

Development of an IoT Device for Reading Analog Dials Based on Artificial Vision

Nuno Gonçalo Vitória Bartolomeu

Dissertação de Mestrado

Orientador na FEUP: Prof. Joaquim Gabriel Magalhães Mendes

Co-orientador no INEGI: Ricardo Joel Carvalho



Mestrado em Engenharia Mecânica

Especialização em Automação

Março 2022

Abstract

In industry it is quite common to use analog dials for process monitoring. Given the evolution of new technologies there is, nowadays, an increasing pressure for the digitalization of the industry - Industry 4.0. Here it is proposed the development of a product for reading analog dials, with a camera, that can be universally used in industry.

In a perspective of the application of the Internet of Things (IoT), a microcontroller, a camera, the communication and Cloud service to be used, as well as the appropriate selection of lighting were chosen. A mount has also been designed for universal attachment on common industrial analog dials.

An algorithm capable of an automatic reading for industrial analog dials, based on computer vision, is developed. It detects the center, radius, beginning and end of the scale and the values present in the scale, for which techniques such as circle Hough transform, line Hough transform, and OCR algorithms were used.

A validation of the algorithm was done in this work using 10 different manometers, having successfully detected 87.5% of its fundamental characteristics for the correct computation of its value. The determination of the represented value, without any user input, obtained a maximum error of 1.6%. For these cases, there is the possibility of parameter adjustment by the user.

Keywords: Analog dial, IoT, Cloud, Computer Vision, Hough Transform, OCR.

Resumo

Na indústria é bastante comum a utilização de manómetros analógicos para monitorização de processos. Dada a evolução das novas tecnologias, existe atualmente uma pressão crescente para a digitalização da indústria - Indústria 4.0. Aqui é proposto o desenvolvimento de um produto para a leitura de manómetros analógicos, com uma câmara, que pode ser universalmente utilizado na indústria.

Numa perspetiva da aplicação da Internet das Coisas (IoT), foi feita a escolha de um microcontrolador, de uma câmara, da comunicação e serviço Cloud a usar, bem como a seleção adequada da iluminação. Foi também concebido um suporte para a fixação universal em manómetros analógicos industriais comuns.

Foi desenvolvido um algoritmo capaz de uma leitura automática para manómetros analógicos industriais, baseado na visão por computador. Este deteta o centro, o raio, o início e o fim da escala e os valores presentes na escala, para os quais foram utilizadas técnicas como a transformada de Hough de círculos, a transformada de Hough de linhas, e algoritmos de OCR.

Uma validação do algoritmo foi feita neste trabalho utilizando 10 manómetros diferentes, tendo sido detetado com sucesso 87,5% das suas características fundamentais para o correto cálculo do seu valor. A determinação do valor representado, sem qualquer input por parte do utilizador, obteve um erro máximo de 1,6%. Para estes casos, existe a possibilidade de ajuste de parâmetros pelo utilizador.

Palavras-chave: Manómetro analógico, IoT, Cloud, Visão computacional, Transformada de Hough, OCR.

Acknowledgments

Firstly, I would like to thank my supervisor, Prof. Joaquim Gabriel Magalhães Mendes, for opening the door for this project, being one of the main reasons why I choose this project, his knowledge and teaching allowed me to broaden my own personal knowledge in this field. It was a privilege to be present at all of his classes, one of the best and most enjoyable classes I've had during this academic path. His teaching and guidance will not be forgotten.

To my co-supervisor, Ricardo Joel Carvalho, who was always so available and helpful. His advices helped me more that he can imagine and I am forever grateful for his shared knowledge. Solutions and suggestions were constantly made in order for my work to be successfully done.

A sincere gratefulness to INEGI, allowing me to develop my work due to its equipments and materials that were always available for my own use. Furthermore, it was here that I was able to grow, to learn and to make mistakes.

To my friends, for always sticking with me and kept trying to cheer me up and show their support.

Finally, and most importantly, I would like to thank my family for their unconditional love and care during these last months. You never doubted my abilities and I hope to make you all proud.

This dissertation was developed under the project with the reference POCI-01-0247-FEDER- 047091 - GRS - Glartek Retrofit Sensors, cofinanced by Programa Operacional Competitividade e Internacionalização (COMPETE 2020), through Fundo Europeu de Desenvolvimento Regional (FEDER).

Cofinanciado por:



Contents

1. Introduction	1
1.1 Project background and motivation	1
1.2 Objective	2
1.3 Thesis structure	3
2. State of Art	4
2.1 Available technology and hardware evolution.....	4
2.2 Internet of Things	5
2.3 Edge devices	5
2.4 Camera lens	7
2.4.1 Lens distortion calibration	8
2.5 Lighting techniques	10
2.5.1 Light sources	11
2.5.2 Light conditionings	12
2.6 Communication	13
2.6.1 IoT communication protocols	14
2.6.2 Cloud services	15
2.7 Image processing	16
3. Prototype.....	18
3.1 Microcontroller.....	19
3.2 Camera Lens.....	21
3.3 Illumination.....	24
3.4 Communication	33
4. Algorithm development	34
4.1 Calibration.....	36
4.2 Dimensionality Reduction	37
4.3 Region of Interest detection	37
4.4 Center detection	38
4.5 Dial detection	39
4.6 Scale limits detection.....	40
4.7 Scale values detection	44
4.8 Symmetry detection.....	46
4.9 Value computation	47
5. Results	48
5.1 Prototype design.....	48
5.1.1 Structure design	48
5.2 Algorithm validation and limitations.....	54
6. Conclusion and Future Works	57
6.1 Conclusion	57
6.2 Future work	58
References.....	59
Annex A - LED diffuser characteristic graph (luminous intensity function of angle)	65
Annex B - Image calibration algorithm	66

