The found pixel locations will then be used to get the spherical coordinates of the antennas with respect to the camera. Finally these spherical coordinates will be transformed to fit to the coordinate system of the ADome itself, by detecting landmarks in the form of fiducial markers which are located at predetermined locations.

The methods of recognition of the antennas within using computer vision are discussed, implemented and tested on real and simulated data. The accuracy of finding the antennas on the real data was within 2 pixels from the true location.

The methods regarding estimating the location of the antenna with respect to the camera are discussed next. These methods include a distortion estimation and correction, after which each pixel will get there corresponding spherical angles θ and ϕ. These methods are also tested on simulated and real data, where the accuracy on the real data falls within 0.5 degrees in comparison to the true data.

Finally the methods of transforming the spherical angle θ and ϕ of the antenna with respect to the camera, to the coordinate system of the ADome are discussed. This starts with a method of recognizing fiducial landmarks (landmarks are accurately known reference points), where in this research there is chosen for ArUco fiducials. After this, the location of these landmarks can be used to determine the location and orientation of the camera within the ADome. The method of finding the distances between each of the landmarks and the camera, is discussed and tested and has an accuracy of 2 cm. The method of finding the location of the camera, and the method of finding the orientation of the camera are discussed, however this has not been tested and fully implemented yet. With this location and orientation the location of an antenna can be easily determined.

In conclusion, this thesis is a good starting point for designing an Optical Calibration System, which could make determining the location of each of the antennas faster, and more accurate.

Preface

For the last two months, we have performed research in order to achieve the degree of Bachelor of Science in Electrical Engineering. In order to achieve this, we have researched the possibility of improving one aspect of the already existing ADome project. Our task involves creating an automatic method of calibrating the antenna nodes in the ADome, such that each antenna knows its locations in spherical coordinates. These locations can later be used to create radiation patterns of antennas under test, placed in the center of the ADome.

Throughout this project, we have broadened our view by researching not only mathematical, but also Computer Science methods in order to realize a system capable of using computer vision to calibrate the antennas. The proposed system is already capable of locating the antennas relative to the camera, and therefore creates a great start for future research and a proof that calibrating the ADome using computer vision is a valid option.

Throughout the duration of the project, we had to deal with the consequences of the COVID-19 pandemic. This caused us difficulties testing certain aspects of the project, as this required access to the ADome which was difficult due to the restrictions set by the government. Nevertheless, we achieved finishing a project with promising results, which could lead us to proceed researching this topic.

We would like to express our gratitude to our supervisors, Ferry Musters and Marco Spirito, for offering us the support, advice, and tools necessary to get to this result. We also want to thank our supervisor Richard Coesoij, although not as experienced in this field and therefore not as involved as the other supervisors, offered us support throughout the weekly meetings. We would like to thank all the supervisors for offering support and ideas where needed, to tackle certain issues regarding the pandemic.

We also would like to thank Eric Bosman for lending us his ND-filter, allowing us to research the possibility of improving the accuracy of detecting the LEDs.

Finally we would like to thank our family and fellow colleagues Alexander James Becoy, Chris Bot, David Denekamp, Jakob Gilcher, and Remy Zhang for supporting us throughout this project and the last 3 years of our Bachelor Electrical Engineering.

D. Roos & T. van Velden
Delft, June 2021

Contents

1 Introduction

Developments in wireless communication ever move towards higher frequency bands. Most recently, the first frequency range of the 5G standard for telecommunication is starting to become more adopted into mainstream consumer technology, with frequencies still below 6 GHz. The second frequency range foreseen for 5G, however, will range into the millimeter-wave spectrum.

Devices for the second frequency range of 5G-NR are currently being developed. With this move to higher frequencies substantially more bandwidth will become available, allowing for higher data throughput. However, higher frequencies also complicate the development of the communication systems. At these higher frequencies, to compensate for the increased free-space loss and increase directivity of the system, antennas are often combined into arrays to use beamforming. These arrays are frequently designed as a monolithic entity to enable further miniaturization. This makes invididual testing of subsystems very complicated to impossible, increasing the relevance of over-the-air (OTA) testing.

In its current form, OTA characterization is a cumbersome process. The most used method involves near- or far-field scanning using a single probe that moves around the antenna-under-test (AUT). Another widely used technique is often known as a Compact Test Range (CATR), where a secondary feed antenna is used with a precisely manufactured reflector to create a region of plane waves where the AUT is placed. The AUT is then rotated in this field of plane waves to measure the response for different angles.

These methods both share weaknesses: they are relatively slow, due to only being able to measure a single point at a time. They require mechanical elements, increasing complexity and points of failure. Additionally, current testing methods typically use a network analyzer in their testing, requiring large lengths of RF cables to the AUT and probing elements.

To solve this problem, the ADome concept has been proposed. The ADome consists of detectors arranged in a half-sphere, each with its own antenna and RMS power detector, allowing the calculation of received power to take place at the node itself. It is a far-field measurement method, but the far-field distance at 5G mm-wave frequencies and above is sufficiently small that it comfortable fits into a regular laboratory. A picture of the structure of the current version can be seen in figure 1.1.

A number of projects were carried out by this group to improve the current ADome prototype. These projects are mostly disconnected from each other as they all consider a different part of the system.

The first project is a new readout protocol. As the sensing probes acquire the measurements locally, the need of a readout protocol becomes necessary for the computer to process these data. The current implementation in the ADome is limited in its functionalities. Antenna probes locations must be computed manually, making it not user-friendly and difficult to use in case of large number of sensing antennas. Furthermore, the probes are not able to store any information, such as its own location. Therefore, a new readout protocol using CAN is proposed where a micro-controller unit (MCU) is integrated at each probe to allow local sampling and readout along with non-volatile memory storage. Due to the presence of local MCUs, the potential functionalities of the ADome can be extended.

The second project concerns an optical calibration system. At the moment, the location of each sensor node needs to be measured and entered manually in the readout system. This project proposes a method to automatically determine the positions of the antennas in the dome. This is done using a

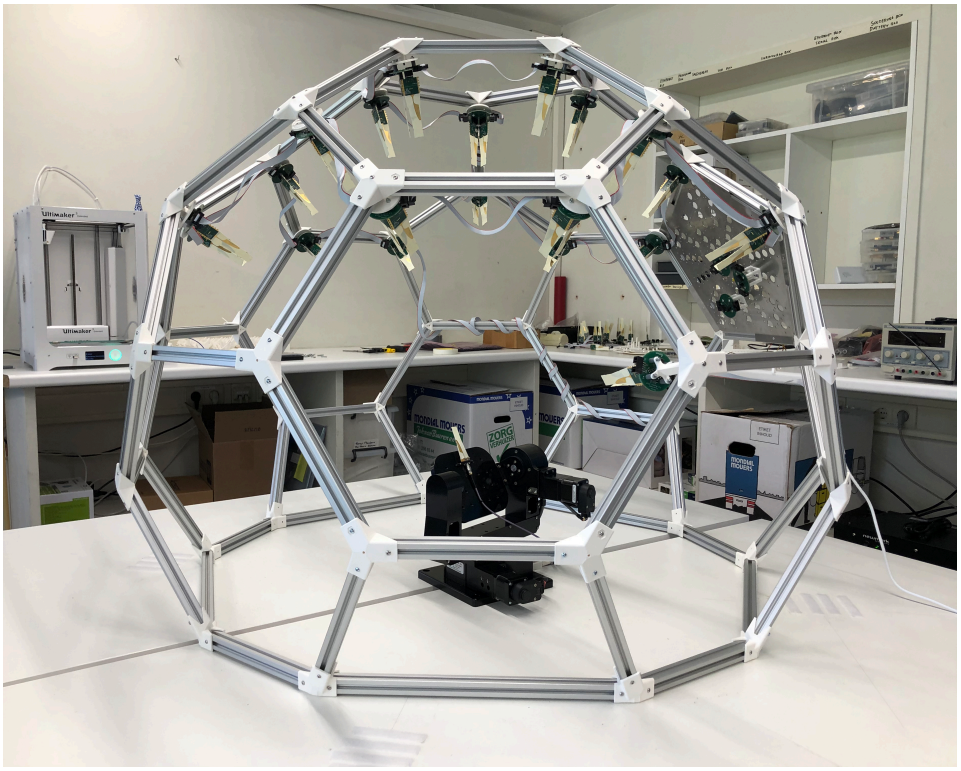


Figure 1.1: Structure of the ADome in development

camera mounted in the center of the ADome. The antennas will be recognized from the images taken by the camera using computer vision algorithms and the location in the images is mapped to a location in the dome.

The third project is an uncertainty assessment of the sensing nodes. The ADome sensing nodes use a RMS power detector to measure the received power. This project aims to characterize the measurement uncertainty of the detector for different excitations. A simulation testbench of the measurement setups is also created to allow assessing the impact of changes in the system without needing to change the physical setup. Characterizations is done for single- and multi-tone signals at different frequency and power levels.

This report details the second project.

In this report if 'antennas' are mentioned, the measuring antennas on the dome are mentioned, not the antenna of which the radiation pattern is going to be determined, unless mentioned specifically.

1.1. Objective

For the creation of the radiation pattern of an AUT the locations of the measuring antennas are very important, since these are the points used for the interpolation to eventually create the radiation pattern.

One of the key features of the ADome is its modularity. The ADome allows the user to use any amount of antennas, in any possible configuration on the dome. This feature leads to the opportunity of placing the antennas at every possible position. Since the configuration can change many times, and the locations of the antennas are necessary to know in order to do correct measurements, the ADome should be calibrated after the setup of a new configuration of antennas, to get the new locations of each of the antennas.

The current state of the ADome forces the user to manually measure the location of each of the antennas. Although this is possible with a low amount of antennas, it introduces unnecessary possibilities for errors, and is very time consuming. To improve this calibration phase, the objective was set to create a proof of concept for a new calibration method which should be able to calibrate the ADome automatically.

This new calibration method will consist of a computer vision system, which should be able to recog-

nize an antenna from an image and accurately determine its location on the dome in spherical coordinates. This system will from now on be referred to as the Optical Calibration System, OCS abbreviated. To determine the absolute location of the antenna within the ADome the OCS must perform three different steps:

1. the OCS must recognize the antenna (or a marker on the antenna) from an image;
2. the pixel coordinates of this antenna should be transformed to its spherical coordinates with respect to the camera
3. these spherical coordinates should be transformed to the coordinate system of the ADome. This transformation will be done according to the spherical coordinates with respect to the camera, and the relative location of several landmarks (reference points at predetermined locations).

The library used for the computer vision is OpenCV[1]. The choices made to use OpenCV will follow in Chapter 6.

The performance of the OCS will be determined by how accurate it can determine the spherical location of antennas within the ADome.

1.2. State of the art

The process of finding the locations in the ADome itself, is a very specific application, which does not have a lot of similar research. The process itself however can be compared to various other research subjects concerning object detection and object localization.

First of all, detecting the antennas. Powered by a computer vision system, the antenna recognition can be done is several ways. The most straight forward way is to perform object detection, this can be done by for example training a Haar detection model on images of the antenna[2]. A more sophisticated method is by training a Deep Neural Network to detect the antennas[3]. This however might be too much, since only one type of object has to be detected, not several different objects. Another method would be to perform edge detection, and detecting the edges of antennas[4]. These could then for example be used as features for a machine learning model. Finally, a different set of methods, more in the field of image processing, is by attaching markers to the antennas, and detecting the markers instead. One type of marker that could be detected would be a type of fiducial[5, 6]. This allows not only position, but also pose estimation (i.e. rotation). The issue with this marker however, is the limited amount of possible markers to be used, since each antenna would need its own marker. A different type of marker would be a color coded marker, such as an LED which could be turned on and off on command[7].

In the process of finding a proper mapping from pixel location to spherical coordinates with respect to the camera multiple camera calibration techniques are possible. Most of these methods consist of using fixed 2 or 3-dimensional patterns to determine the characteristics of the camera[8, 9]. There are also methods using 1-dimensional patterns[10], but these seem to be less accurate and harder to implement. Other methods of finding objects in a 3D space with respect to a camera system is by using a binocular system[11, 12]. Binocular systems use 2 cameras for each localization. The object will then be recognized in by both cameras, and the relation between the two will determine the location. Finally in the trend of localization, inspiration has been drawn from SLAM (Simultaneous localization and mapping) algorithms[13, 14]. These methods are primarily used for the self-localization of autonomous vehicles, domestic rovers, or even planetary rovers and mapping the surroundings around these vehicles.

1.3. Document structure

This document consists of 8 chapters including this introduction. Chapter 2 describes the programme of requirements, what the final deliverable must be able to accomplish and to what extend. Chapter 3 describes the methods of finding the antennas within the ADome, and the performance of finding those. Chapter 4 describes the methods of mapping the pixel location of an antenna or landmark to spherical coordinates with respect to the camera, and its corresponding performances. Chapter 5 discusses the methods of finding landmarks, and how to use these to transform the coordinates from relative to the camera to relative to the ADome. Chapter 6 describes how the Chapters 3 to 5 will be combined to

create a prototype, and the performance of this prototype, as well as some problems that arose during prototyping. Chapter 7 describes the discussion and conclusion of the whole report. This will also give some suggestions for further improvement/research.

2

Programme of requirements

This chapter will specify the specifications and requirements of the OCS. These specifications and requirements tell us what the OCS has to do, under what circumstances, and how it must eventually do this.

First the assumptions under which the OCS operates are listed, which state the circumstances under which the OCS operates. After this, the mandatory requirements are listed, which tell us what the OCS has to do and how this must be done. Finally the trade-off requirements are listed; these are the requirements which should be reached as much as possible, but are not mandatory.

2.1. Assumptions

In order to make the requirements achievable, a few assumptions had to be made.

1. The antennas can be located at any position on the ADome.
2. In each position of the camera, at least 3 landmarks are located within the camera's field of view.
3. The localization system has full control over the behaviour of the LEDs by communicating with the microcontrollers connected to them.

2.2. Mandatory requirements

The requirements that must be fulfilled by the OCS at the end of this research can be subdivided into two groups: the non-functional requirements and the functional requirements. The non-functional requirements define what the system must be able to achieve, while the functional requirements define how the system must achieve this.

1. **Non-functional requirements**
   (a) The OCS must locate the antennas within 0.1 degrees.
   (b) The OCS must be able to recognize any number of antennas within the ADome.
   (c) The OCS must work in both dark and light environments.
2. **Functional requirements**
   (a) The OCS must detect markers in the form of LEDs using computer vision.
   (b) The OCS must detect landmarks in the form of ArUcos using computer vision.
   (c) The OCS must return the location of the landmarks and the LEDs in spherical coordinates.
   (d) The OCS must return the orientation of the antenna.

2.3. Trade-off requirements

The trade-off requirements are the requirements which are not fully mandatory, but should still be reached as well as possible.

1. The OCS should be as accurate as possible.
2. The OCS should be as fast as possible.
3. The OCS should preferably work on a Raspberry Pi.
4. The OCS should preferably work using any wide angle camera.
5. The OCS should be able to pivot and tilt the camera automatically.
6. The OCS should be able to support any number of LEDs per antenna.

3

Antenna localization

In order to determine the locations of the antennas, a computer vision based localization system has to be designed. As this system needs to be able to determine the position of each antenna individually, it is of importance that the system is able to distinguish the antennas. In order to fulfill this requirement, each antenna will be fitted with a controllable marker, in this case an LED. Using this controllable marker it is possible for the system to detect each antenna separately, and allows the system to connect antennas to its respective location.

Since each LED emits light in the form of a color, the LED can be seen as a dot of its color. Therefore, a logical choice would be to detect these LEDs by detecting their color. This can be done by means of a color detection algorithm.

Throughout this chapter, the color detection algorithm itself will be explained, consisting of a so called blob detection algorithm. This algorithm consists of:

1. determining the correct color automatically
2. determining threshold values automatically
3. detecting all pixels in range of the to be detected color
4. combining pixels to form the 'blob' of each LED

After this, the implementation of the LED detection, which relies on the previously developed color detection, will be described.

Finally, each algorithm will be tested and discussed at the end of this chapter.

3.1. Blob detection

Each LED can be seen as a cluster of pixels of the color of the LED. Therefore, if each pixel in this cluster is detected, the LED can be found by looking at clusters of detected pixels. As mentioned in the introduction, a blob detection algorithm will be used to achieve this.

There are different types of blob detection algorithms, where the most common are the connected component labeling method (CCL)[15], the Laplacian of Gaussian method (LoG)[16], and the Difference of Gaussians method (DoG for short). The latter often being used as an approximation of the LoG[17].

Most methods have one thing in common, the input data they use. Normal images are mostly composed of 3 values per pixel, for example Red, Green, and Blue (RGB), or Hue, Value, and Saturation (HSV). This is useful to display the image, but it is too much data to perform blob detection. For instance, to allow the LoG method to work, data assigned to each pixel should be convolved with a Gaussian function. This works best when each pixel only contains 1 value. Therefore, for each method, the input image will be binarized according to specified requirements which will be explained in Section 3.2.3. After the binarization, the resulting data will be passed to the chosen blob detection method.

3.1.1. Laplacian of Gaussian method

The Laplacian of Gaussian method is one of the first and most common blob detectors used in computer vision. The system is based on convolving a Gaussian function with the original image and applying the Laplacian operator ∇^2 on the result. This results in positive responses for dark blobs, and negative responses for bright blobs.

For the method, the Gaussian function in Equation 3.1 is convolved with the data of the image $f(x, y)$ as shown in Equation 3.2[16]. The standard deviation σ in this Gaussian function is then related to the radius of the blobs, $r = \sqrt{2\sigma}$ where r is the radius of the blobs. Therefore, by increasing the value of σ, the radius can be increased.

$$G(x, y; \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{x^2 + y^2}{2\sigma^2} \tag{3.1}$$

$$L(x, y; \sigma) = f(x, y) * G(x, y; \sigma) \tag{3.2}$$

After the convolution, the Laplacian operator ∇^2 is applied to the result of Equation 3.2, see Equation 3.3. This results in a single-scale LoG blob detector[1][16]. The issue with this detector is, that it struggles with detecting multiple blobs of various radii. The result can already by improved by applying the 3σ rule such that 99% of the energy of a Gaussian is within three standard deviations of its mean, causing the formula for σ to change to $\sigma = (r - 1)/3$[16]. However, after this change, the system will still suffer from detecting multiple radii.

$$\begin{aligned} \nabla^2 &= \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \\ \nabla^2 L &= L_{xx} + L_{yy} \end{aligned} \tag{3.3}$$

To allow multiple radii to be detected, a so called multi-scale approach is necessary[16]. To get this multi-scale system, a type of processing of the image will have to be applied to obtain the correct scale selection. One way of obtaining such a multi-scale detector with automatic scale selection, is by using a so called scale-normalized Laplacian operator (see Equation 3.4) and to detect local scale-space extrema[18]. Using this method, the scale of the detected blob is determined by selecting a scale which results in the maximum LoG filter response[16].

$$\nabla^2_{\text{norm}} L = \sigma(L_{xx} + L_{yy}) \tag{3.4}$$

Although this detector is rather effective, its result is limited by the effects of the Gaussian function. Due to the shape of the function, the detector will only work on elliptical shapes, however most effectively on circular shapes[16]. Next to this, after applying a multi-scale system, post-processing on the result will still be necessary, as the system will generate multiple overlapping “blobs”.

3.1.2. Difference of Gaussians

The Difference of Gaussians method is often considered as an approximation of the Laplacian of Gaussian method[17]. This method uses the relationship that the scale-space function $L(x, y; \sigma)$ (as seen in Equation 3.2) satisfies the diffusion equation, see Equation 3.5. When substituting this relationship in Equation 3.4, and performing the approximation that for a small deviation of σ, $\partial_\sigma L = \frac{1}{\Delta\sigma}(L(x, y; \sigma + \Delta\sigma) - L(x, y; \sigma))$, Equation 3.6 is obtained.

$$\partial_\sigma L(x, y; \sigma) = \frac{1}{2}\nabla^2 L(x, y; \sigma) \tag{3.5}$$

$$\nabla^2_{\text{norm}} L(x, y; \sigma) \approx \frac{\sigma}{\Delta\sigma}(L(x, y; \sigma + \Delta\sigma) - L(x, y; \sigma)) \tag{3.6}$$

The result of this operation is similar to that of the LoG, the difference being that it is less operation intensive, therefore improving performance. Next to this, it allows tuning of two σ values, instead of 1. Similar to the LoG method, the method has issues with non elliptical shapes, causing these objects to be detected as multiple different blobs.

[1] A single-scale blob detector is a blob detector only capable of detecting one single size blobs, whereas a multi-scale blob detector is capable of detecting multiple different sizes.

3.1.3. Connected Component Labeling

A third method of detecting blobs, is by labeling pixels that are connected or close to each other, thereby identifying different sets of pixels as blobs. As this system has no mathematical operations, it is almost completely algorithm dependent. This means that the accuracy and efficiency of the detector is fully dependent on the design of the algorithms. Due to this, multiple different algorithms have been designed in the past by various authors, however their workings have a common structure[15].

First of all, the image is scanned to apply labels to pixels which are indicated to be part of a blob. After this, other pixels connected to pixels with the same values should be connected to blobs by assigning the same label to those pixels. Lastly, blobs that share the same value but have different labels should be merged, leaving one large blob with the same label.

The advantage of this method is, that it can be altered to meet the requirements set by the user. By default, the system is independent on the shape of the blob, as it only checks for connected pixels that have the same value as the original pixel. This allows for blobs to be detected that aren't perfectly elliptical, and are of various different sizes. This feature is useful in this scenario. Although LEDs seem to be a perfect circle, this only applies when looked at from above. Next to this, the light emitted by the LED does not have to be circular, especially when captured by the camera. Effects such as lens flare, over-saturation and capturing the LED under a certain angle cause the resulting image to be non-elliptical, as visible in Figure 3.1. Due to this, the previous methods, LoG and DoG, will have a more difficult time detecting the blobs.

As the result should be as accurate as possible, and should allow LEDs of any size, shape, or quality to be detected, the Connected Component Labeling method will be used as blob detection for this system.

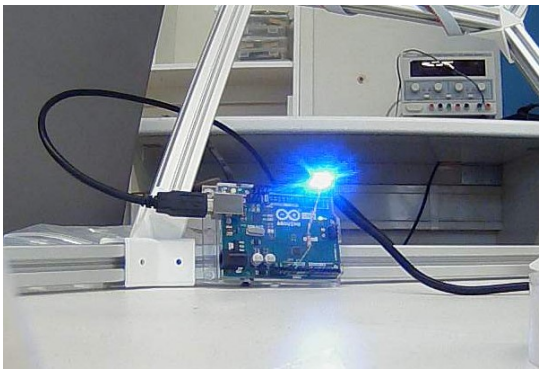


Figure 3.1: Image taken by the camera when aiming straight at the LED, showing the lens flare.

3.2. Algorithm design

As mentioned in Section 3.1, the method selected to detect the LEDs is the so called Connected Component Labeling (CCL) method. In this section, the algorithm, in particular the sub-algorithms, will be discussed that will be used to create the blob detector.

The algorithm will be divided into a few different parts. First of all, as mentioned in Section 3.1, the blob detector will need a binary image as input. This means that the original RGB image will have to be transformed to a dataset with 2 possible values, in this case a gray-scale image with values 0 and 255 (corresponding to black and white). After this, the data can be passed through the blob detection, the CCL method, to find all the clusters of the wanted color.

In order to transform the RGB image to its binary form, a form of pre-processing will be necessary. In this case, the image will be scanned for the wanted color, and each corresponding pixel should be set to 255. The part of selecting the corresponding pixels will be subdivided in an algorithm for selecting the correct color, determining thresholds automatically, and finally selecting the pixels that correspond to the range of colors selected by the threshold determination.

After the correct pixels have been selected, the clusters of the to be detected colors will be found. To perform this clustering of pixels, the CCL blob detector will be used.

The full algorithm, including taking the picture and returning the center coordinates of the detected clusters, can be seen in Figure 3.2.

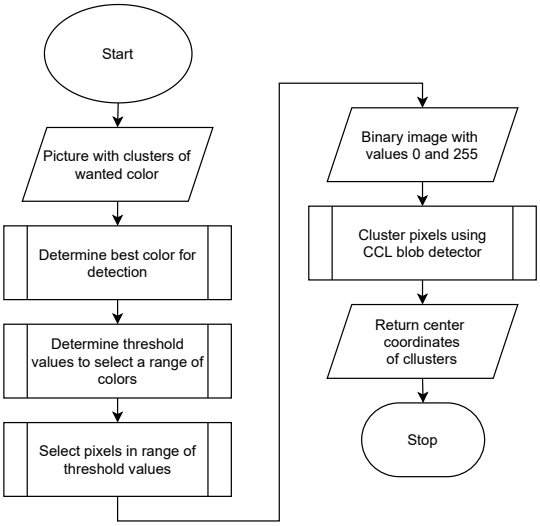


Figure 3.2: Flowchart describing the basic algorithm for detecting clusters of colors: `detectColor`

3.2.1. Automatic color determination

Although the user might want to detect one separate color, automatically determining what color matches this wanted color is a more difficult task than one might imagine. Different lighting circumstances, different camera sensors, all these play a factor in the way colors are captured on the images. Therefore, instead of giving an exact color to detect, one might want to find a color located in the image matching the most, and try to detect that color instead. Especially since it is possible to select a range of colors to detect later on, meaning that colors close to the wanted color can still be detected. To find the most matching color, a sub-algorithm should be designed that is capable of scanning the image for available colors and return the color that matches the most with the color the user wants to detect.

To fulfill this task, each RGB color is compared to the wanted color and a score is applied to the color. The color with the largest score will be used in the later steps of the color detection algorithm. For this task, a precise color is not that important. Especially since the algorithm will be dealing with an LED, and, as shown in Figure 3.1, the amount of light captured can have certain effects on the resulting image. Therefore, instead of choosing one color, each color will receive a score depending on the amount of, for example, blue a color is. This results in the calculation in Equation 3.7, where a score equation for each primary color is given. In this equation, R, G, and B are Red, Green, and Blue respectively. Since the RGB format is a 3 integer value where each element ranges from 0 to 255, meaning that the maximum score possible is equal to 765.

$$\text{Score}_{\text{Red}} = \text{R} + (255 - \text{G}) + (255 - \text{B}) \tag{3.7a}$$
$$\text{Score}_{\text{Green}} = (255 - \text{R}) + \text{G} + (255 - \text{B}) \tag{3.7b}$$
$$\text{Score}_{\text{Blue}} = (255 - \text{R}) + (255 - \text{G}) + \text{B} \tag{3.7c}$$

The algorithm will return a top 5 of most matching colors, in case the user wants to use the other 4 colors as well.

3.2.2. Automatic threshold detection

As mentioned in Section 3.2.1, the most matching color is selected, however, colors close to the selected ‘center’ color will still have to be selected. This is due to some pixels’ RGB values being off due to noise, or due to different lighting. Therefore, a range of colors has to be selected that will be detected instead of just one color. This range cannot be constant, some images might contain a lot less different colors due to for example different lighting, causing false positives if the range is too large. On the other hand, if the range is too small, it could happen that too little pixels will be selected, causing data to be missed, or locations to be off-centered.

To fix this problem, an algorithm will be designed that is able to select thresholds automatically, deciding for itself what range of colors is best. Before the workings of this algorithm will be explained,

the color-space used should be discussed.

Normally, pictures are described in the RGB color-space. As mentioned in Section 3.2.1, the RGB system contains 3 integer numbers ranging from 0 to 255, describing how much of each color is present in the current pixel. This is a useful system for computers to show images on screen, but selecting a range of colors using this system is difficult. First of all, using RGB means that the algorithm should have to determine 3 different ranges, one for each color channel, meaning that determining the best threshold values is a difficult process.

Therefore, instead of using RGB, the HSV color-space will be utilized. HSV stands for Hue Saturation Value, where Hue selects the main color used, Saturation how much of the color is seen (lower means more grey), and Value the brightness of the color (see Figure 3.3). Since Hue is seen as the base of the color, the value of Hue used in the range of colors can be rather constant. As visible in Figure 3.3, each “general” color is equally divided over the circle of colors. Meaning that if a center color is found using Section 3.2.1, its Hue value can have a constant value subtracted and added to have a correct range of colors. Another value that can have a constant range, is the “Value” value. Since this value only changes the brightness of the color, and not the color itself, this range can be each value possible, in this case 0 to 255.

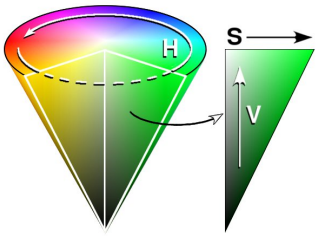


Figure 3.3: Representation of HSV color-space. Source:[19]

This means that the only value left that is able to change, is the Saturation. As mentioned above, the Saturation changes how much of a color is present in a pixel. The lower the value, the more grey the result. In different lighting conditions, colors can become more saturated, or less saturated, meaning that the range of Saturation values has to change accordingly. To tackle this problem, different images would be tested using the color detection algorithm. For each image, the threshold that would result in the best detection would be set manually. When comparing these threshold values to data of the image, a relationship between the mean of each Hue value of each pixel, and the threshold was found, as visible in Figure 3.4.

After this, a 4th order polynomial could be fitted through this data, making sure that when multiple thresholds per hue were possible, the lowest value was chosen. Although this means that certain images will have a lower threshold than possible, having a value that is too high for an image is worse, as this will lead to detection of noise in the images. The polynomial fit can be used as a function to determine the Saturation threshold needed for different average Hue values.

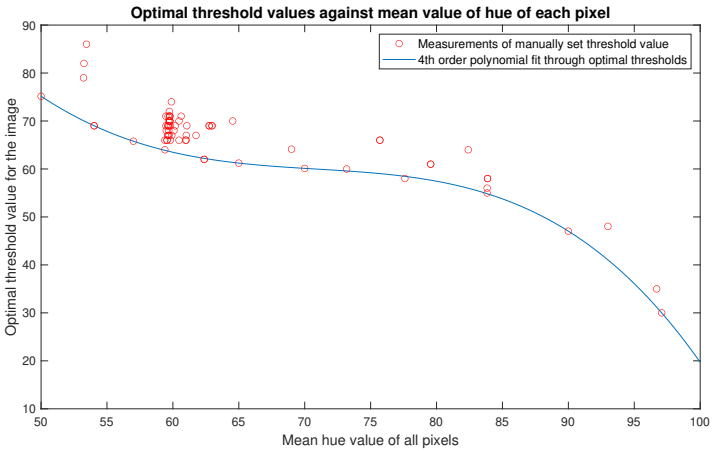
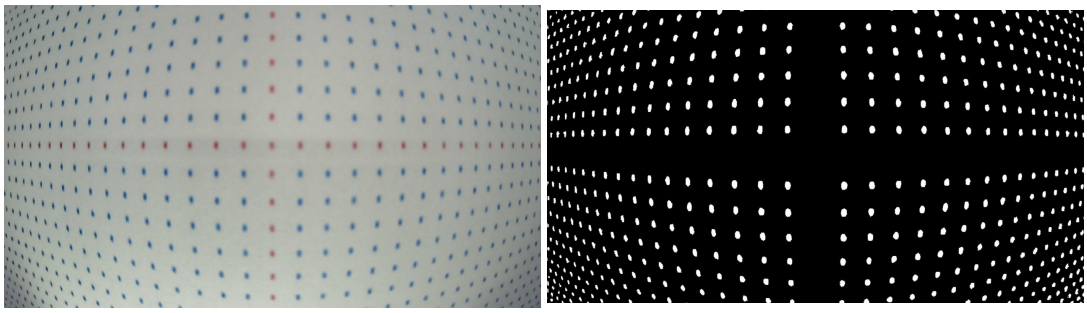


Figure 3.4: Image of polynomial fitted to measured threshold data

3.2.3. Pixel detection

After selecting the range of HSV values that should be detected, the image should be transformed into a binary image that can be used by the blob detection algorithm. As mentioned in the introduction of this section, the binary data will be a gray-scale image with values equal to 0 for pixels that do not appear in the range of colors, and a value of 255 that do appear in the range of color.

Performing this transformation is a rather simple process. First of all, a matrix is created the size of the image. After this, each pixel is compared to the upper and lower value of the range created in Section 3.2.2. If the value matches, the location of the pixel in the matrix will be set to 255, 0 otherwise. This will leave an image similar to Figure 3.5b.



(a) Picture of the grid used to test the color detection

(b) Binary image acquired after scanning the picture of the grid

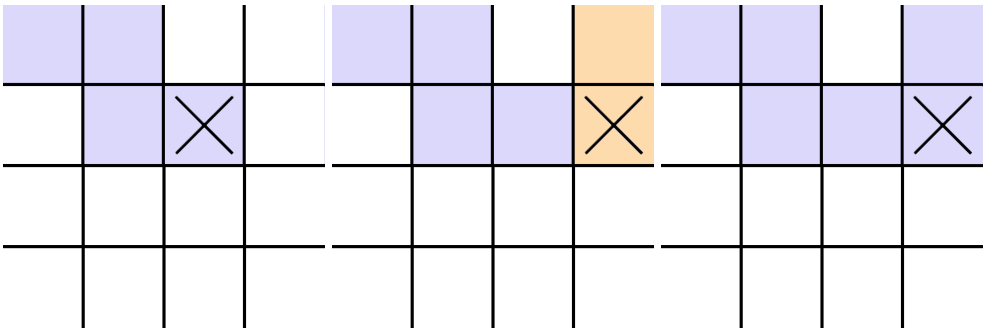
Figure 3.5: Picture of the grid before and after the pixel detection, original image is used later in Chapter 4 for calibrating the lens

3.2.4. Pixel clustering

As mentioned in the introduction of this chapter, the algorithm chosen to detect the LEDs is based on a color detection algorithm powered by a blob detection algorithm, the Connected Component Labeling method to be more precise. This method uses the binary image created in Section 3.2.3 to determine whether a pixel is of a matching color (its value is set to 255) or not (its value is set to 0), and clusters nearby pixels to form the blobs of the LEDs.

The main principle behind the algorithm designed to match the idea of the CCL method, is the use of so called pixel clusters. The pixel clusters are structures containing all the coordinates of the pixels part of the cluster, the number of pixels in the cluster, and the average value of the coordinates in the cluster resulting in the center of the cluster. Each time a pixel with the value 255 is found that cannot be stored in an existing cluster, a new cluster is created to store the current pixel.

To find whether pixels can be stored in an already existing cluster, the binary image is scanned for pixels with values equal to 255. Each time such a pixel is found, all pixels in a radius will be scanned to see whether any of those pixels are stored in a cluster. If such a pixel is found, the current pixel will be stored in the existing cluster, and the new center coordinates will be calculated. In case another cluster is located in range of the current pixel, while the current pixel already is part of a cluster, the clusters are merged resulting in one large cluster. A simplified representation can be seen in Figure 3.6.



(a) First pixels added to first cluster

(b) First pixels added to second cluster

(c) Merging neighbouring clusters

Figure 3.6: Process of creating clusters, each cluster indicated with different colored background, cross indicating the current pixel

When the full image has been scanned, all clusters that contain less pixels than an indicated threshold will be removed. These are clusters created by for example noise. The final list of found clusters, or in other words detected blobs, will be returned, with each cluster containing its center pixel position.

3.3. LED detection

Using the algorithm discussed in Section 3.2, the system is able to detect blobs of color in an image. The only issue is that this algorithm is only capable of detecting colors in perfect circumstances, such as the one shown in Figure 3.5a. When “noise” is introduced, such as object of similar color in the background, the algorithm will detect this as a false positive. Next to this, the algorithm will try to detect as many clusters as possible, this is non-ideal since, as mentioned in the introduction of this chapter, each antenna should be detected separately. This means that only one LED should be turned on at a time, and therefore only one blob should be returned by the color detection algorithm. This means that “noise” such as reflections of the LED should be filtered and only the location of the LED should be returned.

In this section, the implementation of combining the color detection with taking pictures and returning the pixel coordinates of an LED is discussed. This includes removing “noise” from the input image, and selecting the correct blob and returning its pixel coordinates. The system created in this section can be seen in Figure 3.7.