Annex C - Scale limits detection's scan	68
Annex D - Analog dial reader algorithm.....	69
Annex E - Algorithm experimental results	87
Annex F - Algorithm variables to adjust the calculated value.....	89
Annex G - Analog dial reader algorithm, 2nd reading	91
Annex H - Prototype technical drawings.....	92

List of Figures

Figure 2.1 - CYPRESS ENVIROSSYSTEMS Gauge Reader applied in a testing environment [11].	6
Figure 2.2 - Schematic representation of the fundamental characteristics for image acquisition [20].	7
Figure 2.3 - Camera lens FOV, WD, AFOV and focal length parameters [20].	8
Figure 2.4 - Lens distortion external parameters coordinates illustration [21].	9
Figure 2.5 - Radial types of distortions [21].	10
Figure 2.6 - Advantages and disadvantages of different light sources,	11
Figure 2.7 - Dark-field vs Bright-field illustration [30].	12
Figure 2.8 - Light technics, a) Diffuse dome light, b) Flat dome light,	13
Figure 3.1 - Proposed architecture of the prototype under development.	18
Figure 3.2 - Microcontroller ESP-32 CAM [69].	20
Figure 3.3 - Lenses used in the experimental tests, a) 66.5° lens, b) 160° lens, c) 180° lens, d) 235° lens.	22
Figure 3.4 - Rehearsal performed lens choice, a) Working distance illustration, b) Support designed for testing.	23
Figure 3.5 - Lenses used in the experimental tests, a) 66.5° lens, a) 160° lens, a) 180° lens, d) 235° lens.	24
Figure 3.6 - Rehearsal performed for light choice.	25
Figure 3.7 - Light technics testing, a) bright field, b) dark field.	25
Figure 3.8 - LED diffusers experimental evaluation.	26
Figure 3.9 - Industrial gauge system.	27
Figure 3.10 - Diffuse dome light technique experiment (A - aperture for lens, B - opening for light, C - dome diameter).	28
Figure 3.11 - Diffuse flat dome light technique experiment (A - distance from LEDs).	28
Figure 3.12 - Light filter illustration. (A - distance from LEDs, B - distance from LED to light filter).	29
Figure 3.13 - Results from filtering LED light, a) and b) with fabric filter, c) and d) with acrylic filter.	29
Figure 3.14 - Distance from LED to acrylic experiment, a) 10mm, b) 20mm, c) 35mm.	30

Figure 3.15 - Distance from LED to acrylic experiment, a) 5mm, b) 10mm, c) 15mm, c) 20mm.	31
Figure 3.16 - Binarization with threshold = 90 of Figure 3.15 images, 1) image result, 2) histogram graph.	32
Figure 4.1 - Gauge images used to develop algorithm, a) white background gauge result, b) black background gauge result.	34
Figure 4.2 - Implemented workflow for detection of features present in an analog dial.	35
Figure 4.3 - Image calibration process, a) Image to calibrate, b) Checkerboard pattern recognition, c) Image calibrated.	36
Figure 4.4 - Binarization result images, a) white background gauge's results b) black background gauge's results.	37
Figure 4.5 - ROI result images, a) white background gauge's results b) black background gauge's results.	38
Figure 4.6 - Detected centers comparison results, left - black background gauge's results, right - white background gauge's results.	39
Figure 4.7 - Dial detection illustration, a) white background gauge's results b) black background gauge's results.	40
Figure 4.8 - Image polar coordinate transform, a) illustration of the area of interest, b) white background gauge polar transform c) black background gauge polar transform. ..	41
Figure 4.9 - Illustration of different scans performed to image, orange rectangle ($d \times 60$), a) white background gauge scan with d advances, b) white background gauge scan with $d/2$ advances, c) black background gauge scan with d advances, d) black background gauge scan with $d/2$ advances.	42
Figure 4.10 - Gauge position relative to polar coordinate system.	43
Figure 4.11 - Scale limits detection illustration, a) white background gauge's results b) black background gauge's results.	44
Figure 4.12 - OCR Tesseract result on inverted binary image, black background gauge.	45
Figure 4.13 -Tesseract Error rate function of Letter pixel height graph [66].	45
Figure 4.14 - OCR detection, a) white background gauge, b) black background gauge.	46
Figure 4.15 - Illustration of angular amplitude between detected values.	47
Figure 5.1 - Detailed view of ESP-32 on developed structure.	49
Figure 5.2 - Illustration of the LED lighting on the structure.	49
Figure 5.3 - Detailed view of the acrylic fixation.	50
Figure 5.4 - Product view in open position, using a hinge.	50
Figure 5.5 - a) Top view of prototype without upper cover, b) real image batteries support .	51