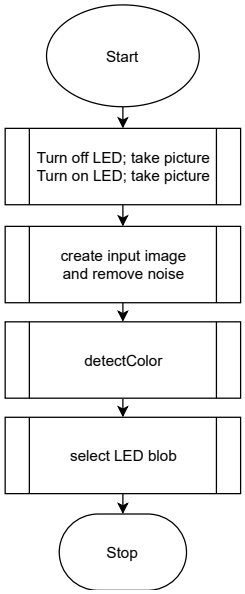
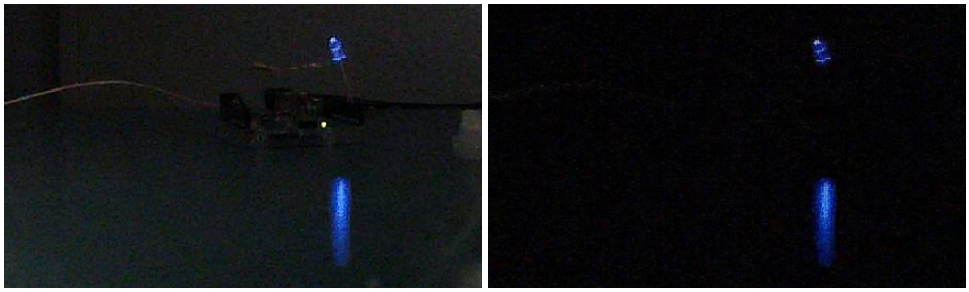


Figure 3.7: Flowchart describing system for detecting LED: `locateLED`

3.3.1. Strategy for removing background “noise”

As mentioned in Chapter 1, the antenna localization algorithm will be used on the inside of the ADome, a truncated icosahedron. In best circumstances, the inside of the dome is closed off from the outside world. In this case it is possible for the system to directly work with the color detection algorithm discussed in Section 3.2, since the only color equal to the one of the LED, is the LED itself. However, as mentioned in Chapter 2, the LED detection should work in light and dark environments. This means that it is possible for the dome to only exist of its frame, thus with its side-panels removed. This introduces the “noise” mentioned in this section’s introduction.

To remove this “noise”, a strategy has been designed which is able to remove a large amount of this noise. Since each LED is controllable due to the connection to each antenna’s microcontroller, it is possible for the LED detection algorithm to turn each LED on and off at request. This allows the LED detection to take two pictures in sequence, one with LED on and one with LED off. The RGB values of both images can then be subtracted, leaving an image such as Figure 3.8. In this way, the normal color detection can be used, as the only data left is noise introduced by the camera, and most importantly the LED itself.



(a) Original image of LED turned on (b) Result of subtracting two images of the LED on and off

Figure 3.8: Before and after subtracting two images of the LED turned on and off. Figure 3.8a is dark due to the ND-filter being attached. More on this filter in Chapter 6

3.3.2. Reduce noise by camera

Although subtracting the two images removes a large amount of background “noise”, such as objects of similar color, it does not remove noise introduced by the sensor of the camera. This means that, although appearing black, a large part of Figure 3.8 is not black. To reduce the possibility of detecting such noise, two techniques have been applied.

The first step involves a technique called eroding the image. In this technique, a structuring element, a shape such as a cross or a square, is used to ‘reduce’ the source image[20]. In the simplest case, the erosion is carried out by moving the center of the structuring element along the image. When the structuring element is contained within the source image, meaning that logical AND of the structuring element with its underlying pixels is true, the value of the pixel remains. If the logical AND is false, the value is removed[20]. This process smooths the edges and removes unwanted noise.

The second step involves creating a mask of pixels that should be ignored. Each pixel in the image is scanned. When the RGB value of the pixel is below a certain threshold RGB value, the pixel will be indicated in the mask to be ignored. This means that later algorithms will ignore this pixel, and will therefore not detect the pixel. This prevents small changes in for example lighting to be falsely detected by the color detection.

3.3.3. Blob selection

When the source image has been created, “noise” has been removed, and blobs have been detected, it is still possible that more than one blob has been detected. As visible in Figure 3.8, not only the LED is visible in the picture, but its reflection as well. It is therefore possible for the color detection algorithm to detect both the LED and its reflection. To make a selection of the correct blob, the circularity of each blob will be calculated, see Equation 3.8 (the derivation can be found in Appendix C.1), which will approach 1 the more it matches a circle. Since the LED is close to a circle when viewed from the top, and still closer to a circle than a large reflection when viewed from the side, each cluster’s circularity can be calculated and used to sort the clusters on likelihood of being the LED.

$$C = \frac{4\pi \text{Area}}{\text{Circumference}^2} \tag{3.8}$$

As visible in Equation 3.8, one of the parameters is the circumference. This means, that the edge of each cluster has to be found to get this value. The blob detector will, after finishing the blob detection, find the most outer pixels of each blob and store them. The amount of pixels in each edge can then be used as circumference, while the total amount of pixels in the cluster can be used as area.

After sorting by circularity, the pixel locations of the blob with highest circularity will be returned, as only one blob should equal the LED.

3.4. Results and discussion

In this section, the algorithms described above will be tested and their results will be discussed. This includes testing the color detection, Section 3.2.1 up and until Section 3.2.4, and the LED detection algorithm discussed in Section 3.3.

First the color detection will be tested in Section 3.4.1. The color detection algorithm will be tested in two stages, first the algorithm is pushed to its maximum, by disabling the automatic threshold detection

discussed in Section 3.2.2, and determining the thresholds manually. After this, the same images will be pulled through the algorithm, but this time the automatic threshold detection will be enabled. These color detection tests will be performed on pictures similar to Figure 3.5a, this calibration grid will be used later in Chapter 4 as well.

Finally, the LED detection will be tested in Section 3.4.2. Multiple pictures in different scenarios will be fed through the algorithm, and the algorithm will attempt to recognize the LED. The detected pixel locations will then be compared to the real pixel location of the LED.

3.4.1. Color detection

As mentioned in the introduction of this section, the color detection algorithm will be tested in two stages. A stage where the threshold will be selected manually to obtain the most optimal results, and a stage where the threshold will be determined automatically.

To test the performance of the color detection, the algorithm will have to detect multiple dots placed in a grid. The grid used for this test is visible in Appendix H.1. This grid will be placed underneath the camera, taking up the full field of view of the camera. In this way, the only parts that can be seen by the camera are: the dots, the white background, and the noise introduced by the background.

Another reason for placing the grid in the full field of view, is that it allows inspecting whether certain areas of the cameras field of view have more issues detecting the dots.

Manual threshold selection

The main color detection algorithm, meaning without automatic threshold detection enabled, will be tested under the following three conditions: a “normal” environment, the paper will be illuminated by environmental lights such as TL-lights, and a “light” environment, where the paper will be illuminated by a light placed such that the paper will be evenly illuminated.

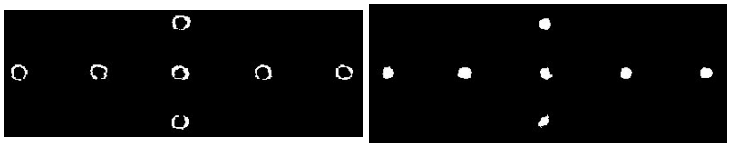
The results of these tests are visible in Table 3.1. The table shows the mean missed dots for red and blue dots, under the “light”, “normal”, and both environments (light and dark combined). The table does not contain the green results, since these are very similar to the blue results, and therefore do not add extra information.

Table 3.1: Mean values for amount of dots not detected

Color	Type	Mean misses	Mean misses [%]	Mean number of points
Blue	Normal environment	1.333	0.250	533.0
	Lighted surface (even lighting)	1.400	0.261	534.4
	Both environments	1.351	0.253	533.4
Red	Normal environment	0.370	0.864	43.33
	Lighted surface (even lighting)	0.200	0.467	43.40
	Both environments	0.324	0.756	43.35

The first thing that can be seen is that the algorithm is capable of detecting almost every point on the grid. With an average of 0.25% misses for blue, and an average of 0.864% for red, it can be safely said that the algorithm is accurate enough to be used in the final product. What should be noted, is that all the points that were not properly detected by the algorithm, were all located in either the far bottom left, or the far top left corner. One could therefore state that the issue of not detecting the dots could be caused by the camera itself instead of the algorithm.

Although the results are promising, one difference is interesting, namely the difference in the effect of illumination on the results. For blue detections, the influence of evenly lighting the surface is minimal. For red colors however, this difference is more noticeable. Although the table might look as if the difference is minimal as well, comparing two binary maps will make the difference more obvious (see Figure 3.9). The darker environment produces hollow circles, where the lighter environment allows the algorithm to detect the full dots.



(a) Binary image of red dot detection with uneven lighting

(b) Binary image of red dot detection with even lighting

Figure 3.9: Difference in binary image with even and uneven lighting

Similar as the issue with the corners in the left side of the images, this issue might be related to the camera. Objects of the red color will become under-saturated more easily, especially under lower light conditions. Next to this, colors of objects located closer to the edges of the image will start to change as well, with one of the key changes being the saturation that changes (explaining why dots are missed more easily in the corners of the image).

Although this can be fixed by increasing the range of HSV values, this also introduces the detection of noise. Therefore, detecting these dots will remain a difficulty.

Automatic threshold selection

Since the performance of the color detection algorithm itself has been verified, the performance of the color detection algorithm with automatic threshold detection enabled can be tested. The performance of the full algorithm can then be compared to results obtained in Section 3.4.1.

The data in Figure 3.10 show the following results, a dataset has been fed through the color detection algorithm, for each of these images the optimal threshold has been selected and the missed dots for this threshold have been noted. After this, the automatic threshold detection has been enabled, and the missed dots can then be compared to the previous obtained missed dots.

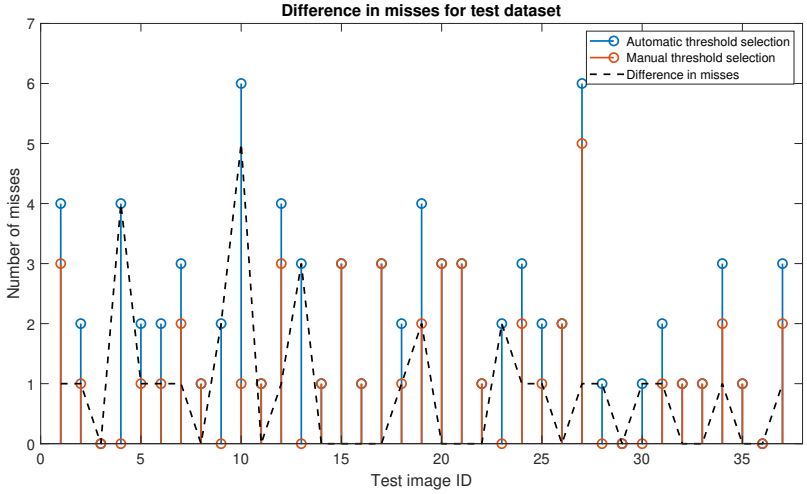


Figure 3.10: Difference in missed dots between manual and automatic threshold detection

When comparing the performance of the color detection algorithm with and without automatic threshold detection, the differences are small. Approximately 40% of this dataset were detected the same, while the other detections mostly differ only 1 missed dot. Similar to the results in Section 3.4.1, the newly introduced misses by the different threshold values were all located close to the already missed dots, in the left corners.

Although there are differences between the two measurements, there is still an option for improvement. The fit created in Section 3.2.2 could be changed to, for example, include more points with higher threshold values. To be safe and reduce the detection of noise, these points had been left out of the fit.

On a final note, the difference of (most of the time) one extra miss, is almost negligible when keeping in mind that on average 534 dots were available for detection.

3.4.2. LED detection

As mentioned in the introduction of this section, the LED detection will be tested by comparing real pixel locations to detected pixel locations of LEDs. To fully test the performance of this system, a

feature which will be discussed further in Chapter 6 will be used during these tests. This feature is a so called ND-filter which, in short, reduces the intensity of incoming light. Because of this, lens flare (see Figure 3.1), incoming light from background sources, and other “noise” introduced by objects in the background can be reduced. More on this will be discussed in Chapter 6.

ND-filter installed The first tests performed are shown in Table 3.2. These tests are done with the ND-filter installed on the camera, in 4 different scenarios. Dataset 1 was performed by moving the LED on the vertical axis with respect to the camera. Dataset 2 and 3 were performed on the horizontal axis, where dataset 2 was performed with a darker, less ”noisy” background, and dataset 3 with a lighter more noisy background. Finally, dataset 4 was created by pointing the LED straight into the camera, which would normally induce lens flare. The full datasets can be found in Appendix E.

Table 3.2: Mean deviation in pixels with ND-filter installed (ND-filter will be discussed in Chapter 6)

Dataset name	Mean x deviation	Mean y deviation	Mean absolute deviation
Dataset 1	1.250	0.750	1.644
Dataset 2	0.429	1.286	1.438
Dataset 3	0.833	1.000	1.443
Dataset 4	0.857	0.429	1.067
Total	0.792	0.875	1.365

When comparing the results of all the datasets, the differences between the mean values are rather similar. Although the mean x deviation of dataset is higher than the other deviations, its y deviation is on the lower side. Therefore, it is difficult to state that there is a noticeable difference between vertical and horizontal movement. The differences between lighter and darker environments is also negligible. Although the x deviation is smaller for dataset 2, its y deviation is larger. It is therefore difficult to conclude that there exists a difference between lighter and darker environments. Because of the small differences between scenarios, it can be said that the system is able to perform well under various different circumstances, and with mean values of approximately 1 pixel deviation the results are promising.

No ND-filter installed The second set of tests performed are shown in Table 3.3. These tests have been performed without the ND-filter installed, to research the difference in performance induced by over-saturation and lens-flare when looking straight at an LED.

The datasets created for this test were performed in 3 different scenarios. Dataset 1 was created by moving the LED on the vertical axis with respect to the camera. Dataset 2 and 3 were performed on the horizontal axis, where dataset 2 contains windows in the background, introducing over-saturation induced by the sunlight.

The first noticeable difference, is the effect of the windows on the measured deviation. As visible in Table 3.3, the absolute deviation is more than 1 pixel higher. This difference could be explained by the over-saturation of the windows, causing the camera having issues detecting other light-sources. Since the LED itself will mostly be seen as an over-saturated dot surrounded by a circle of its color (see Figure 3.1), most of the circle will be difficult to detect.

Table 3.3: Mean deviation in pixels without ND-filter installed (ND-filter will be discussed in Chapter 6)

Dataset name	Mean x deviation	Mean y deviation	Mean absolute deviation
Dataset 1	0.714	3.000	3.169
Dataset 2	2.667	4.833	5.656
Dataset 3	2.200	3.600	4.295
Total	1.778	3.778	4.311

Apart from this difference, the other performances are rather similar. Comparing this data to the data from Table 3.2 however shows larger differences. The lens-flares and over-saturation induced by the LED cause the algorithm to have more issues finding the real center, as the algorithm now relies on the lens-flare surrounding the LED instead of the LED itself. Therefore, although both scenarios still allow detecting the LED, the filter certainly improves the results.

4

Lens calibration

The antennas (or the markers on the antennas) can now be recognized within the image, however that is not enough, since the location of the antenna within the image is just a pixel location and (at this point) this pixel location does not tell anything about the location within the dome, or with respect to the camera. Therefore a Lens Calibration will be applied that will map each pixel to a specific spherical coordinate with respect to the camera.

This chapter describes to process of mapping the pixel locations to spherical coordinates θ and ϕ with respect to the camera.

The goal is to be able to accurately recognize each of the markers present in the calibration grid, of which the locations are accurately known, after which the pixels in which these markers are located can be linked to the spherical coordinates corresponding to these accurately known locations.

The process of mapping the pixels to spherical location comprises of four stages: Recognition of the point cloud, distortion estimation and correction, mapping the recognized 'undistorted' points back to the original grid, and finally mapping the pixels to spherical coordinates.

4.1. Recognition of point cloud

The first step of creating the mapping will be creating a controlled environment. This controlled environment will be created by placing several well-recognizable markers at accurately known locations with respect to the camera.

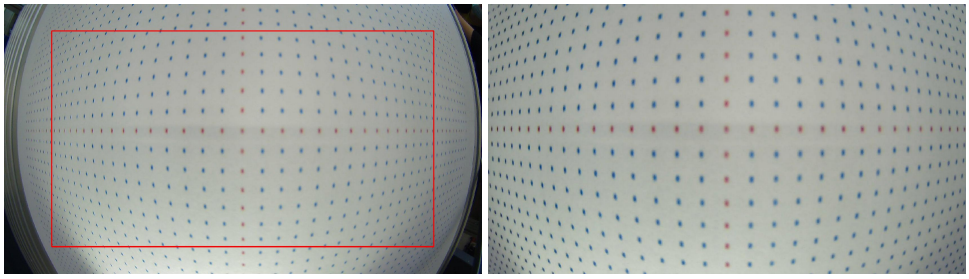
The markers used in this calibration are colored dots on a piece of paper. The distance between all of these dots is accurately known and these dots are distributed uniformly over the vertices of a square grid. The color of these dots is blue, except for the dots located at the line dividing the grid horizontally into equal parts and dividing the grid vertically into equal parts. A smaller version of the calibration grid is shown in Appendix H.1.

Using the same camera as the camera used for antenna recognition and landmark detection, a picture will be taken of this grid. This camera is placed on a CNC-machine using a custom 3D-printed camera mount to be able to accurately move to camera above to grid. Now the camera is moved above the calibration such that the center dot is in the center of the picture, and the picture lies straight (so the horizontal and vertical lines are at approximately 0 and 90 degrees) beneath the camera. In Figure 4.1a such an image is shown, in this picture also the region of interest is drawn.

In Figure 4.1a a somewhat warped image can be seen. The distribution is not uniform anymore, since the density of dots near the corners of the image is much higher than near the center of the image. This is caused by the distortion of the lens and will be further discussed in Section 4.2.2. Besides the distortion, also chromatic aberration can be noticed near the corners, meaning that the colors will become less saturated, or in other words more gray-like.

These dots will be recognized using the algorithm as discussed in Section 3.1. Therefore the more saturated the color of a dot is the better it can be detected. As mentioned earlier the saturation will decrease the further away the dot is from the center of the lens, until it will become so under-saturated that it will not be recognized as a color. Therefore, the decision has been made on forehand to cut out these 'far' parts of the image, by only using 80% of the width and 80% of the height of the image. In

Figure 4.1a the region of interest (ROI) is shown in the original image and Figure 4.1b shows only this ROI.