Figure 5.6 - Mounting details, a) real image from 40 mm support, b) real image from 70 mm support, c) exploded view of filling rims, d) tightening detail, e) detail view of filling rim fittings.	52
Figure 5.7 - Developed Prototype, a) complete view, b) sectional view, c) exploded view. ..	53
Figure 5.8 - Beginning of scale correction.	55
Figure 5.9 - OCR error illustration.	56
Figure 6.1 - LED diffuser characteristic's graph - luminous intensity function of angle	65
Figure 6.2 - Illustration of the 15 first scans performed on the image after polar transform..	68
Figure 6.3 - Illustration of the results of manometers 1 to 5 used in the validation of the algorithm.....	87
Figure 6.4 - Illustration of the results of manometers 6 to 10 used in the validation of the algorithm.....	88
Figure 6.5 - Scheme for interactive parameter setting.	90

List of Tables

Table 2.1 - Internet vs Bluetooth comparison..... 14

Table 2.2 - IoT communication protocols comparison. 15

Table 2.3 - Cloud services comparison. 16

Table 3.1 - ESP-32 CAM Specifications. 20

Table 4.1 - Characteristics of the gauges used to develop the algorithm..... 35

Table 4.2 - Image scan results - (N.D. = Non Detected, N.A. = Non Acceptable). 43

Table 5.1 - Algorithm validation results. 54

Abbreviations

AFOV	Angular Field of View
API	Application Programming Interface
BLE	Bluetooth Low Energy
CoAP	Constrained Application Protocol
CPU	Central Processing Units
CTF	Contrast Transfer Function
DOF	Depth of Field
FOV	Field of View
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IWSN	Industrial Wireless Sensor Network
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
MQTT	Message Queuing Telemetry Transport
OCR	Optical Character Recognition
OpenCV	Open Source Computer Vision
OS	Operating System
PSF	Point Spread Function
RAM	Random Access Memory
ROI	Region of Interest
SoC	System-on-Chip
SSL	Secure Socket Layers
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
WD	Working Distance

1. Introduction

1.1 Project background and motivation

This dissertation was developed within the scope of the Integrated Master in Mechanical Engineering, specialization in Automation, at the Faculty of Engineering of the University of Porto. This project began in October 2021 and lasted until the end of March 2022. It was carried out at Institute of Science and Innovation in Mechanical and Industrial Engineering (INEGI) where it was proposed to develop a system capable of reading analog dials values with computer vision.

Analog dials are widely used in industry production, and there are several types such as voltmeters, ammeters, temperature meters, and pressure meters. Many times, it is relied on personnel to record a specific state, by directly reading the values on the gauge. As with any manual operation this is susceptible to errors and has no guaranteed periodicity. In large industrial spaces, maintenance technicians have to travel long distances to monitor the values on the analog dials. Some gauges are also in places that are difficult for the worker to access. Also, when workers change their shift, there is a period when no one is available to record the values on some specific gauges.

The modern industrial production has changed traditional manual inspections to computer-assisted methodologies. In addition, the results of the research and development of new methodologies in industrial automation revolutionized the entire product development strategy. Image analysis applied to computer vision allows readings from analog instrumentation, using image processing algorithms to extract the information contained in the image. The result of image analysis is a measurement that can be converted into a numerical value of the quantity underlying the image. The immediate advantage is that it is possible to display the result on the web, communicate the reading to and from any location, without having to adapt digital analog converters to analog measuring instruments. Thus, through computer vision and appropriate digital image processing, the result can be interpreted for further processing or applied to process decision making. The evolution of technology has allowed the emergence of artificial vision that is more accurate than human vision by reducing ambiguity, making the methods based on computer vision as or more effective than the human vision, automating and reducing errors during the verification process.

After a detailed search on the market, it was found that there are few products available for this purpose, and those that exist are expensive or do not have an open platform. This is not sustainable for a company that wants to equip all the analog dials on its factory and has already subscribed to a Cloud server to store their data. Therefore, this work intends to overcome this same fact, thus making available to the market a low cost and functional product, developed using non-proprietary software that allows intercommunication with any desired Cloud.

1.2 Objective

This work aims to project an analog dial reader that can be installed on analog dials for non-invasive monitoring, this means, there is no need to cut pipes, shut down processes, perform leak checks or run wires. For this, a broad type of requirements needs to be established in this project, although the need to have a low-cost product and a low power consumption device is always kept in mind. This will allow the product to be widely used throughout the industry due to its low cost and being able to run on its own power source for a long period of time.

This work proposes to create a device that connects multiple devices to the Cloud. Such architecture allows interoperability between the various devices that exist today. A rapid growth of technologies provides us with powerful tools and give us the ability to get the information from all over the world. In this project the goal is to create a device that takes a picture of the analog dial, connects to the Internet and allows information sharing over the Cloud. Such architecture allows interoperability between the various devices connected to the network. For this purpose it was necessary to look for the different components capable of providing such requirements.

The proper microcontroller and a camera module capable of capture images with enough quality for this application, must be selected. The data communication method has to be chosen accordingly. The existence of analog dials with quite different diameters will be something to consider, as well as the possibility of being able to see the analog dial manually. This device, once designed to have its own power supply, requires a low power consumption. For this project the image processing will be performed in the Cloud. It is also necessary to choose a proper camera lens. This will be made to allow the construction of an equipment with the smallest possible dimensions. The illumination is also something to consider in this system. Light selection determines the conditions of captured images, which is very important for image processing. The environmental light or the lack of it is also a concern, in which a constant light needs to be established to obtain the best image processing calculation.

A mechanical structure will be also developed to secure all components (battery, illumination, microcontroller, and others) and ensure the axial position of the camera relative to the analog dial. As said before, it is important to create a small and compact system which can be attach to a wide range of cylindrical gauges. Some gauges are confined in very strict spaces, or with other gauges next to them, so it is important to have an overall small size support.

Finally, after being capable of acquiring acceptable images from the gauges, it will be developed an algorithm capable of processing the image of the analog dial and extract the value represented on it. The programming language used to develop the application to support this task, with data processing and image acquisition capabilities, is the Python language [1]. The choice for this language is due to the fact that it is free, easy to implement, with abundant dedicated information on the Internet, and capable of producing a prototype completely independent of paid licenses, which gives more value to this work.

1.3 Thesis structure

This document is divided into 5 chapters. This 1st chapter includes an introduction to the problem that affects the overall conceptual design, and some requirements needed to ensure a good quality image and a proper processing of the digital image.

Chapter 2 briefly describes methodologies followed in the literature of the vision field. Here are referenced methodologies followed for hardware choice, for appropriate illumination type selection, as well as some techniques used for image processing.

In chapter 3, the choice of the different components used such as the microcontroller, lenses and lights are presented. It also presents the experimental work done to define the materials and distances to use for the construction of the prototype.

The algorithm developed for the detection of the value represented in an analog dial is presented on chapter 4.

Chapter 5 presents a general discussion of the results obtained and its limitations. Follows chapter 6 where we present the conclusion of this work with a perspective of future development.

Attached are the code to perform the camera calibration, the code developed for the detection of the analog dial, data sheets of materials used, the experimental results of the code, as well as the technical drawings of the designed parts and the respective exploded view.

2. State of Art

The unprecedented evolution and cost reductions of electronics in recent times has had repercussions on the advancement of computer and information technology and consequently on the increase in the calculating capacity of software and hardware, allowing the image processing-oriented branch of science to give rise to computer vision. In this work it is planned to develop a product, using non-proprietary software. For this it will be analyzed several works from the literature with special focus on microcontrollers, camera lens, light techniques, wireless communication, and image processing algorithms.

2.1 Available technology and hardware evolution

As mentioned above, the acquisition of analog values by computer vision has been used for years in areas where access is difficult. Computer vision is a tool that allows to analyze the content of the image and extract, for this desired case, the value represented on the analog dial by its needle. After an image acquisition and interpretation of information that they contain, it is possible to translated this information for the purpose for which it is intended, for example, in a measure.

At the hardware level many evolutions have been seen over the years, where it can be found a wide variety of devices with different features in the market. Microcontrollers like Arduino, or more recently introduced, ESP-8266 and ESP-32 became available as low-cost alternatives. Recent advancements in microcontroller technology have enhanced the capabilities and overall functioning providing additional features such as live streaming over distance, augmenting mobile applications, storage capabilities, cost effectiveness and easier implementation.

Microcontroller-based board is a single integrated circuit that houses several components such as processing unit, memory, programmable I/O peripherals. They are often cheap with limited processing power, memory, and connectivity. Several surveys [2]–[6] demonstrated the features of IoT devices found in the market, accompanied by their own specifications, such as processing and memory capabilities, communication and connectivity interfaces, operating system (OS) support, security, power specifications, size and cost.

2.2 Internet of Things

In a generalized approach, the concept of IoT refers to the idea of connecting any device to the Internet [7]. This includes everything from cell phones, home appliances, wearable devices, etc. Despite that, today it is also common to associate IoT with devices equipped with sensors and actuators of different types, through which a given system has the ability of collecting and sharing information online.

However, the actual term IoT was first used by Kevin Ashton in 1999 during his work at Procter&Gamble [8]. Ashton, who was working on supply chain optimization, wanted to draw management's attention to a new and exciting technology called RFID. Because the Internet was the new trend in 1999 and because it somehow made sense, he called his presentation the "Internet of Things".

From the above information it can be seen that the notion of IoT has evolved over time, it differs across different practitioners and researchers and has been influenced by the availability of new technologies and social trends. According to Floris and Atzori [9], three primary generations of IoT were pointed out, which are associated with different types of components:

- **RFID tags**, commonly used for monitoring processes, with logistical tracking applications.
- **Sensors and Actuators**, which allows to send/receive information between digital and the physical worlds.
- **Virtual objects**, it's the most recent IoT generation, developed based on the creation of a virtual object to every physical object involved, mediated through multimedia means.

According to GSMA Intelligence [10], it is forecasted that there will be about 25 billion IoT devices by 2025 and this implies the development of several new versions of IoT devices, platforms, and technologies. IoT devices are typically small, combining microcontrollers and microprocessors with other components, such as memory, I/O, peripherals, and wireless connectivity chips in a pre-built, ready to program package. IoT devices can also be differentiated based on microprocessor-based devices and microcontroller-based devices [5].