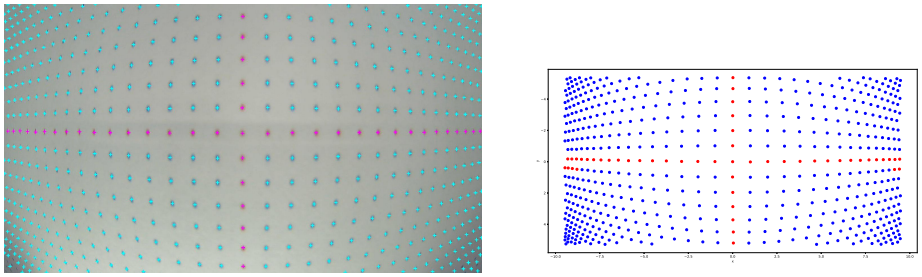
(a) Picture of the calibration grid showing the ROI in red

(b) ROI of the picture taken from the calibration grid

Figure 4.1: Figures showing the ROI

All the dots shown in Figure 4.1b can be accurately seen by the naked eye, while the dots outside the ROI are harder to spot due to the color aberration and the low resolution of the camera; this can be seen outside the ROI of Figure 4.1a.

Within this ROI, the dots will be recognized. The result of this recognition is shown in Figure 4.2a on the original image, and in figure 4.2b on a blank space.



(a) Dot recognition of the region of interest. Red dots are marked with a magenta marker and blue dots are recognized with a cyan marker

(b) Recognized dots shown in an empty space

Figure 4.2: Dot recognition

4.2. Distortion estimation and correction

Now that all the dots are recognized from the picture, the structure at which these points were located on the original grid must be recognized. The structure is chosen to be fairly simple, since now all the points are located on the vertices of a square grid. However, as can be seen in Figure 4.2a the dots are not in this structure anymore due to distortion.

For the dots located near the center it is not a problem to recognize the square grid, since as you can see, you can still easily recognize which points are located in all four cardinal directions. For the dots near the edges of the frame, it becomes fairly hard to determine this. Therefore, a distortion estimation and correction will be applied to make the grid more square-like, such that the recognition of the square grid becomes more accurate.

This section will first discuss the theory of distortion shortly, and how camera systems introduce distortion to images. After which a distortion algorithm will be discussed, and finally the method used to correct for the distortion will be described.

4.2.1. Theory

As described in the introduction of this section, the lens introduces distortion. There are two types of distortion that can occur due to lenses: Barrel distortion and pincushion distortion. The first step is understanding the differences between the two.

To explain the difference between barrel and pincushion distortion, we first take a look at the mathematical background in how distortion works. The distortion introduced by lenses can be described using Equation 4.1[21].

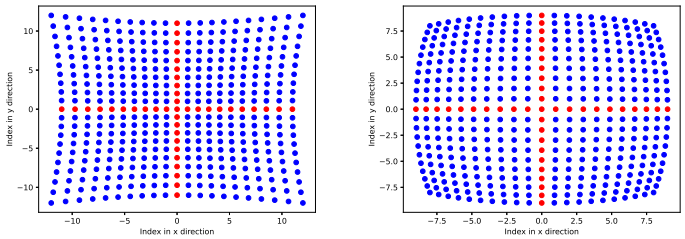
$$\begin{bmatrix} x_{out} - x_c \\ y_{out} - y_c \end{bmatrix} = \left(1 + K_1 r_d^2 + K_2 r_d^4 \dots K_N r_d^{2N}\right) \begin{bmatrix} x_{in} - x_c \\ y_{in} - y_c \end{bmatrix} \tag{4.1}$$

Where (x_{in}, y_{in}), (x_{out}, y_{out}) and (x_c, y_c) correspond to the distorted point, undistorted point and the distortion center respectively. K_i represents the i^{th} order distortion coefficient, and r_d is the radius from the center of distortion to $(x_i n, y_i n)$ as shown in Equation 4.2.

$$\sqrt{(x_{in} - x_c)^2 + (y_{in} - y_c)^2} \tag{4.2}$$

As becomes clear from Equation 4.1, the further a point lies from the center of distortion, the more effect distortion will have on this point. When K_i is positive, the term corresponding to this coefficient will move the point further away from the center of distortion, and when K_i is negative it will move the point closer to the center of distortion.

When the contribution of all distortion coefficients moves the points further away from the center of distortion we speak of pincushion distortion, and when this contribution moves the points closer to the center we speak of barrel distortion. In Figure 4.3a the calibration grid as shown in Figure H.1 is distorted using only a first order distortion with $K_1 = 0.001$, resulting in a pincushion distortion. In Figure 4.3b, the same grid is used and distorted using only a first order distortion with $K_1 = -0.001$, resulting in a barrel distortion. The center of distortion of both of these examples is the center dot of the red dots.



(a) Pincushion distorted calibration gird. The distortion coefficient $K_1 = 0.001$, the other distortion coefficients equal 0

(b) Barrel distorted calibration gird. The distortion coefficient $K_1 = -0.001$, the other distortion coefficients equal 0

Figure 4.3: Distortion examples using the calibration grid

The distortion as described above only discusses radial distortion. Besides this, a tangential distortion could be present as well. Tangential distortion would occur when the sensor and lens do not align. For this research the assumption is made that the lens and camera do align, and therefore tangential distortion will not be discussed, or accounted for.

4.2.2. Distortion estimation

Figure 4.2 shows the distorted image of the calibration grid. To be able to accurately recover the structure of the calibration grid (and therefore the exact location of all the dots), the distortion must be corrected for.

Since the distortion coefficient of the lens is not yet known, this must be estimated. For this research only the first order distortion coefficient will be approximated, since the higher order terms will only result in marginal differences[22]. Further more, perfectly modelling the characteristics of the lens is not within the scope of this paper; this modelling will only be used to better fit a square grid through the points.

In this paper 4 methods will be discussed; each of these methods will use the same principles but will be applied slightly differently. First this main principle will be discussed after which the differences between the 4 methods will be discussed.

The first step is normalizing the grids such that the center of distortion is located at (0, 0) and that the distance between the center and it four nearest neighbors equals 1. The four methods will try to get the distance between two neighboring dots to become 1. This will be achieved by distorting the image in the opposite way as has been done by the camera until the distance between these two neighboring dots approaches 1.

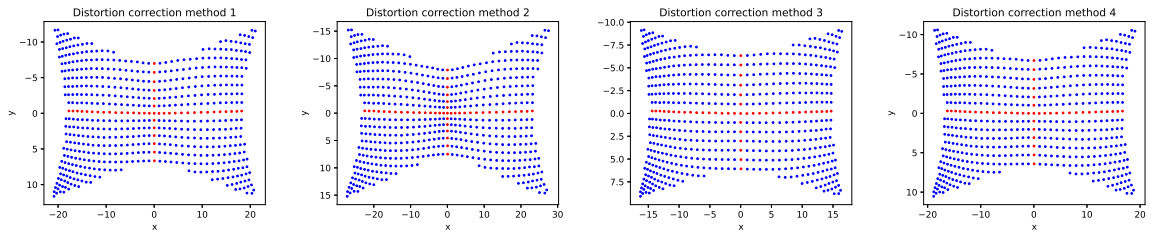
Method:

1. is based on only the furthest **red** dot from the center (which would be the most distorted) and its direct **red** neighbor. The estimation algorithm will try to get the distance between these points to get as close to 1.
2. is based on only the furthest dot and its direct neighbor. These dots can be either red or blue (in all probability these dots will be blue). The estimation algorithm will try to get the distance between these points to get as close to 1.
3. will use all the **red** dots and their direct **red** neighbors. The estimation algorithm will calculate the RMS error on how far off each of the distances are from 1, and will try to minimize this error.
4. will use all (both blue and red) dots and their direct neighbors. The estimation algorithm will calculate the RMS error based on how far off each of the distances are from 1, and will try to minimize this error.

The pseudocodes of the 4 methods can be found in Appendix D.1.

4.2.3. Distortion correction

Once the distortion coefficient has been estimated using one of the algorithms as stated in Section 4.2.2, this distortion coefficient will be used to 'undistort' the point cloud recognized in Figure 4.2b. This undistortion will again be done by distortion the pointcloud in the different direction as has been done by the camera. In Figure 4.4 the result of such distortion corrections on Figure4.2b are shown. Figures 4.4a to 4.4d show the results of the distortion corrections using distortion estimation methods 1 to 4 respectively.



(a) Distortion correction using $K_1 = 0.01047$. K_1 is found using distortion estimation method 1

(b) Distortion correction using $K_1 = 0.01631$. K_1 is found using distortion estimation method 2

(c) Distortion correction using $K_1 = 0.00628$. K_1 is found using distortion estimation method 3

(d) Distortion correction using $K_1 = 0.00865$. K_1 is found using distortion estimation method 4

Figure 4.4: Distortion correction on the point cloud shown in Figure 4.2b. For enlarged version see Appendix F.1

As shown in Figures 4.4a to 4.4d, distortion estimation methods 1 and 2 will result in a much more aggressive distortion correction in comparison to methods 3 and 4, where method 2 is the most aggressive. This is due to the fact that methods 1 and 2 only use a single heavily distorted point, while method 3 and 4 use all the points.

While methods 1 and 4 seem the best, as these (on average) look the best like a square grid, the decision has been made to use method 2. Method 2 has the most aggressive correction, which results in the most aggressive correction near the edges of the frame. The edges are the locations where it is hardest to fit a grid through the points since these remain the most distorted, even after distortion correction. While, due to the aggressive distortion correction, the center of Figure 4.4b seems to be distorted, this is still less distorted than the edges of the other figures.

4.3. Map to calibration grid

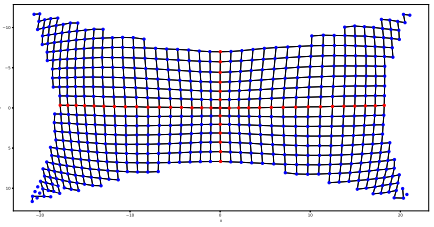
As has been said in Section 4.1, the structure of the calibration is a square grid. This means that every dot in the picture (excluding the boundaries) has 4 direct neighbors: one in each of the cardinal directions. This knowledge makes recovering this structure from the point cloud a little simpler.

As you might imagine recognizing these neighbors becomes a lot harder once they lie not simply in the cardinal directions, as occurs in the distorted image shown in Figure 4.2. The distortion correction

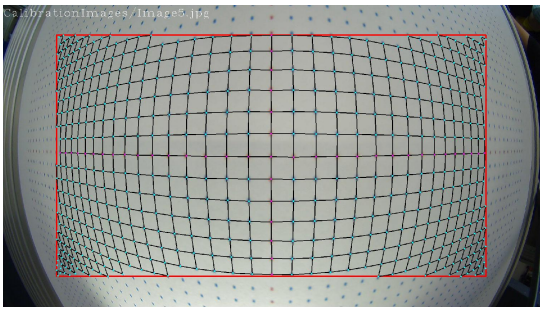
algorithms discussed in Section 4.2.2 make the dots be more like the original grid, and therefore it becomes much simpler to recognize which dots are the direct neighbors of other dots.

To eventually find the structure of the calibration grid, first the 4 direct neighbors of each dot must be found. This is achieved by starting at the center (red) dot. From there we'll find a dot closest to this center dot, for each of the 4 cardinal directions. Once such a near neighbor is found, the reference will be moved to that dot, and the 4 closest in each cardinal direction of that dot will be found. This will happen recursively until all the dots are used, or do not fit the criteria to become a neighbor of another dot.

In Figure 4.5 a connection is drawn between two dots if they are each others direct neighbors. The dots in Figure 4.5a are the same as the dots shown in Figure 4.4a, thus the distortion has been corrected using the distortion coefficient found with distortion estimation method 2. In Figure 4.5b the connections from Figure 4.5a are drawn on the original image as shown in Figure 4.1a.



(a) Undistorted grid where the nearest neighbors are connected recursively



(b) Dots connected on the original image

Figure 4.5: Distortion examples using the calibration grid

Using the knowledge of which dots are neighbors of which other dots, the calibration grid (or at least part of it) can be found easily. Again start at the center and recursively go through the nearest neighbors of each of the dots. This time however, a new coordinate is given to each of the points. This coordinate belongs to the UnitGrid.

The UnitGrid is a grid where the distance between each of the neighboring dots is 1, much like the grid shown in Figure H.1. The UnitGrid describes the integer coordinates of the dots with respect to the center.

The result of the mapping to the UnitGrid of Figure 4.5 is shown in Figure 4.6.

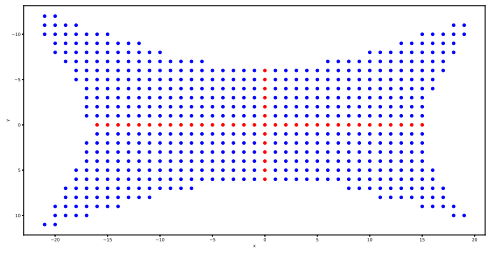


Figure 4.6: UnitGrid of Figure 4.5

Each of the dots shown in Figure 4.6 corresponds to one of the dots of the calibration grid. Now the location of a dot on the picture is known in pixels (Figure 4.1a), as well as its location on the calibration grid.

4.4. Map pixels to angles

Now the location of each dot in the picture and its location on the calibration grid is known. The only step is to map these to spherical coordinates θ and ϕ with respect to the camera. The distance between each of the dots on the calibration grid is known, as well as the approximate height of the camera. Using

these, the Cartesian coordinates can be easily calculated where (0,0, CameraHeight) is the center dot. Now the conversion from Cartesian to spherical coordinates, as shown in Equation 4.3, will be applied to get the spherical coordinates of each of the dots with respect to the camera.

$$\theta = \arctan \frac{\sqrt{x^2 + y^2}}{z} \tag{4.3a}$$

$$\phi = \arctan \frac{y}{x} \tag{4.3b}$$

4.4.1. Plane approximation of N points

Now the location of each of the dots are known, which correspond to a select set of pixels. The antennas could be present in any pixel within a picture. Therefore, when an antenna is found an approximation will be done according to the dots from the calibration to get the location of the antenna.

Whenever the location of an antenna is found, the N dots of the calibration grid closest to this newly found point will be found. After which a XYZ-plane will be fitted through these points for both θ and ϕ. The method of fitting the plane through the points is given in Appendix C.2. This will give an equation in which the pixel location can be filled in to get the values for both θ and ϕ.

The mapping of all pixels to their corresponding (ϕ, θ) is stored in an so called AngleMap.

4.5. Results and discussion

This section will describe the performance of the methods described in Sections 4.2 to 4.4. This performance will mainly be determined on how well the algorithms operate on test images. These test images will be generated by taking the grid shown in Appendix H.1 and distort it (using only a first order distortion) with Equation 4.1, with both positive and negative values for K_1.

First, the accuracy of the distortion estimation (Section 4.2) will be determined. Then the accuracy of the square grid fitting as discussed in Section 4.3 will be determined, for both the distorted and its corrected situation to show the purpose of the distortion correction and how well the dots can be mapped back to the calibration grid.

4.5.1. Accuracy of distortion estimation

This section discusses the performance of the distortion estimation. The error of an estimate is determined according to Equation 4.4.

$$\text{Error} = \frac{K_1^E - K_1}{K_1} \tag{4.4}$$

In Figure 4.7 the errors for the different methods of distortion estimation are shown for different values of K_1. In this case only the negative values of K_1 are shown since the distortion estimation algorithm is optimized for this type of distortion. This optimization is done since almost all lenses will introduce distortion in this way. Also the approximate area in which the distortion coefficient of the camera lies is shown in Figure 4.7.

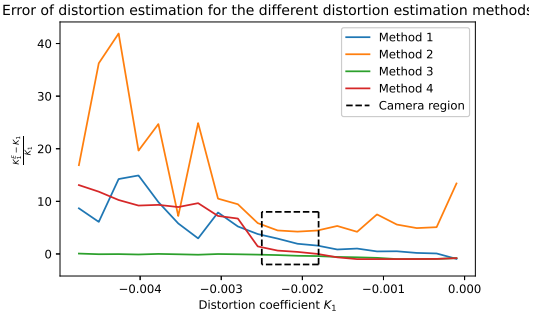


Figure 4.7: Errors of the 4 methods of distortion estimation for different values distortion coefficient K_1. The images used to determine these errors are computer generated

As shown in Figure 4.7, Method 3 seems to have the lowest overall error. These red points will only distort in 1 direction (either x or y), and will therefore remain their relation to the other red points. There will be no points which could become closer than their direct neighbors, since there could occur no diagonal movements within this set of points.

For higher values of K_1, the grid is more heavily distorted. This will lead to two diagonal points to be pushed further towards each other. Eventually, the distortion is so high that those two points become closest to each other; Method 2, which will probably look at these two point will then set the distance between those points to 1, instead of these points and their proper neighbors. This will therefore introduce more deviation.

4.5.2. Accuracy of fitting the square grid

Without distortion correction First the accuracy for the square grid fitting is determined without distortion correction. Now, when distortion is applied too much, it can be determined whether this new situation will probably still have a proper grid, since this new grid may now be inside the domain for which a proper grid can be found. In Figure 4.8 the amounts of disconnected dots, and the amount of wrong connections are plotted against different distortions. The less of both the better. Wrong connections do not instantly mean that the UnitGrid found through the points to which these wrong connections belong is also wrong, it could happen that the UnitGrid is still found properly through other connections.

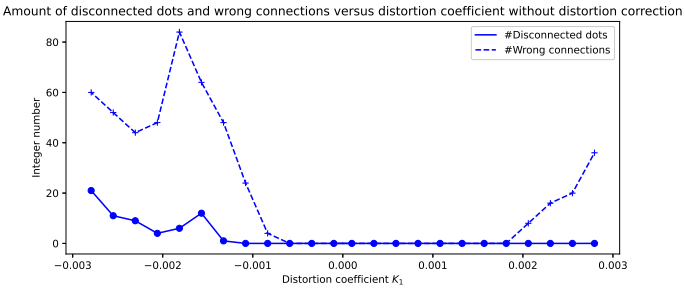


Figure 4.8: Amount of disconnected dots and wrong connections versus distortion coefficient without distortion correction

With distortion correction As shown in Figure 4.8, as soon as a little negative distortion is introduced a lot of wrong connections are also introduced; the camera used for this thesis has a negative distortion coefficient below -0.001. Therefore, a distortion correction will be applied to increase these accuracies. In Figure 4.9 the amount of disconnected dots, and the amount wrong connections are shown for same images as used for Figure 4.8, but now distortion correction is applied. For this research only the negative distortion coefficients are used, since these are the only relevant ones in this research.