2.3 Edge devices

CYPRESS ENVIROSYSTEMS developed a Wireless Gauge Reader with charge-coupled device camera, one microcontroller, two light-emitting diode (LED) illumination, which acts as a flash for image capture, a wireless transceiver module with antenna powered by two 3,3V batteries, with an announced battery life of three to five years at a 15-minute sampling rate [11]. In this system the calibration procedure usually takes about 5 - 10 minutes and involves setting six parameters - minimum and maximum angles, minimum/maximum needle travel angle, needle rest correction, and gauge tilt angle -, for each gauge, plus selecting a series of circles to detect the needle. This product

processes the image in-place and displays it in a liquid crystal display (LCD) on the front of the device. The product mentioned above, applied to a manometer for testing, is shown in Figure 2.1.



Figure 2.1 - CYPRESS ENVIROSSYSTEMS Gauge Reader applied in a testing environment [11].

Honeywell also developed a related product with the same features and same dimensions [12]. Both products do not specify the algorithm and the microcontroller used.

Dietz et al., describes the design, construction, and experimental evaluation of a low-cost large-format digital camera [13]. Here a wireless IoT device, fully remote controllable via Bluetooth and Wi-Fi was implemented. This camera had the purpose to serve as a testbed for novel ways to improve capture quality for scenes that are not completely static during the scan interval.

According to Pertab Rai et al., IoT enabled devices were implemented that provides end user with cost effective and portable smart surveillance system [14]. The proposed implementation acquires low resolution video using detachable Arducam-Mini-2 Mega Pixels (MP) camera module which is interfaced with Arducam ESP-32 UNO board. The acquired video is transmitted using integrated Wi-Fi of ESP-32 and video is shown on a display module.

Emil Gatia, et al., tried to summarize the most important issues and best practices learned during the integration of low-cost and energy efficient microcontroller device used for industrial wireless sensor network (IWSN) [15]. ESP-32 was chosen, in that paper, as energy efficient and still powerful microcontroller that was proven by many method [16] [17] and products [18] [19] implemented so far. The flexibility and energy efficiency of ESP-32 System-on-Chip (SoC) is supported by many different operating modes.

2.4 Camera lens

The correct imaging lens selection has a great importance in the quality of the generated image. For this reason some parameters should be understood. Knowing these parameters will help in selecting the ideal lens and camera combination. These variables are represented in the detail in Figure 2.2, allowing a better understanding of them.

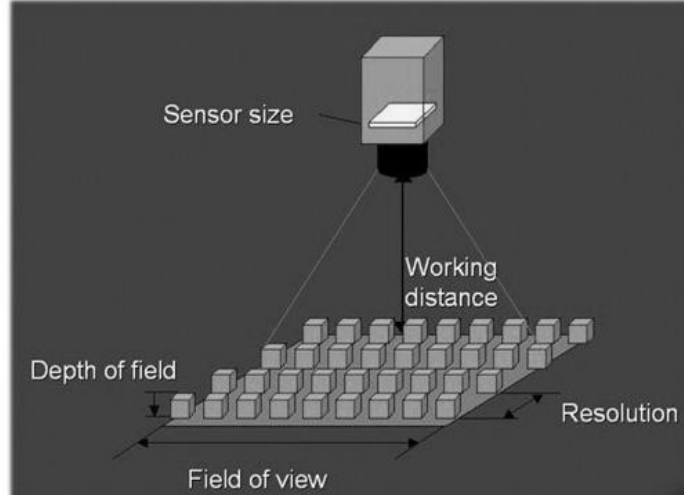


Figure 2.2 - Schematic representation of the fundamental characteristics for image acquisition [20].

- **Field of View (FOV):** The viewable area of the object under inspection. This is the portion of the object that fills the camera's sensor.
- **Working Distance (WD):** The distance from the front of the lens to the object under inspection.
- **Resolution:** The minimum feature size of the object that can be distinguished by the imaging system.
- **Depth of Field (DOF):** The maximum object depth that can be maintained entirely in acceptable focus. DOF is also the amount of object movement (in and out of best focus) allowable while maintaining focus.

As explained, the equipment has to be small so it is important to have a minimum working distance as possible and yet be able to capture the full gauge image. Assuming that the Contrast Transfer Function (CTF) or Point Spread Function (PSF) of camera+lens does not blur the image, the pixel resolution needed to make an accurate measurement of the image can be achieved assuming a minimum of two pixels per smallest feature that is needed to detect. Knowing this, the minimum required sensor resolution can be determined using the Equation 2.1 [20]:

$$\text{Image Resolution} > 2 \times \left(\frac{FOV}{Resolution} \right) \quad (2.1)$$

With the Equation 2.2 is possible to know an approximation ratio of focal length to working distance [20]. Such relationship is illustrated in Figure 2.3.

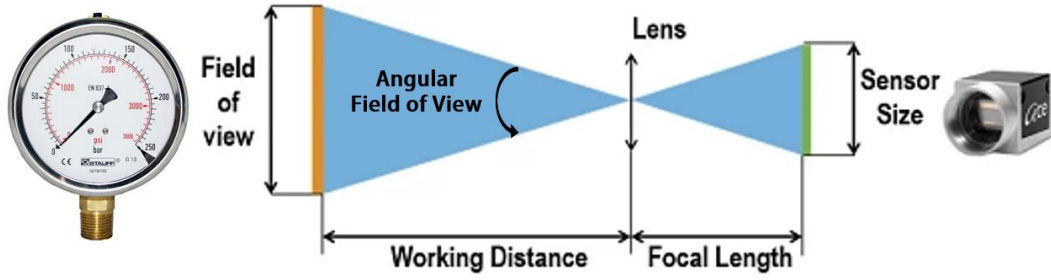


Figure 2.3 - Camera lens FOV, WD, AFOV and focal length parameters [20].