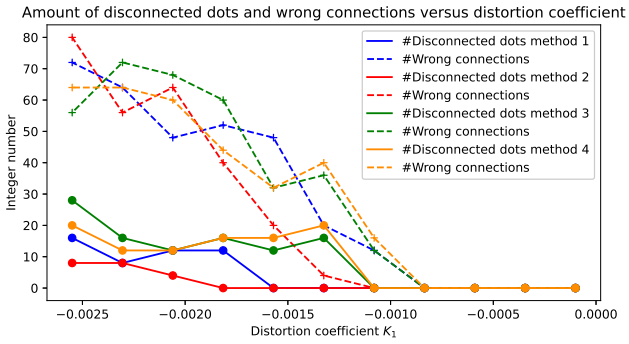


Figure 4.9: Amount of disconnected dots and wrong connections versus distortion coefficient

Figure 4.9 shows that none of the estimation methods are accurate across to whole domain, but all are already better than using no distortion correction as shown in Figure 4.8. Method 2 seems to work the best for finding the grid, while this was the method that was the least accurate in finding the true distortion coefficient. This is most likely due to the fact that finding the grid is hardest to find near the

corners of each frame where the frame is most distorted, and since method 2 is the most 'aggressive' method, it will undistort these corners the best. From Figure 4.8, we can deduct that the grid could still be found for distortion coefficient K_1 higher than 0, so if the correction for the distortion is too aggressive, the overall distortion coefficient will go to this region above 0 where the grid could be found accurate, according to Figure 4.8.

4.5.3. Accuracy of the angle mapping

Now the performance of the distortion estimation, and the performance of the fitting to a square grid are determined, the performance of the system as a whole can be determined. Here the camera will be placed in front of an LED light under a predetermined angle, after which this LED will be recognized, and mapped according to the map created by the calibration. The angles used here are multiples of 10 degrees, and this measurement is repeated for each of the cardinal directions. The calibration is done under 2 circumstances. In situation 1 the calibration is performed using the center of the image as the true center, so the red dot at the center of the calibration grid is placed there. In situation 2, an estimate is made to determine the center of distortion, after which the center red dot is placed at that position. In Table 4.1 the results of these measurements are shown.

Table 4.1: Average deviation of spherical coordinates from real coordinates

Cardinal direction	North	South	East	West	Average
Avg Deviation in degrees using calibration 1	0.4	0.3	0.7	0.5	0.5
Avg Deviation in degrees using calibration 2	1.8	1.5	1.4	0.9	1.4

From Table 4.1 it is clearly shown that the calibration method using the center as the image for the center dot is the better method for determining the spherical coordinates θ and ϕ with respect to the camera.

Figure 4.10 shows an image acquired during the testing of this section. In this image, a yellow dot at the center is located, this is the center of the image and corresponds to the coordinates $(\phi, \theta) = (0, 0)$. Further more, a magenta dot is drawn, this is the location of the recognition of the LED. Next to this magenta dot, the spherical coordinate (θ, ϕ) are drawn. These are the spherical coordinates with respect to the camera. In this setup the camera was rotated 30 degrees to the left, so the LED should be at 30 degrees to the right.

Figure 4.10: Example of test showing the determination of the spherical coordinates with respect to the camera. The LED in this situation is located 30 degrees to the right of the camera.

5

Landmark localization

The antennas have now been recognized within the picture, and the spherical coordinates θ and ϕ are known with respect to the camera. However the final goal is to get the location of the antennas with respect to the coordinate system of the ADome. To achieve this, landmarks are used. Landmarks are fixed points within the ADome of which the locations are accurately known, and will be used as reference points to transform the coordinates from spherical with respect to the camera to spherical with respect to the coordinate system of the ADome. The landmarks used for this report are ArUco fiducial codes.

This chapter will first describe the fiducial code ArUco and why this is used, after which the methods which describe the mapping of the location of an antenna with respect to the camera to the coordinate system of the ADome will be described. In the final section of this chapter the testing of both the ArUco recognition, and mapping will be discussed.

5.1. ArUco

There are many types of fiducial codes in the field of computer vision. One of the most well-known fiducial codes used right now is the QR-code. These codes can store many different types of data, from text files and urls, to small images and pdfs.

The problem with these complex fiducial codes, like QR codes, is that these are complex, fairly high resolution codes. These codes are equipped with error checking, and error correcting codes, making it harder to detect than simpler codes. Due to the relatively low resolution of the camera, an alternative (simpler) tag will be used, which in itself has a lower resolution and will therefore be easier to detect. In Figure 5.1 two fiducials are shown which both only carry the data of a simple integer value of 1; Figure 5.1a is a QR code and Figure 5.1b is an ArUco code.

(a) Example of QR code carrying the data '1'

(b) Example of 4x4 ArUco code carrying the data '1'

Figure 5.1: Example of complex versus simple fiducial code

An example of a simpler fiducial code is the ArUco code. These codes are especially designed for augmented reality (AR), as the name specified (Augmented Reality University of Cordoba). Other examples of fiducials used in AR are AprilTag[5] and ARTag[6]. All these examples only store a simple integer ID within the tag, instead of the complicated data stored in the general use case. All fiducial

codes in the field of AR have an orientation. This means that the upper left corner will always be the upper left corner, no matter how this fiducial is rotated in the picture.

For this research there is chosen for the ArUco tag, over the AprilTag and ArTag. The ArUco and AprilTag are the simplest tag, whereas both of these can be set to a lowest resolution of 4x4. The ARTag is already a more complex code, which already uses a uniqueness property between the markers which reduces inter-marker confusion[23]. Due to this the uniqueness, the resolution is limited to a minimum of 6x6. The difference between the AprilTag and ArUco however is marginal. The main reason ArUco is used over AprilTag is due to the simpler setup process for ArUco. The detection and decoding of ArUco codes is implemented in the contributors edition of OpenCV. This recognition and decoding is based on the older version of ArUco, since the newer version is licensed under GPL instead of BSD. The other main reason for the use of ArUco codes, is the lower detection fail rate in comparison to AprilTag[24].

5.2. Transformation of relative to camera to relative to dome

As mentioned in the introduction of this chapter, the locations of the antennas have been detected by the color detection, and transformed into spherical coordinates. The only issue with these coordinates is, that they do not represent the real coordinates inside the ADome. Since no knowledge is available on the orientation and location of the camera itself, the spherical coordinates are related to the camera as if the camera is located in $(x, y, z) = (0, 0, 0)$, and looking towards $\theta = 0$ and $\phi = 0$. Therefore, these spherical coordinates should be transformed such that they fit into the coordinate system of the ADome itself.

As mentioned in the introduction of this chapter, this task can be performed by using landmarks with predetermined locations within the ADome. This way, the spherical coordinates of these landmarks can be detected, and used to find a transformation for the detected antenna coordinates.

Before the transformation can be determined, the position of the camera with respect to the center of the ADome has to be estimated. To achieve this, two steps will have to be taken. The first step is determining the distance of the camera to certain predetermined landmarks. These distances can be then be used in the second step to find the location of the camera.

When the location is found, the spherical coordinates of the antenna can be transformed to be centered around the center of the ADome. This is done by detecting landmarks with respect to the antennas, and transforming these coordinates such that they fit the coordinates of the coordinate system of the ADome.

5.2.1. Distances to landmarks

The Cartesian locations of the landmarks are known, and the Cartesian coordinates of the camera have to be found. Although these coordinates are 3 dimensional, for an easier analysis, the relationship between landmarks and camera will now be considered as a two dimensional triangle, visible in Figure 5.2. In this figure, the positions of two markers are indicated by M_i and M_j, and the position of camera is indicated by C.

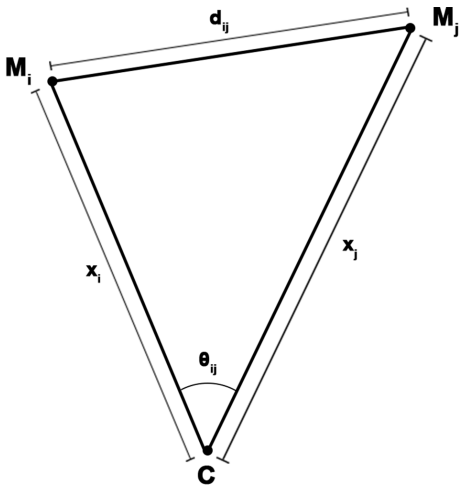


Figure 5.2: Relationship between two landmarks and the camera

For this situation, two sets of variables are known. Since each landmark's location is known, the distance between two landmarks (d_{ij}) is know as well. Next to this, the spherical angles from the camera to both landmarks is known. These angles can be used later to determine the angle between the two markers θ. Since we know these variables, the simple equation for determining the length of a line across an angle can be used, visible in Equation 5.1. Since the length of this line is already known, d_{ij}^2 will be constant and can be subtracted from both sides, leaving a formula of 2 variables (the distances from C to M_i and M_j, x_i and x_j respectively) that should be equal to zero [25].

$$d_{ij}^2 = x_i^2 + x_j^2 - 2x_i x_j \cos(\theta_{ij}) \tag{5.1}$$

To solve this equation, at least three different points have to be used. Therefore, when using 3 landmarks, a system of equations of three equations will remain (see Equation 5.2). In the following part of this subsection, the analysis of the system of equations will be discussed in general. For the full analysis, the reader should refer to Appendix C.3.

$$\begin{cases} f(x_1, x_2) = 0 \\ f(x_1, x_3) = 0 \\ f(x_2, x_3) = 0 \end{cases} \tag{5.2}$$

In order to solve this system of equations, the formulas should first be reduced to 1 single variable. To achieve this, the so called Sylvester resultant method is used [26]. In short, the method places the coefficients of two polynomials in the Sylvester matrix (see Equation C.15). If these polynomials share a common zero, the determinant of this matrix should equal to zero. Therefore, by applying this method on two of the three polynomials, a new 4th order polynomial ($h(x_1, x_2)$) is created, consisting of 2 variables instead of 3. Since this equation also has to be equal to zero, it can be added to the system of equations. This new system of equations is visible in Equation 5.3.

$$\begin{cases} f(x_1, x_2) = 0 \\ h(x_1, x_2) = 0 \end{cases} \tag{5.3}$$

After this, the same method can then be applied to both the newly created polynomial $h(x_1, x_2)$, and the remaining polynomial $f(x_1, x_2)$. This results in an 8th order polynomial $h(x_1)$, which should be equal to zero as well. This final equality can be solved, resulting in a value for the distance between landmark 1 and the camera, and can be used to determine the other 2 distances as well.

5.2.2. Location of the camera

When the distances to each landmark have been approximated, the location of the camera can be calculated. This calculation can be done in multiple ways, in this section two methods will be discussed.

The first method involves the use of triads to find a translation from one coordinate system to the other [27]. These triads will be created using the locations of the three landmarks and the real positions

of the landmarks. The translation and scaling of these coordinate systems is a simple task, once the rotation of the coordinate systems is known. To find this rotation, the triads will then be used to create a closed form solution in the form of a rotation matrix. To find an accurate rotation, the solution should be found using a specific least-squares method. This method uses quaternions to create a 4th order polynomial which can be solved to find the rotation [27]. The mathematics behind this method are of a very high level, and due to time constraints, this method is left to further research.

The second, and also selected method for this system, is by using Equation 5.4. In this equation, M_i is a landmark in the ADome, and $M_{i,\text{estimated}}$ is the location of this landmark estimated using the calculated distances between the camera and the landmarks. Subtracting the Cartesian coordinates of this estimation from the real Cartesian coordinates should equal zero when the camera is located in $[0, 0, 0]$. When the camera is translated, the vector will be non-zero. When the camera is not only translated but also rotated, two offset angles θ_o and ϕ_o will have to be added to the spherical coordinates of the landmark, in order to compensate for the rotation of the camera.

$$\text{spher2cart}(M_{i,\text{spherical}} + [0, \theta_o, \phi_o]) - M_{i,\text{estimated}} \tag{5.4}$$

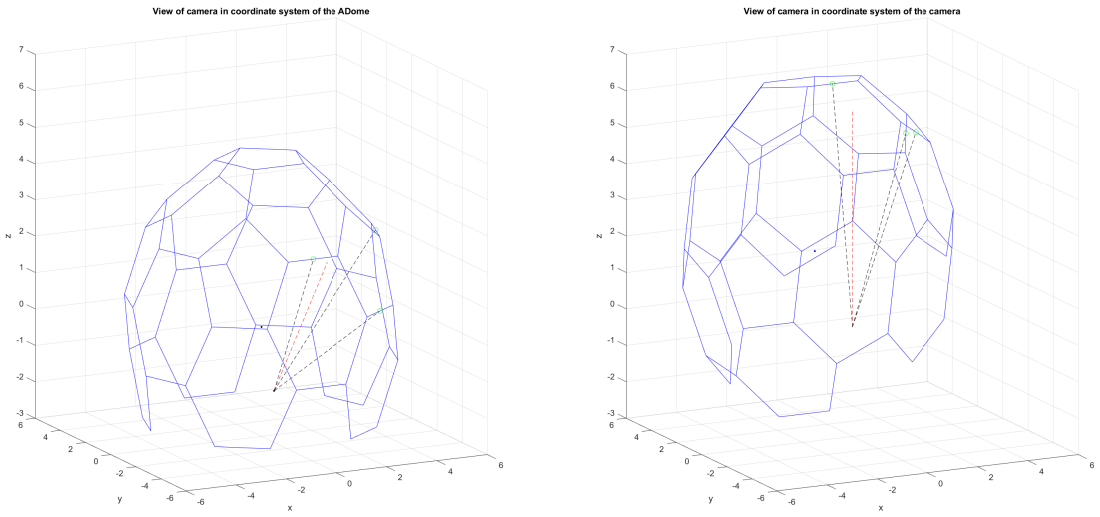
This equation will be used twice for different markers to find these offsets, by subtracting them and setting them to zero. The solution for the required angles will then be substituted in one of the equations leading to the location of the camera.

5.2.3. Location of the antenna

Now the location of the camera is known, the final tasks is to find the absolute location of the antenna in the coordinate system of the ADome. The camera can have any orientation, which means the coordinate system of the camera will most likely not coincide with the coordinate system of the ADome.

During the process of determining the location of the camera, the spherical coordinates of each landmark are found, since θ and ϕ are used to determine the R for each of the landmarks. Now these three landmarks can be placed in the coordinate system of the camera. The landmarks will still have the same distances between them as in the coordinate system of the ADome. The camera and the ADome both have a different coordinate system, the coordinate system of the camera is defined around the camera, with the origin being present at the camera, and the positive z direction is away from the center of the lens. The coordinate system of the ADome has its origin at the center of the ADome, with the positive z direction going straight up.

In Figure 5.3a a schematic view of the ADome is shown with the camera inside having a different orientation and an offset with respect to the center of the ADome. This is in the coordinate system of the ADome. The orientation of the camera is shown with a red dashed line, and the black dashed lines are the lines which connect the camera to the landmarks. Figure 5.3b shows the same situation as shown in Figure 5.3a, but now from the coordinate system of the camera.



(a) View of camera in the coordinate system of the ADome.

(b) View of camera in the coordinate system of the camera

Figure 5.3: View of camera in different coordinate systems. In these examples the camera is tilted -45 degrees around the x-axis

As can be seen by comparing Figure 5.3a and 5.3b, the coordinate system of the camera is a translated and rotated version of the coordinate system of the ADome. Therefore a method is designed to find this transformation, such that, once this transformation is found, the inverse transformation can be applied to the location of the detected antenna, such that the antenna can be mapped to the coordinate system of the ADome.

The location of the camera in the coordinate system of the ADome is found in Section 5.2.2.

To find the orientation difference between the coordinate system of the camera and the coordinate system of the ADome, a plane will be fitted through the landmarks in both coordinate systems using the plane fit method discussed in Appendix C.2. The orientation of both planes will be determined, after which the difference between the two will be taken.

Now the full transformation from the coordinate system of the camera to the coordinate system of the ADome can be determined. To find the absolute location of the antenna in the coordinate system of the ADome, the ADome will be approximated by a sphere. This sphere will then be drawn in the coordinate system of camera, after which a line will be drawn from the camera through the sphere, using the θ and ϕ found in Chapter 4. The crossing between the sphere and the line, will then be mapped to the coordinate system of the ADome. This final mapping places the antenna in the coordinate system of the ADome.

5.3. Results and discussion

5.3.1. Limitations of the ArUco detection and decoding algorithm

This section discusses the limitations and the testing regarding the use of ArUcos.

First there will be determined what size of ArUco can be used at which distance from the camera, to know which size of ArUco could be used within the antenna. After which, a test will be done which discusses how well the detection works at certain angles. This test determines where in an image, i.e. at what angles, the detection still operates.

Distance / size For the first test, a test sheet with ArUco codes of different sizes (0.5x0.5 to 2.25x2.25 inch in increments of 0.25 in both direction, see Appendix H.2 for the test sheet) is placed at a fixed location. Now, the camera will be positioned at fixed distances from this test sheet, where a picture will be taken of the test sheet from all these distances. A schematic of the test setup is shown in Appendix G.1. In all the pictures, the detection algorithm will try and find the ArUcos. The amount of ArUcos found at each distance are shown in Figure 5.4.

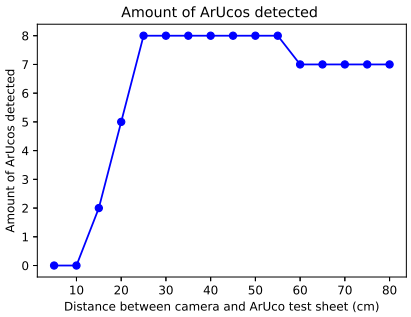
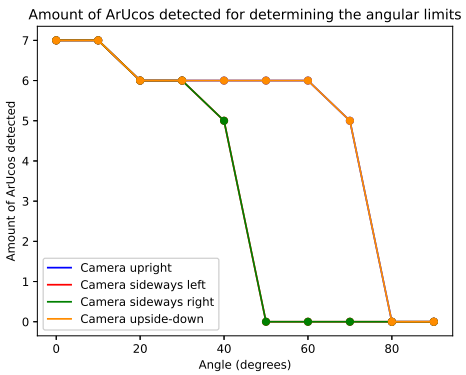


Figure 5.4: Amount of ArUcos detected for different distances using the ArUco test sheet (Appendix H.2)

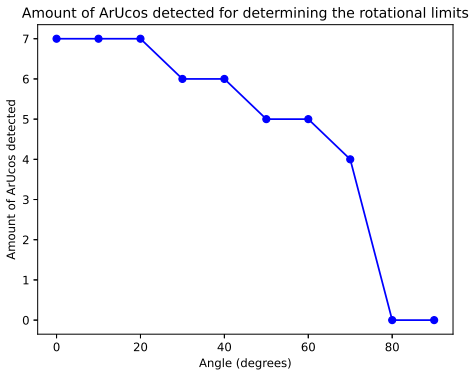
The ‘bad’ detection at very close distances (< 25cm) is due to the fact that at these short distances not all the ArUco codes from the test sheet are within the frame. At the larger distances (> 25 cm) almost all the ArUcos can be detected most of the time, with the exception of the smallest ArUco codes which could not be detected from a distance of 60cm.

This means markers from the size 0.75x0.75 inch and larger could be used within the ADome, given that the marker is located in a straight line to the camera. The marker of size 1x1 inch will be used within the ADome, since the larger the landmark the easier it is to detect, while still not taking too much space.

Angle of operation The second test to determine the limitations of the ArUco recognition, determines the angles at which the detection algorithm is still working. Here the camera is placed at a fixed position at the center of a protractor, after which the ArUco test sheet is moved along the edge of the protractor. At every 10 degrees of the protractor the testsheet is placed after which the ArUcos will be recognized at that point. The schematic test setup is shown in Appendix G.2. After each test the camera is rotated 90 degrees on its side and the test is repeated. This rotation ensures that the algorithm is tested in all cardinal directions. Also a test is done which rotates the test sheet around it own axis, as shown in Appendix G.3, to see how well the detection algorithm could recognize such rotated images. The results of these tests are shown in Figure 5.5a and 5.5b respectively.



(a) Results showing how many ArUcos of the test sheet are still recognized at what camera rotation



(b) Results showing how many ArUcos of the test sheet are recognized when the test sheet is rotated with respect to the camera

Figure 5.5: Test results of ArUco recognition for the angle of operation section

The drop at 80 degrees for upright and upside-down, and the drop at 50 for the sideways in Figure 5.5a is due to the test sheet falling outside the FoV of the camera. Therefore none of the markers will be detected then.

Most of the ArUco codes on the test sheet are visible, when limiting the region of interest from the full image to only the region of interest as discussed in Chapter 4. This means there will always be at

least 5 markers recognizable, if it is not rotated extremely. Since each of the markers has a different size, now the choice of marker, which can be used within the dome can be made.

If the camera is located roughly in the center of the ADome, the rotation of the landmarks would approximately face the camera, therefore the rotation of the landmarks would be very small. The landmark that is going to be used, could still be recognized up to a rotation of 40 degrees, as shown in Figure 5.5b, suggesting (rotation wise) the camera could always locate the landmarks within the ADome.

Finally the region of interest used by the camera, limits the field of view of the camera. The full field of view of the camera is 160 degrees (Appendix B.1), which will be lowered due to this region of interest. The aspect ratio of the camera is approximately 2:1, suggesting the field of view horizontally is approximately twice as large as vertically. The usage of the region of interest as discussed in Section 4.1, would lower this field of view to at most 140 degrees horizontally, which would be 70 degrees in both direction. According to Figure 5.5a, the landmark used can still be detected at 70 degrees. Therefore, the usage of the ArUco code of size 1x1 inch would be the smallest landmark, which would still have full coverage over the region of interest.

5.3.2. Accuracy of the mapping from camera to dome

The mapping from the coordinate system of the camera to the coordinate system of the ADome will be performed in three steps. First, the distances from the camera to the three detected landmarks will be calculated. Second, the position of the camera will be approximated using these distances. And finally, using the position of the camera, the ADome is ”rotated” such that its coordinate system matches that of the camera.

The first step, as discussed in Section 5.2, has been fully developed and tested. The final two steps however need some background information. Due to time constraints, and the project taking place during the COVID-19 pandemic, there was no possibility to test the methods described in Sections 5.2.2 and 5.2.3 in real life. Because of this, it is impossible to prove that in particular the method discussed in Section 5.2.2 is capable of determining the position of the camera.

Due to this situation, the methods described in Sections 5.2.2 and 5.2.3 will be delivered as theoretical, and should (in the future) be tested in real life as well. It must be said that, as visible in Figure 5.3, the method for finding the location of the antenna does work. However, since this method only relies on the accuracy of both Sections 5.2 and 5.2.2, no results will be presented other than the conclusion that this method should work, as long as the previous methods supply accurate information.

Finding distances from the camera to 3 landmarks

In order to test the accuracy of this method, positions of the landmarks and camera were simulated in MATLAB, and tests were performed using the following procedure. For each test, 3 different landmarks were selected. After this, the camera was positioned at different positions in the 60 cm radius ADome and different viewing angles. For each of the camera’s position, the angles to the landmarks were determined. These, and the distance between two landmarks, were then used to estimate the distance between the landmarks and the camera.

Testing the method in a simulation environment under perfect circumstances, the system was able to estimate the distances equal to the real distances to each landmark perfectly. This means that if the angles to each marker are accurate enough, the distances can be measured perfectly. However, it could be that the angles to the markers have a slight offset, which could cause the system to be less stable.

To fully test the stability of this system, the spherical angles θ and ϕ were altered multiple times by a random variable. This variable was generated using a normal distribution such that the error of these angles matched (at maximum) the errors discussed in Section 4.5.3. These altered angles were then used to estimate the distances between the landmarks and the camera. The results of these tests can be found in Table 5.1.

Table 5.1: Inaccuracies in meters after finding distance to camera using random errors in detected angles

Mean	Variance	Median	Standard Deviation	Min	Max
0.0266	0.0008	0.0150	0.0283	0.0011	0.0902

The first thing that can be concluded is that at maximum, the errors can induce an inaccuracy of approximately 10 cm. This is a relatively high value, however comparing this value to the mean distance, which is equal to approximately 3 cm, this is probably due to the errors being relatively large. This is confirmed, when looking at the following data in Table 5.2, as these error offsets are located on a large range.

One might conclude that, following this data, errors of around 1 degree make the system unstable. However, comparing this to the data presented in Table 5.3, this is not the case, as these values only reach to an inaccuracy of maximum 3 cm.

Table 5.2: Error values in degrees for inaccuracy of 10 cm

Angle	Marker 1	Marker 2	Marker 3
θ	1.171	-0.729	-0.924
ϕ	-0.814	1.488	-0.621

Table 5.3: Inaccuracies in meters for multiple absolute offsets of 1 degree

Offsets for θ	Offsets for ϕ	Minimum inaccuracy	Maximum inaccuracy
$[1, 1, 1]$	$[1, 1, 1]$	0.006	0.03
$[1, -1, 1]$	$[1, -1, 1]$	0.005	0.02
$[-1, 1, -1]$	$[-1, 1, -1]$	0.005	0.02
$[1, 1, 1]$	$[-1, -1, -1]$	0.001	0.02

Although these results are promising, the method should still be heavily tested in real life scenarios, in order to confirm that the inaccuracy is on average at maximum 3 cm.

6

Prototype implementation and validation results

In the previous chapters, the implementation of several different parts of the full prototype have been discussed. These parts work together to locate the antennas, find the position of the camera, and transform the pixel coordinates to the spherical coordinates of the antennas.

In this chapter, the full system will be discussed. The combination of all the parts, and how these parts communicate will be analysed in this chapter. Next to this, the system itself, including the camera setup and filters, will be discussed. Finally, a method will be proposed to determine each antenna in the ADome using the presented system.

6.1. Complete optical calibration system

In Chapters 3 to 5, methods have been developed to find the spherical coordinates of the antennas in the ADome. In each of these chapters, a single algorithm was presented that performs one of the following key processes necessary to find these coordinates. First, in Chapter 3, a method for locating the antennas has been presented. This method uses color detection to detect the LEDs connected to the antenna. For this process, as described in Section 3.3, the algorithm requires two images, one with the LED turned on, and one with the LED turned off. Using these images, the pixel coordinates of the LED will be returned, when the LED has been located.

The coordinates returned by the `locateLED` algorithm are Cartesian coordinates, where position $(0, 0)$ is located in the top left corner of the image. Since these coordinates are not the requested coordinates, namely spherical coordinates, a method was developed in Chapter 4 which transforms the Cartesian coordinates to spherical coordinates.

To perform this transformation, the camera is calibrated beforehand using the calibration grid in Figure H.1, such that certain pixel coordinates can be connected to spherical coordinates. The `findCoords` algorithm will use this calibration to return the transformed spherical coordinates. These spherical coordinates, are relative to the camera's position and orientation, not to the coordinate system of the ADome itself. Therefore, these coordinates are passed to a method which transforms these coordinates.

This transformation is done by detecting at least 3 different landmarks, the landmarks being so called ArUco landmarks. Using these landmarks, the location and orientation of the camera is determined, which is discussed in Chapter 5. Now that the real location and orientation is known, the spherical coordinates can be transformed, such that they fit into the coordinate system of the ADome.

This full system, including checking whether any antenna has been detected, is visible in Figure 6.1.

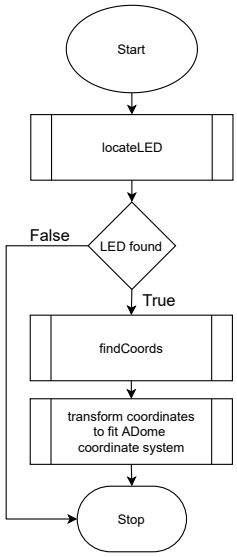


Figure 6.1: Flowchart describing the complete system

Although this described system is able to locate all the antennas in the ADome, one of the requirements was that the system should also be able to estimate the rotation of the antennas themselves. Although a proposal for such method has been created, which will be discussed later on, this does not yet have an implementation. Due to the antenna PCB designs being currently in a very early stage, it was difficult to develop a method which would be able to work on any PCB. Therefore, no method was yet implemented and should be implemented in future research.

Estimating the rotation of an antenna In order to estimate the rotation of an antenna, at least two markers (in this case LEDs) should be attached to the antenna. When both LEDs are detected and their spherical coordinates are estimated, the difference in locations in combination with the locations of the LEDs on the PCB can be used to estimate the rotation of the antenna. Since the antenna itself can, when viewed from certain angles, block the vision of the camera to some of the LEDs, the PCB might be fitted with more than 2 LEDs. This is done such that it will be possible to always detect at least two LEDs.

6.2. Full system setup

In this section, the hardware side of the system will be discussed. This section will discuss features that are required for the OCS to work, and some features that have been used in this prototype but are not certainly necessary or can be replaced for other parts.

The camera setup Since the system is a computer vision system, one of the most important components is the camera setup. For this system the camera described in Section B.1 is used (see Figure 6.2a). It is a 2MP camera with a 160 degree field of view. The wide field of view allows the system to see more of the ADome at once, allowing faster recognition of the antennas. Although the wide field of view might be useful, it is not a requirement, as the system should work with any type of camera.

Apart from the camera itself, a filter was attached to the camera (see Figure 6.2c). As mentioned in Chapter 3, when the LED shines directly into the camera, it causes over-saturation at the location of the LED, and a lens flare around the LED. This does not only make detecting the LED more difficult, it also decreases the accuracy as shown in Section 3.4.2. To reduce these effects, a Neutral Density (ND-)filter was attached to the camera. This filter reduces the intensity of light, and therefore decreases over-saturation. Next to this, since the used ND-filter is a variable ND-filter, the strength of the filter could be set manually. This allows the filter to either let through a lot of light, or almost no light at all. When set correctly, it lets through almost no other light, meaning that only the LED itself is left, increasing the chances of recognizing the LED.

Since the camera itself is rather small, see Figure 6.2a, the filter used could not be attached to the camera directly. Therefore, a mount was designed that allows connecting the filter to the PCB of the camera (see Figure 6.2b).

(a) The camera used for this prototype (b) Mount for the ND-filter (c) ND-filter, mount and camera combined

Figure 6.2: Camera system used in this prototype

The computer setup In order to meet one of the requirements set in Chapter 2, the software had to be designed such that it could be executed on a Raspberry Pi as well. To meet this requirement, the environment chosen to execute the computer vision is using the computer vision library OpenCV [1]. The OpenCV library was selected for multiple reasons:

- The library is open source, this allows anyone to use the library.
- The library can run with both Python as C++. Since C++ is a relatively low-level language, the Raspberry Pi will be able to support the written software.
- Also since the library runs on C++, the computation times can be kept low, in comparison to Python running OpenCV or even Matlab running the computer vision toolbox [28].

The computer setup itself will also be able to communicate with the antennas. As mentioned in the introduction of Chapter 3, one of the reasons LEDs were chosen as the markers to detect, is that the LEDs can be turned on and off. This allows to detect one antenna at a time. To allow the LEDs to be controlled, one feature of the antennas should be mentioned. In order to record data of the antennas, microcontrollers will be placed on the PCB of the antenna, which will communicate over the CAN-bus principle [29]. This means that not only data can be send to and from the microcontrollers, it also allows commands to be send to the antennas, such as enabling an LED fitted on the PCB. Therefore, the computer (or Raspberry Pi) will be connected to the antenna network via a serial connection to: send calibration data, request the amount of antennas connected, and to turn LEDs on and off.

Gimbal A static camera, although having a 160 degree field of view, will not be able to see the whole ADome at once. It would be possible to move the camera by hand, but the main idea of the OCS is that the system can work autonomously. Therefore, the camera will be mounted on top of a gimbal, which allows rotation and tilt to move the camera through the ADome.

6.3. Method for detecting all antennas in the ADome

Although the full system for detecting antennas has been developed, see Section 6.1, this does yet only detect a single antenna. This means that, if placed in the ADome, it expects that the antenna is placed in front of the camera, such that it can be detected. Therefore, in this section, a method is proposed to determine each antenna in the ADome. For this method, the following considerations should be kept in mind.

1. The OCS has an active connection with the microcontrollers connected to each antenna. It can request the amount of antennas connected, and control the LED of each antenna individually.
2. The OCS will be placed on a controllable gimbal, which allows the system to rotate and tilt the camera in order to view to full ADome.
3. The ADome will be fitted with enough ArUco landmarks, in range of the field of view of the camera, such that each possible image taken shows at least 3 landmarks.

The system described in this section (see Figure 6.3) follows these requirements, and therefore works when at least the mentioned requirements are met.

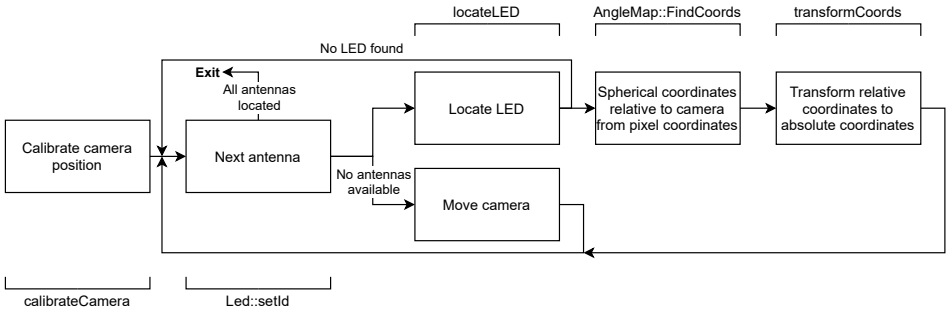


Figure 6.3: Block diagram describing the method for calibrating each antenna in the ADome

The first step in the detection process, is determining the location of the camera. Although it would be possible for the camera to be fitted in the center of the ADome perfectly, rotating or tilting the camera could cause the camera to shift from this perfect $(0, 0, 0)$. Therefore, the camera localization method discussed in Section 5.2 will be executed at first. After the camera has been localized, and assuming that with the current orientation antennas are in the field of view, the main process can start. As visible in Figure 6.3, the main process starts by selecting the next antenna in the list. After selecting this antenna, the process described in Figure 6.1 will start. The OCS will attempt to recognize an antenna. If this attempt fails, it determines that the current antenna is not in the field of view of the camera. Therefore, it will select the next antenna in the list, and the main process starts all over again.

When an antenna is in the field of view of the camera, the next two steps in Figure 6.3 will be executed. The antenna coordinates will be transformed to its spherical coordinates, and these coordinates will be stored on the microcontroller of the respective antenna. After this, the next antenna in the list is selected and the main process starts all over.

When all possible antennas have been selected, but not all antennas have been calibrated yet, the camera will be moved using the gimbal (as visible in Figure 6.3). The position of the camera will be calibrated again, since moving the camera can change its position, and the main process starts all over.

This process will repeat until every antenna has been calibrated.

7

Discussion, conclusion, and future research

This chapter will give a conclusion on the project. First a conclusion is given on Chapters 3 to Chapter 5. This conclusion will also mention how the results of these chapters correspond to the Programme of Requirements (Chapter 2). After this conclusion, a short paragraph is written with regards to how this would fit within the whole ADome project. Finally, a few suggestions will be given to further improve the design of this optical calibration system.

7.1. Discussion and conclusion

The first step in determining the true location of an antenna is finding the LED markers, which are located on the antennas. Chapter 3 described the methods regarding this recognition. The combination of the blob detection, color determination, automatic thresholding led to an already fairly accurate recognition of LEDs in images. The accuracy of the recognition was then increased by a noise reduction algorithm and carefully selecting the blobs based on their circularity. This satisfies mandatory requirement 2a: *The OCS must detect markers in the form of LEDs using computer vision*. The LEDs can be recognized in both light and dark environments, due to the blinking LED method. This method was designed specifically for the light environment, but also works in dark environments (for dark environments a simple blob detection would have sufficed). This satisfies mandatory requirement 1c: *The OCS must work in both dark and light environments*.

The second step is determining the location of the antennas in the coordinate system of the camera. This is achieved by first creating an AngleMap, which maps each pixel in the image taken by the camera, to a specific specific coordinate (ϕ, θ), disregarding the radius. The (ϕ, θ) according to such an AngleMap have an average deviation of 0.5 degrees. Using this AngleMap the location of the antennas, as well as the landmarks can be placed in the coordinate system of the camera. This part did not meet any requirements, since none of the requirements were directly related to this part.