$$Focal\ Length \times FOV = Sensor\ Size \times WD \quad (2.2)$$

Generally, lenses have fixed focal lengths, by that it is required to test different lens in order to find a configuration that allows a minimum working distance as possible. After analyzing Equation 2.2 it can be noted that a lens with a longer focal length will require a proportionally longer working distance. However, the working distance is also a function of the angular field of view (AFOV) of the lens. Lenses with a larger AFOV allows shorter working distances, for the same camera used, as represented in the following Equation 2.3 [20].

$$AFOV = 2 \times \tan^{-1} \left(\frac{FOV}{2 \times WD} \right) \quad (2.3)$$

Using lenses with higher angular fields of view (AFOV) cause image distortion, but it is possible to get around this by performing an image calibration, relative to the camera used. Wide-angle lenses are widely utilized in various applications, such as computer vision, sky observation, military scenario, medical imaging system, and so on. However, wide-angle lens distortion bends straight lines into circular arcs. This would reduce the accuracy of quantitative measurements in determining the geometric position and size of objects in the image, and even cause misjudgment in the process of image recognition. Therefore, it is especially crucial to implement distortion correction.

2.4.1 Lens distortion calibration

Camera calibration can be defined as the technique to estimating the characteristics of the camera and lens set. It means that all the camera's information is known, like parameters, which are needed to determine an accurate relationship between a 3D point in the real world and its corresponding 2D projection in the image acquired by that

calibrated camera. For this, one needs to know the intrinsic and extrinsic parameters. World points are transformed to camera coordinates using the extrinsic parameters. The camera coordinates are mapped into the image plane using the intrinsic parameters.

2.4.1.1 Extrinsic or External Parameters

The external parameters describes the orientation and location of the camera. This refers to the rotation and translation of the camera with respect to some world coordinate system. The extrinsic parameters consist of a rotation, R , and a translation, t . The origin of the camera's coordinate system is at its optical center and its x-axis and y-axis define the image plane. These parameters are demonstrated in Figure 2.4.

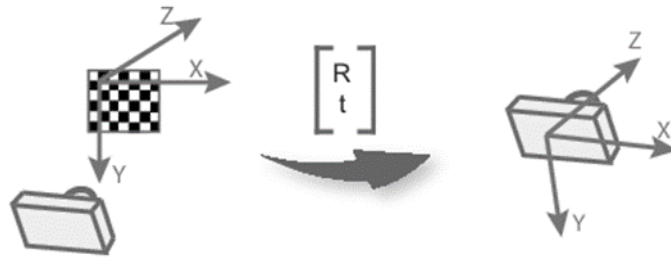


Figure 2.4 - Lens distortion external parameters coordinates illustration [21].

2.4.1.2 Intrinsic or Internal Parameters

It allows the mapping between pixel coordinates and camera coordinates in the image frame. To do this it is necessary to set parameters for the lens in use. The intrinsic parameters include the camera matrix and the distortions parameters.

The camera intrinsic matrix, K , include parameters like the focal length (f_x and f_y) and the optical center, also known as the principal point (c_x , c_y), expressed in pixels coordinates. The camera intrinsic matrix, K , is defined as shown in Equation 2.4 [21]:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (2.4)$$

Radial and tangential distortion are two of the most relevant types of distortions caused by the lens of a camera. Figure 2.5 illustrates different types of radial distortion.

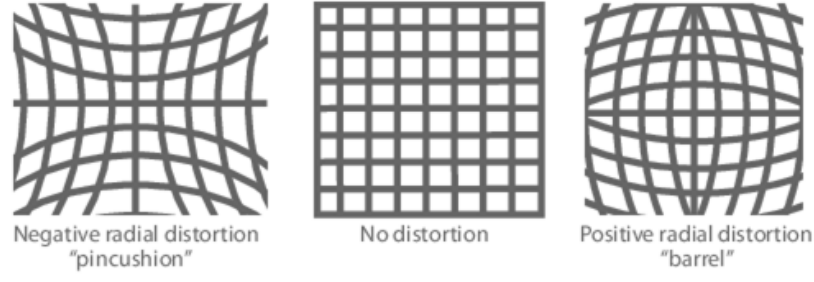


Figure 2.5 - Radial types of distortions [21].

Radial distortion, in particular that which causes the "barrel" effect, is the most frequent when using lenses with a high angle field of view. Using the function provided in the OpenCV [22] library allows the determination of the radial distortion of the image and its subsequent correction. Radial distortion occurs when light rays bend close to the edges of a lens than they do at its optical center. The smaller the lens, the greater the distortion. For the determination of this distortion the Equation 2.5 and Equation 2.6 are used [21].

$$x_{distorted} = x \times (1 + k_1 \times r^2 + k_2 \times r^4 + k_3 \times r^6) \quad (2.5)$$

$$y_{distorted} = y \times (1 + k_1 \times r^2 + k_2 \times r^4 + k_3 \times r^6) \quad (2.6)$$

where (k_1, k_2, k_3) are the radial distortions parameters to be determined by the calibration process.

Tangential distortion can also be seen in imaging distortion, this distortion happens when the lens is not completely parallel to the camera sensor. For the determination of tangential distortion, Equation 2.7 and Equation 2.8, are used [21].

$$x_{distorted} = x + [2 \times p_1 \times x \times y + p_2(r^2 + 2 \times x^2)] \quad (2.7)$$

$$y_{distorted} = y + [p_1(r^2 + 2 \times y^2) + 2 \times p_2 \times x \times y] \quad (2.8)$$

where (p_1, p_2) are the radial distortions parameters to be determined by the calibration process.

2.5 Lighting techniques

In computer vision the lighting environment is very important to create a quality, robust, and timely vision inspection. In addition, to an understanding of illumination types and techniques it is necessary to take into account different parameters such as geometry, filtering, sensor characteristics, and light color.

2.5.1 Light sources

Light is a photon propagation as a transverse electromagnetic energy wave characterized by its frequency, photon intensity and wavelength [23]. These characteristics depend particularly in the type of light source used. The following lighting sources are now commonly used in machine vision:

- Fluorescent
- Quartz Halogen
- Fiber Optics
- Light Emitting Diode (LED)
- Metal Halide (Mercury)
- Xenon

Strobing Fluorescent, quartz-halogen, and LED are by far the most widely used lighting types in machine vision, particularly for small to medium scale inspection stations, whereas metal halide and xenon are more typically used in large scale applications, or in areas requiring a very bright source [24]. Metal halide, also known as mercury, is often used in microscopy because it has many discrete wavelength peaks. A xenon source is useful for applications requiring a very bright, strobed light. Figure 2.6 highlights the main characteristics of each of the most commonly used light sources, so that a comparison can be made between them.

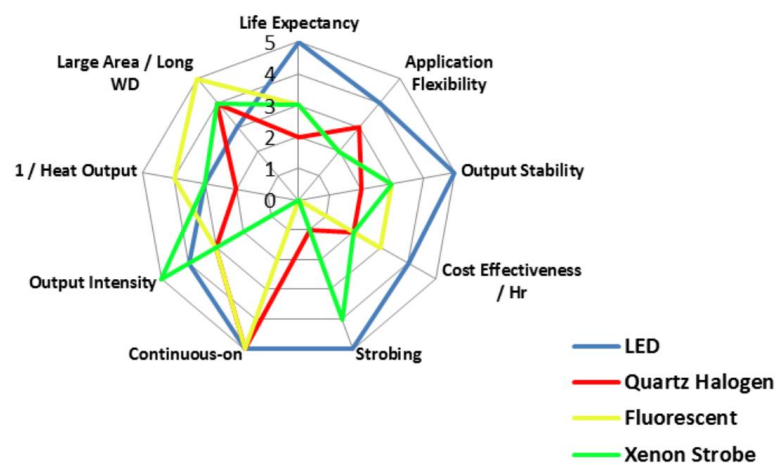


Figure 2.6 - Advantages and disadvantages of different light sources, applied to machine vision [25].

LEDs provide a unique level of performance for high-value large-scale display systems and potential cost saving lighting products because of many key advantages such as high efficiency and reliability, compact size, low power consumption, and even environmental friendliness. A typical LED has a Lambertian distribution [26] that may cause problems in assuring the brightness uniformity. A surface-mount LED chip-set may offer the uniform irradiation in both the near field and the far field by optimizing the arrangement of LEDs [27]–[29].

2.5.2 Light conditionings

Different image contrasts can be reached by using different geometry positioning, light techniques, and light colors. Effective contrast changes via geometry involves moving the sample, light, and/or camera positions until a suitable configuration is found. For example, a co-axial ring light (one typically mounted around the lens, or at least around the camera/lens optical path) may generate hot spot glare on a semi-reflective surface, but by simply moving the light off-axis, the hot spot glare is also moved out of the camera's view. With this strategy it can be divided in two different light illuminations, known as bright-field and dark-field. With a bright-field illumination, the light is reflected into the camera and with dark-field the light is reflected away from the camera, as illustrated in Figure 2.7.

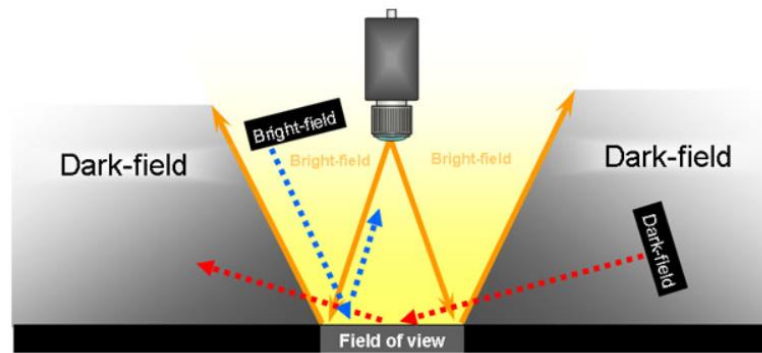


Figure 2.7 - Dark-field vs Bright-field illustration [30].