The third step is mapping the location from the coordinate system of the camera, to the coordinate system of the ADome. This would satisfy mandatory requirement 2c: *The OCS must return the location of the landmarks and the LEDs in spherical coordinates*. The first step in transforming from the coordinate system of the camera, to the coordinate system of the ADome is recognizing the landmarks, so the location and orientation of the camera can be determined. The recognition of the landmarks, which were ArUco fiducial codes, and the decoding of these was handled by OpenCV. The testing showed that the recognition and decoding of these fiducials, would operate within the environment of the ADome. Therefore mandatory requirement 2b: *The OCS must detect landmarks in the form of ArUcos using computer vision* was satisfied.

These landmarks could then be used to find the location and orientation of the camera. Using the locations of the landmarks in the coordinate system of the ADome, in comparison to the location of the landmarks in the coordinate system of the camera (in this system only θ and ϕ are known), the distances between the camera and the landmarks could be determined. This distance approximation would deviate at most 2cm for each of the 3 distances, when the θ and ϕ with respect to the camera

would have an error of 1 degree. Sadly, due to time constraints, the location determination using these distances has not been tested and implemented completely. Therefore we can only make assumptions from here on out. The estimation of the location and the estimation of the orientation will both introduce some form of error.

Mandatory requirement 1a: *The OCS must locate the antennas within 0.1 degrees* stated the maximum error in the final spherical coordinate. As mentioned before, the AngleMap introduces an error of 0.5 degrees, the distance estimation can be off by multiple centimeters, and the final location estimation would introduce even more error. Summing all of these, the conclusion can be drawn, using this design for this system, that the OCS would not satisfy this requirement. Plans for future research have been made to solve this issue, and will be discussed in Section 7.3.

The final two mandatory requirements are 2d and 1b. Mandatory requirement 2d: *The OCS must return the orientation of the antenna* has not been implemented, since the estimation of the location of the antennas had priority, and implementing a method was difficult since the current PCB is in an early stage of development. Further research will therefore carry out the implementation of the method described in Section 6.1.

Chapter 6 already proposed a method for mandatory requirement 1b: *The OCS must be able to recognize any number of antennas within the ADome*. This method has not been implemented since (again) the priority was with finding the location of a single antenna, before scaling to multiple antennas.

7.2. Relation to the ADome

As mentioned in Section 1.1 the current of method of determining the location of all the antennas was done by hand, which was a time consuming, inaccurate method. The concept of the Optical Calibration System given in this thesis could be solution for this, since, while not fully finished, it is shown there is potential in finding the location of all the antenna accurately using this computer vision method. Further research must show, whether usage of the algorithm given to find the location and orientation of the camera, would also lead to the desired result.

With further research it should be possible to show methods for improving the accuracies of the other algorithms, to find the antennas within an accuracy of 0.1 degree (mandatory requirement 1a).

7.3. Future research

As discussed, the method presented complies with most of the requirements, with a few exceptions. To make the OCS comply with each requirement set, future research is necessary.

One of the key requirements that should be met, is that the system is able to return the spherical coordinates of the antennas. As the method for transforming the coordinate system of the spherical coordinates is yet unproven in real-life situations, the first continuation of this project should be to research whether the proposed method works, and whether the method should be improved or not.

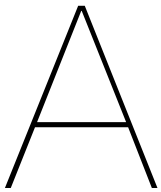
The next key requirement that should be improved, is the accuracy of the localization. The requirement was set at an accuracy of 0.1 degrees which, as discussed earlier in this chapter, is not achievable in the current situation. Due to, among other things, the resolution of the camera and the AngleMap of the camera. Therefore, other cameras, lenses, and filter should be tested. Next to this, the calibration of the camera should be further researched, especially whether a less wide field of view would improve the accuracy of the angle mapping, since a wider field of view causes a lower density of pixels per degree.

Finally, the developed OCS is unable to determine the rotation of the antenna itself. Therefore, possibilities should be researched of utilizing more LEDs on one PCB, which can then be measured to estimate the rotation of the antenna.

The previous suggestions for further research were suggestions on improving already discussed methods; next a suggestion will be given for a completely different method, which might also improve accuracy.

Instead of finding the AngleMap of the camera, research could be done to a completely different method, the 'pin pointing' method. This method would rotate the center of the image towards the marker on the antenna using a precision gimbal. Once the camera is pointed correctly to the antenna, the rotation and orientation of the gimbal may be used as angles θ and ϕ which would be used further in the process. This method still uses the antenna recognition methods discussed in Chapter 3 and the

mapping of the location of the antenna in the coordinate system of the camera (or in this case gimbal) to the coordinate system of the ADome.



Software

All the software written for this project has been written in C++. This software is stored on a git repository and can be found at `https://gitlab.com/daan99/adome`.

In order to execute the software written, the following steps have to be taken.

1. Make sure that OpenCV is installed [1].
2. Make sure that a g++ compiler is installed.
3. Make sure that cmake is installed.
4. Compile the source code using `cmake`. The master branch contains a `CMakeLists.txt` to compile all the files.

The current state of the code contains all the parts ranging from detecting an antenna and finding the spherical coordinates with respect to the camera. The transformation between the camera's coordinate system and the coordinate system of the ADome are located in the `matlab` sub-folder, and will be translated to C++ in the future, once the methods described in this thesis have been tested in a real life scenario.

B

Hardware

B.1. Camera

The camera used during this project is the Arducam B0202.

B.1.1. Specifications

- USB-camera
- 160 degree field of view fisheye lens
- Resolution of 2MP
- Video resolution of 1920x1080@30FPS
- Dynamic range of 80DB
- Software controllable exposure, brightness, and contrast.
- Can capture at a minimum illumination of 0.001 Lux

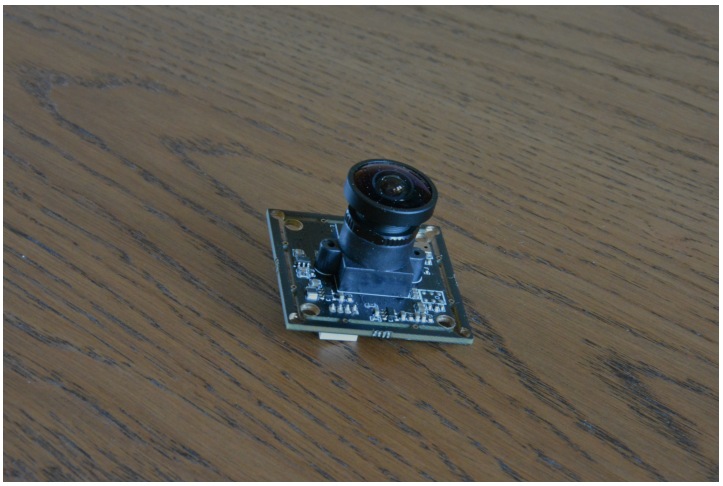


Figure B.1: The camera used in this project

C

Mathematical derivations

C.1. Derivation for the circularity formula

There are several ways to define a circle, its area, its circumference and its radius. The latter can be calculated in two ways, as visible in Equation C.1.

$$r = \sqrt{\frac{\text{Area}}{\pi}}$$
$$r = \frac{\text{Circumference}}{2\pi} \tag{C.1}$$

Since we can calculate the radii in two different ways, we can show that the closer both radii are to each other, the closer the object reassembles a circle. Therefore, by stating the equations to be equal, Equation C.2 is obtained.

$$\sqrt{\frac{\text{Area}}{\pi}} = \frac{\text{Circumference}}{2\pi}$$
$$\frac{\text{Area}}{\pi} = \frac{\text{Circumference}^2}{4\pi^2} \tag{C.2}$$
$$\frac{4\pi\text{Area}}{\text{Circumference}^2} = 1$$

Therefore, the closer the object is to a circle, the more the relationship in Equation C.2 approaches 1. This can therefore be rewritten to the general formula in Equation C.3, which states that, the closer its solution is to 1, the closer it is to a circle.

$$C = \frac{4\pi\text{Area}}{\text{Circumference}^2} \tag{C.3}$$

C.2. Planar fit through N points

This section will describe the process of fitting a plane through a set on N points.

The z-value of a plane van be written as:

$$z(x,y) = A \cdot x + B \cdot y + C \tag{C.4}$$

A, B and C are at this point still unknown. To find the value of A, B and C, Equation C.5 [30].equation has to be solved.

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} y_i^2 & \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} y_i & \sum_{i=1}^{n} 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \sum_{i=1}^{n} x_i z_i \\ \sum_{i=1}^{n} y_i z_i \\ \sum_{i=1}^{n} z_i \end{bmatrix} \tag{C.5}$$

In Equation C.5, the values x_i, y_i and z_i correspond to the coordinates from the points through which the plane is fitted.

The quality of the fit can be calculated using C.6. The closer the value to 1 the better the quality.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (z_i - z(x_i, y_i))^2}{\sum_{i=1}^{n} \left(z_i - \frac{1}{n} \sum_{i=1}^{n} z_i \right)^2} \tag{C.6}$$

Every plane has its tip and tilt angles. These angles correspond to how much z changes with respect to y and x. These angles can be calculated using Equation C.7.

$$R_x = \arctan\left(\frac{\partial}{\partial y} z(x,y) \right) = \arctan(B) \tag{C.7a}$$

$$R_y = \arctan\left(-\frac{\partial}{\partial x} z(x,y) \right) = \arctan(-A) \tag{C.7b}$$

C.3. Derivation for estimating distance between camera and 3 landmarks

Although the landmarks are located in a 3 dimensional system, the relationship between the camera and the two landmarks can be simplified by viewing the system as a triangle. As visible in Figure C.1, the distances from the camera to the two landmarks M_i and M_j, x_i and x_j respectively, and the distance between the two landmarks d_{ij} form a triangle with viewing angle θ_{ij}. Since the new reference frame now consists of a triangle, Equation C.8 can be used as a base for finding the distances x_i and x_j. Since the cartesian coordinates of the landmarks in the ADome are known, the distance between two landmarks d_{ij} is known as well. Because of this, Equation C.8 can be rewritten to the equality in Equation C.9, this means that this equality can be used in a system of equations to solve for the distances x_i and x_j.

$$d_{ij}^2 = x_i^2 + x_j^2 - 2x_i x_j \cos(\theta_{ij}) \tag{C.8}$$

$$f(x_i, x_j) = x_i^2 + x_j^2 - 2x_i x_j \cos(\theta_{ij}) - d_{ij}^2 = 0 \tag{C.9}$$

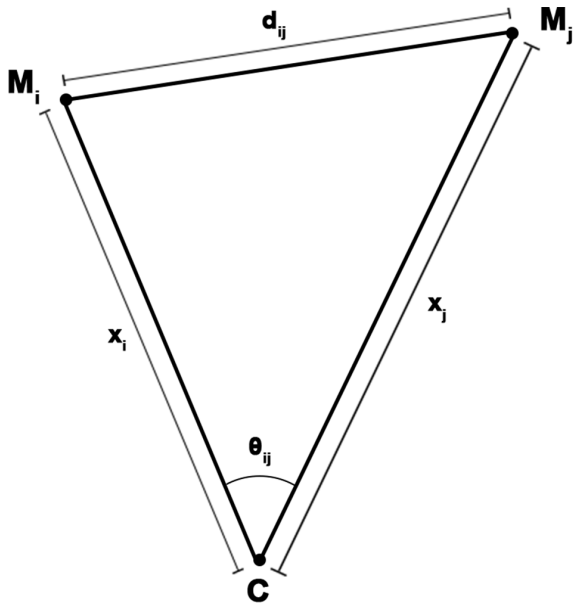


Figure C.1: Relationship between two landmarks and the camera

Although the angle θ_{ij} is not directly known, using the algorithms described in Chapters 3 and 4 the spherical angles (θ_i,ϕ_i) can be found. These angles in combination with the yet unknown distances x_i and x_j can be used to construct an equation for $\cos(\theta_{ij})$ using Equation C.10.

$$\cos(\theta_{ij}) = \frac{\vec{u}_{C,M_i} \cdot \vec{u}_{C,M_j}}{||\vec{u}_{C,M_i}||||\vec{u}_{C,M_j}||} \tag{C.10}$$

As visible in Equation C.11, the angles found, with the yet unknown distances can be used as spherical coordinates, and transformed back to cartesian. Since the coordinates are calculated using spherical coordinates, the magnitudes of vectors $\vec{u}_{C,M_i}$ and $\vec{u}_{C,M_j}$ are equal to x_i and x_j respectively. This means that Equation C.9 can be rewritten to Equation C.12. Using this equation, the only unknown values are x_i and x_j. These values can be found, when at least 3 landmarks are used. In this case the system of equations from Equation C.13 is created.

$$\vec{u}_{C,M_i} = [x_i \cos(\phi_i) \sin(\theta_i), x_i \sin(\phi_i) \sin(\theta_i), x_i \cos(\theta_i)] \tag{C.11a}$$
$$\vec{u}_{C,M_j} = [x_j \cos(\phi_j) \sin(\theta_j), x_j \sin(\phi_j) \sin(\theta_j), x_j \cos(\theta_j)] \tag{C.11b}$$
$$||\vec{u}_{C,M_i}|| = x_i \tag{C.11c}$$
$$||\vec{u}_{C,M_j}|| = x_j \tag{C.11d}$$

$$f(x_i, x_j) = x_i^2 + x_j^2 - 2(\vec{u}_{C,M_i} \cdot \vec{u}_{C,M_j}) - d_{ij}^2 = 0 \tag{C.12}$$

$$\begin{cases} f(x_1, x_2) = 0 \\ f(x_1, x_3) = 0 \\ f(x_2, x_3) = 0 \end{cases} \tag{C.13}$$

To solve this system, first x_3 has to be eliminated between $f(x_1, x_3)$ and $f(x_2, x_3)$. This is done by using Sylvester's resultant method [26]. This method states that, if two polynomials are transformed in the Sylvester matrix, its determinant should be equal to zero if and only if the two polynomials have a common zero [26]. Since the polynomials hold a same variable, and come from the same system of equations, this statement applies.

With this knowledge, the Sylvester matrix has to be created. First, the $f(x_i, x_j)$ equation has to be rewritten to form a polynomial, as visible in Equation C.14, after that, the Sylvester matrix can be created from its coefficients (see Equation C.15). The matrix will be a 4 × 4 matrix, since both polynomials are second order polynomials, resulting in a $(2 + 2) \times (2 + 2)$ matrix [26].

$$f(x_i, x_j) = x_j^2 + (-2(\vec{u}_{C,M_i} \cdot (\frac{\vec{u}_{C,M_j}}{x_j})))x_j + (x_i^2 - d_{ij}^2) \tag{C.14}$$

$$S = \begin{bmatrix} 1 & -2(\vec{u}_{C,M_1} \cdot (\frac{\vec{u}_{C,M_3}}{x_3})) & x_1^2 - d13^2 & 0 \\ 0 & 1 & -2(\vec{u}_{C,M_1} \cdot (\frac{\vec{u}_{C,M_3}}{x_3})) & x_1^2 - d_{13}^2 \\ 1 & -2(\vec{u}_{C,M_2} \cdot (\frac{\vec{u}_{C,M_3}}{x_3})) & x_2^2 - d_{23}^2 & 0 \\ 0 & 1 & -2(\vec{u}_{C,M_2} \cdot (\frac{\vec{u}_{C,M_3}}{x_3})) & x_2^2 - d_{23}^2 \end{bmatrix} \tag{C.15}$$

Calculating the determinant of this matrix will create a polynomial $h(x_1, x_2)$ that should be equal to zero to hold. This means that the system of equations now changes to the one visible in Equation C.16. The polynomial $h(x_1, x_2)$ will be of order 4, since the matrix contain multiple components of x_1^2 and x_2^2, resulting in x_1^4 and x_2^4 at maximum after calculating the determinant.

$$\begin{cases} f(x_1, x_2) = 0 \\ h(x_1, x_2) = 0 \end{cases} \tag{C.16}$$

The same process for eliminating x_3 can now be used to eliminate x_2. From both functions, the coefficients of x_2 are extracted and used to create a new Sylvester matrix. This matrix will be a $(2 + 4) \times (2 + 4)$ matrix, and its determinant will result in a 8th order polynomial in terms of x_1, due to the determinant squaring x_1^4.

The resultant of these two polynomials will result in the polynomial $g(x_1)$, special about this polynomial is that only the even terms have nonzero coefficients. Because of this, the polynomial can be rewritten to a 4th order polynomial, by substituting $x = x_1^2$, this creates the 4th order polynomial $g(x)$. This polynomial, by means of the Sylvester resultant theorem, should be equal to 0, and by finding its roots, the value for the distance x_1 can be found by taking the square root of those roots.

This result can then be used to substitute in $h(x_1, x_2)$ to find x_2, and later in for example $f(x_1, x_3)$ to find x_3.

What should be noted, is that due to the polynomial $g(x)$ being a 4th order polynomial, 4 solutions will be found. Therefore, the correct solution still has to be selected.

D

Pseudocode

D.1. Distortion Estimation

Algorithm 1 Distortion estimation method 1

Normalize the point cloud
Find the furthest dot out of the red dots → X_1^u
Find the direct neighbor to X_1^u → X_2^u
$DISTANCE = |X_1 - X_2|$
while $|DISTANCE - 1| > 0.001$ **do**
    $K_1 = K_1 + dK_1$
    Distort(X_1^u, K_1) → X_1^d
    Distort(X_2^u, K_1) → X_2^d
    $DISTANCE = |X_1^d - X_2^d|$

Algorithm 2 Distortion estimation method 2

Normalize the point cloud
Find the furthest dot out of all the dots → X_1^u
Find the direct neighbor to X_1^u → X_2^u
$DISTANCE = |X_1 - X_2|$
while $|DISTANCE - 1| > 0.001$ **do**
    $K_1 = K_1 + dK_1$
    Distort(X_1^u, K_1) → X_1^d
    Distort(X_2^u, K_1) → X_2^d
    $DISTANCE = |X_1^d - X_2^d|$

Algorithm 3 Distortion estimation method 3

while Current Error < Previous Error **do**
    Previous Error = Current Error
    $K_1 = K_1 + dK_1$
    for All red dots X_1^u **do**
        Find direct neighbor of $X_1^u \rightarrow X_2^u$
        Distort(X_1^u, K1) $\rightarrow X_1^d$
        Distort(X_2^u, K1) $\rightarrow X_2^d$
        $DISTANCE = |X_1^d - X_1^d|$
        Local Error $= |DISTANCE - 1|$
        Error Sum $+=$ $\frac{\text{(Local Error)}^2}{N}$, with N the amount of dots (blue and red)
    RMS Error $= \sqrt{ErrorSum}$
    Current Error = RMS Error

Algorithm 4 Distortion estimation method 4

while Current Error < Previous Error **do**
    Previous Error = Current Error
    $K_1 = K_1 + dK_1$
    for All dots X_1^u **do**
        Find direct neighbor of $X_1^u \rightarrow X_2^u$
        Distort(X_1^u, K1) $\rightarrow X_1^d$
        Distort(X_2^u, K1) $\rightarrow X_2^d$
        $DISTANCE = |X_1^d - X_1^d|$
        Local Error $= |DISTANCE - 1|$
        Error Sum $+=$ $\frac{\text{(Local Error)}^2}{N}$, with N the amount of dots (blue and red)
    RMS Error $= \sqrt{ErrorSum}$
    Current Error = RMS Error

E

LED detection test data

E.1. Detection with ND-filter installed

Dataset 1	Real x value	Real y value	Detected x value	Detected y value
	770	393	769	392
	755	522	754	522
	755	516	752	515
	771	626	771	625

Table E.1: All data from the first dataset recorder with ND-filter installed

Dataset 2	Real x value	Real y value	Detected x value	Detected y value
	768	390	767	389
	896	385	895	383
	1019	388	1019	387
	1145	387	1144	386
	1264	386	1264	385
	1393	386	1393	385
	1495	386	1495	384

Table E.2: All data from the second dataset recorder with ND-filter installed

Dataset 3	Real x value	Real y value	Detected x value	Detected y value
	766	392	765	393
	649	393	649	393
	520	394	519	394
	396	395	394	397
	274	398	274	396
	150	400	149	401

Table E.3: All data from the third dataset recorder with ND-filter installed

Dataset 4	Real x value	Real y value	Detected x value	Detected y value
	766	426	764	427
	877	424	877	424
	889	431	889	431
	1014	427	1014	426
	1147	427	1145	427
	1277	427	1277	427
	1395	428	1393	427

Table E.4: All data from the fourth dataset recorder with ND-filter installed

E.2. Detection without ND-filter installed

Dataset 1	Real x value	Real y value	Detected x value	Detected y value
	845	416	845	414
	845	415	844	414
	843	543	841	540
	843	679	843	682
	842	801	841	795
	842	802	841	796
	843	674	843	674

Table E.5: All data from the first dataset recorder without ND-filter installed

Dataset 2	Real x value	Real y value	Detected x value	Detected y value
	795	358	798	364
	925	359	923	364
	1297	362	1299	364
	790	357	794	365
	1058	361	1063	356
	1183	362	1183	359

Table E.6: All data from the second dataset recorder without ND-filter installed

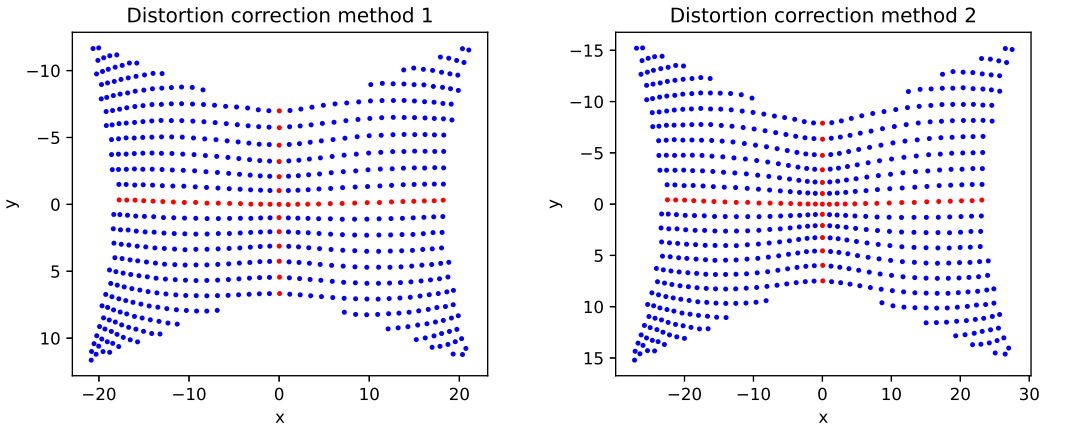
Dataset 3	Real x value	Real y value	Detected x value	Detected y value
	807	357	809	363
	673	355	676	358
	547	354	549	358
	421	353	419	351
	294	350	296	353

Table E.7: All data from the third dataset recorder without ND-filter installed

F

Enlarged figures

F.1. Distortion correction



(a) Distortion correction using $K_1 = 0.01047$. K_1 is found using distortion estimation method 1

(b) Distortion correction using $K_1 = 0.01631$. K_1 is found using distortion estimation method 2

(c) Distortion correction using $K_1 = 0.00628$. K_1 is found using distortion estimation method 3

(d) Distortion correction using $K_1 = 0.00865$. K_1 is found using distortion estimation method 4

Figure F.1: Enlarged version of Figure 4.4. Distortion correction on the point cloud shown in Figure 4.2b.

G

Test setups

G.1. Test setup for testing the limit and/or size limitations of ArUco detection algorithm

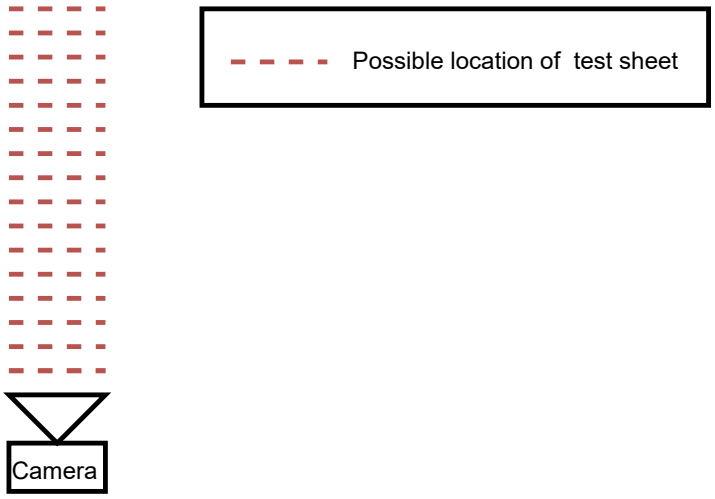


Figure G.1: Test setup for testing the limit and/or size limitations of ArUco detection algorithm. The distance between each possible location of test sheet is 5cm

G.2. Test setup for testing the angle limitation of the ArUco detection algorithm

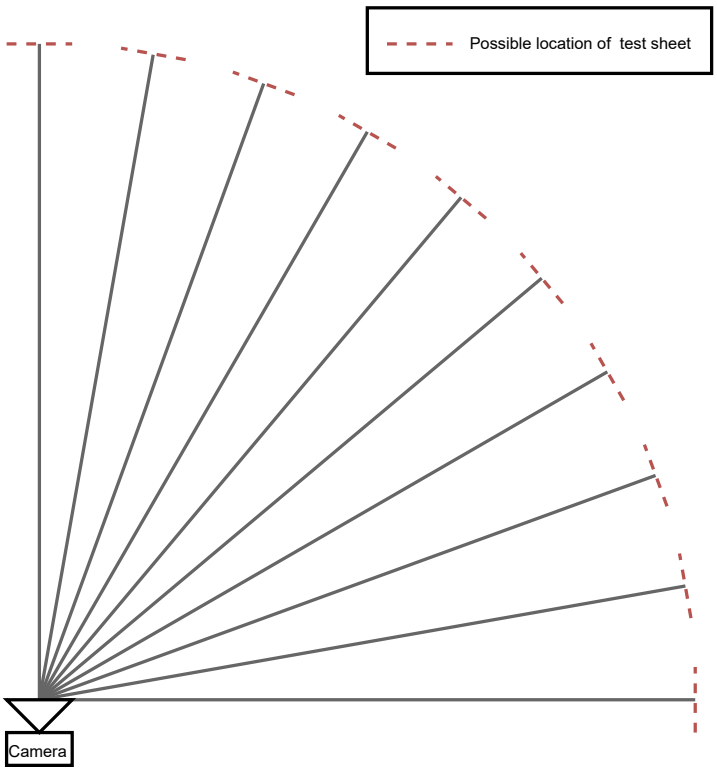


Figure G.2: Test setup for testing the angle limitation of the ArUco detection algorithm

G.3. Test setup for testing the roation limitation of the ArUco detection algorithm

Figure G.3: Test setup for testing the rotation limitation of the ArUco detection algorithm

H Images used for testing

H.1. Calibration grid

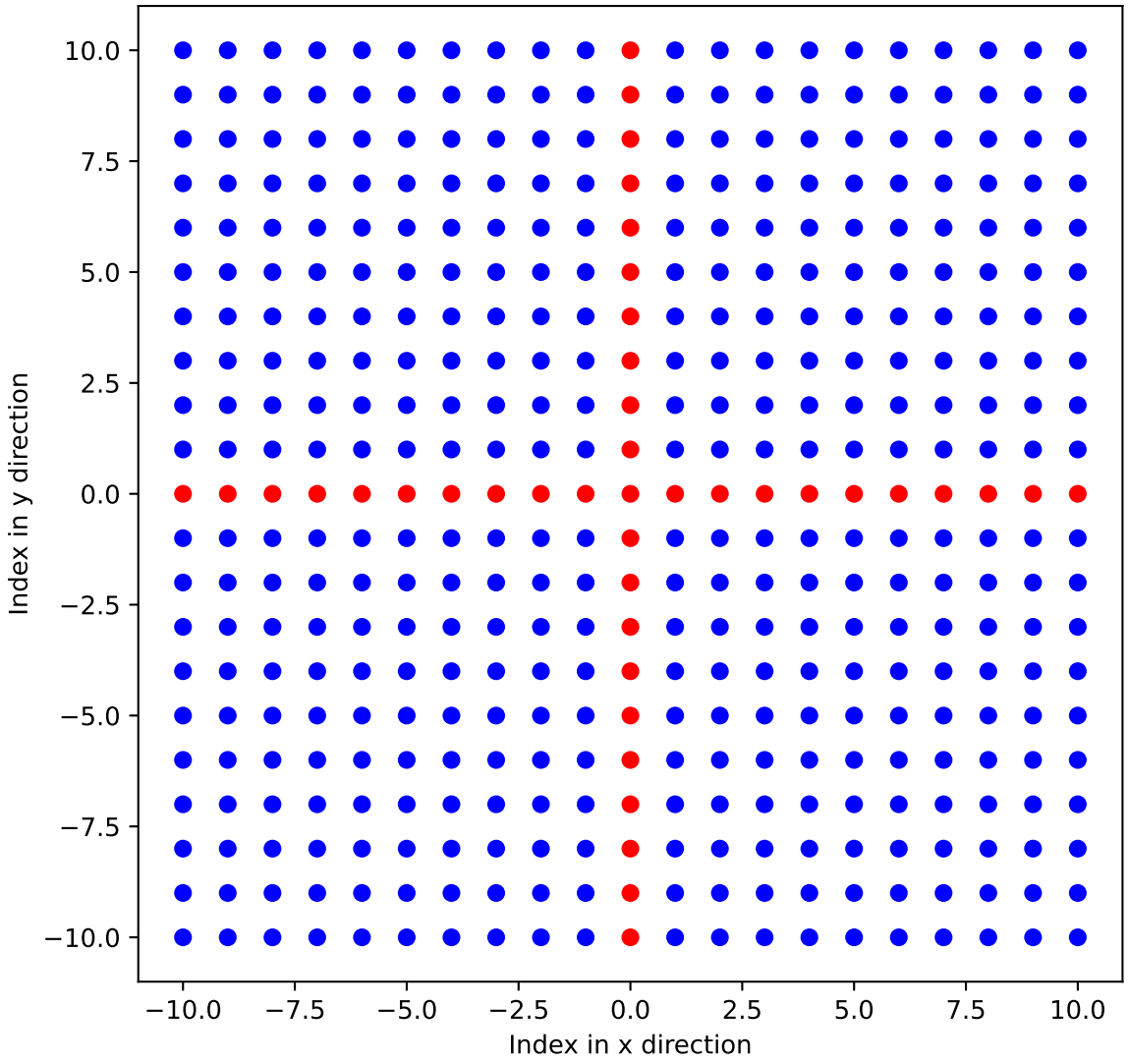


Figure H.1: Calibration grid used for determining the mapping of pixels to angles

H.2. Test sheet for ArUco detection

Figure H.2: Test sheet used for the testing of ArUco detection

Bibliography

[1] *Home*, May 2021. [Online]. Available: https://opencv.org/.

[2] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International journal of computer vision*, vol. 38, no. 1, pp. 15–33, 2000.

[3] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," 2013.

[4] Z. Jiang, Y. Liu, B. Wu, and Q. Zhu, "Monocular vision based uav target detection and ranging system implemented on opencv and tensor flow," in *2019 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, 2019, pp. 88–91. DOI: 10.1109/DCABES48411.2019.00029.

[5] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4193–4198. DOI: 10.1109/IROS.2016.7759617.

[6] M. Fiala, "Artag, a fiducial marker system using digital techniques," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, 2005, 590–596 vol. 2. DOI: 10.1109/CVPR.2005.74.

[7] A. Breitenmoser, L. Kneip, and R. Siegwart, "A monocular vision-based system for 6d relative robot localization," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 79–85. DOI: 10.1109/IROS.2011.6094851.

[8] W. Song, Z. Miao, and H. Wu, "Automatic calibration method based on improved camera calibration template," pp. 301–305, 2013. DOI: 10.1049/cp.2013.2429.

[9] L. Liu, S. Cao, X. Liu, and T. Li, "Camera calibration based on computer vision and measurement adjustment theory," pp. 671–676, 2018. DOI: 10.1109/IMCCC.2018.00145.

[10] Z. Zhang, "Camera calibration with one-dimensional objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 7, pp. 892–899, 2004. DOI: 10.1109/TPAMI.2004.21.

[11] W. Hong, H. Xia, X. An, and X. Liu, "Natural landmarks based localization algorithm for indoor robot with binocular vision," pp. 3313–3318, 2017. DOI: 10.1109/CCDC.2017.7979078.

[12] L.-F. Gao, Y.-X. Gai, and S. Fu, "Simultaneous localization and mapping for autonomous mobile robots using binocular stereo vision system," pp. 326–330, 2007. DOI: 10.1109/ICMA.2007.4303563.

[13] B. Han and L. Xu, "A monocular slam system with mask loop closing," pp. 4762–4768, 2020. DOI: 10.1109/CCDC49329.2020.9164164.

[14] G. Tuna, K. Gulez, V. Cagri Gungor, and T. Veli Mumcu, "Evaluations of different simultaneous localization and mapping (slam) algorithms," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, 2012, pp. 2693–2698. DOI: 10.1109/IECON.2012.6389151.

[15] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recognition*, vol. 42, no. 9, pp. 1977–1987, 2009, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2008.10.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320308004573.

[16] H. Kong, H. C. Akakin, and S. E. Sarma, "A generalized laplacian of gaussian filter for blob detection and its applications," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1719–1733, 2013. DOI: 10.1109/TSMCB.2012.2228639.

[17] L. Assirati, N. R. Silva, L. Berton, A. A. Lopes, and O. M. Bruno, "Performing edge detection by difference of gaussians using q-gaussian kernels," *Journal of Physics: Conference Series*, vol. 490, p. 012 020, Mar. 2014. DOI: 10.1088/1742-6596/490/1/012020. [Online]. Available: https://doi.org/10.1088/1742-6596/490/1/012020.

[18] T. Lindeberg, “Feature detection with automatic scale selection,” *International Journal of Computer Vision*, vol. 30, no. 2, pp. 79–116, 1998, QC 20111101. DOI: 10.1023/A:1008045108935. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-40224.

[19] E. Pierce, *HSV_cone*. Sep. 2005. [Online]. Available: https://nl.wikipedia.org/wiki/HSV_(kleurruimte).

[20] R. M. Haralick, S. R. Sternberg, and X. Zhuang, “Image analysis using mathematical morphology,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 4, pp. 532–550, 1987. DOI: 10.1109/TPAMI.1987.4767941.

[21] M. Lee, H. Kim, and J. Paik, “Correction of barrel distortion in fisheye lens images using image-based estimation of distortion parameters,” *IEEE Access*, vol. 7, pp. 45 723–45 733, 2019. DOI: 10.1109/ACCESS.2019.2908451.

[22] J. Park, S.-C. Byun, and B.-U. Lee, “Lens distortion correction using ideal image coordinates,” *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 987–991, 2009. DOI: 10.1109/TCE.2009.5278053.

[23] M. Fiala, “Artag revision 1. a fiducial marker system using digital techniques,” Nov. 2004, * published as NRC/ERB-1117. November 24, 2004. 46 Pages. NRC 47419.

[24] A. Zakiev, K. Shabalina, and E. Magid, “Pilot virtual experiments on aruco and apriltag systems comparison for fiducial marker rotation resistance,” 2019.

[25] L. Quan and Z. Lan, “Linear n-point camera pose determination,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 8, pp. 774–780, Aug. 1999, ISSN: 0162-8828. DOI: 10.1109/34.784291. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1109/34.784291.

[26] J. S. F. R.A.S., “Xxiii. a method of determining by mere inspection the derivatives from two equations of any degree,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 16, no. 101, pp. 132–135, 1840. DOI: 10.1080/14786444008649995. eprint: https://doi.org/10.1080/14786444008649995. [Online]. Available: https://doi.org/10.1080/14786444008649995.

[27] B. K. P. Horn, “Closed-form solution of absolute orientation using unit quaternions,” *J. Opt. Soc. Am. A*, vol. 4, no. 4, pp. 629–642, Apr. 1987. DOI: 10.1364/JOSAA.4.000629. [Online]. Available: http://josaa.osa.org/abstract.cfm?URI=josaa-4-4-629.

[28] S. Matuska, R. Hudec, and M. Benco, “The comparison of cpu time consumption for image processing algorithm in matlab and opencv,” in *2012 ELEKTRO*, 2012, pp. 75–78. DOI: 10.1109/ELEKTRO.2012.6225575.

[29] A. J. Becoy and R. Zhang, “Adome - design of a new readout protocol,” 2021.

[30] J. Innovation, *Fit plane through points*, Accessed: 21-05-2021, 2021. [Online]. Available: https://www.jpe-innovations.com/precision-point/fit-plane-through-points/.