Contrast changes via the shape of the light projected on the sample is generally achieved using specific lighting techniques. Contrast changes via color lighting are related to differential color absorbance vs. reflectance. If the object has specular reflection properties, then an error may occur in the reconstructed data according to the noise of intensity of acquired images. To inspect reflective, curved, or uneven surfaces diffuse lighting is considered the most appropriate [31], shown in Figure 2.8 a). Traditionally, imaging applications that require diffuse lighting to mitigate point-source reflections rely on standard dome lights [32] with a camera hole. While these lights suit many applications, the flat dome light [33] eliminates the mechanical constraints often encountered when using these lighting methods, making it especially advantageous for compact vision systems and where added flexibility in working distance and degree of light collimation is useful, shown in Figure 2.8 b). Although, when presented with strong specular reflection, the aperture leaved in the light dome for the camera can create some non-uniformity. For these cases coaxial light [34], [35] can be a solution for some cases but given its constructive dimensions it cannot be used in tight spaces, shown in Figure 2.8 c).

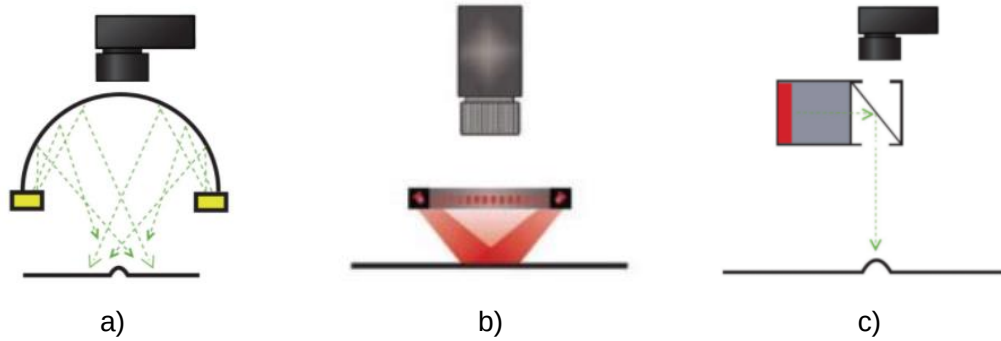


Figure 2.8 - Light techniques, a) Diffuse dome light, b) Flat dome light, c) Coaxial diffuse light [36].

The design of a diffuse reflection lighting system must take in consideration the size of the object to observe, the light source used and the space available for the system. Some techniques are presented in the literature with the use of LED as light source [37], [38]. It must be studied to determine if a diffuse light source is required, and if it can be applied because of limited close-up, top-down access. In some cases when presented with unwanted glare from the vision lighting, polarizing filters [39] can be used, although it must be taken in consideration the price and applicability on the system.

2.6 Communication

Interoperability is a key module in the definition of the IoT network, allowing for an open communication between several devices connect to the Cloud. One of the initial premises of this project is that it would need to have an open communication. For this reason, it was considered that using a software-based communication rather than hardware, with open interfaces, would be the most appropriate way to do the installation in any type of environment. This approach also allows to make communication agnostic to the hardware where it is installed, this means it is possible to have standard communication with devices either by wireless, ethernet, Bluetooth, etc. Cloud Computing is therefore taken into consideration.

A wired network employs wires to link devices to the Internet or another network, on the other hand, a wireless network uses radio-frequency waves rather than wires. Wireless communications are a fast-growing technology that enables people to access networks and services without cables. Bluetooth and Wi-Fi (*IEEE 802.11*) are two communication protocol standards that define a physical layer and a MAC layer for wireless communications within a short range (from a few meters up to 100 m) with low power consumption (from less than 1 mW up to 100 mW) [40], as shown on Table 2.1.

Table 2.1 - Internet vs Bluetooth comparison.

Criteria	Bluetooth	Wi-Fi
Bandwidth	800 [Kbps]	11 [Mbps]
Range	2 [m]	5 - 140 [m]
Latency	200 [ms]	150 [ms]

As seen from Table 2.1, Wi-Fi has better performance in several criteria such as speed, communication distance, and delays. Wi-Fi is well suited for operating full-scale networks because it enables a faster connection, wider range from the base station, and better wireless security than Bluetooth.

2.6.1 IoT communication protocols

As mentioned before, there are available in the market a wide variety of IoT devices, as a consequence of such diversity the communication of devices is a prominent challenge. Due to its remote nature and the need for wireless networking, IoT systems must be able to handle potentially intermittent and low bandwidth connections [41]. Additionally, IoT allows connected devices to turn into smart systems, but such devices are usually heterogeneous and they need a common interaction mechanism.

Communication protocols are of great importance in heterogeneous systems, which define an accepted framework for device interaction [42]. Recently, there have been proposals to put sensor data onto Clouds, be they private or public. Cloud data acquisition in an IoT Cloud architecture can be achieved using a Hypertext Transfer Protocol [43] (HTTP) server, a Message Queuing Telemetry Transport [44] (MQTT) server, or a Constrained Application Protocol [45] (CoAP) server. Just as web clients access HTTP servers through a push-pull Application Programming Interface (APIs), IoT devices can use a subscribe-publish API to communicate with MQTT servers residing in the cloud. A comparison between these communications is presented in Table 2.2.

Table 2.2 - IoT communication protocols comparison.

	HTTP	CoAP	MQTT
Transport Layer	TCP	UDP	TCP
Type	push-pull	push-pull subscribe-publish	subscribe-publish
Mechanism	one-to-one	one-to-one	one-to-one
Methodology	document-oriented	document-oriented	message-oriented
Level of quality	one	two	three
Security	SSL, TLS	DTLS	SSL, TLS

2.6.2 Cloud services

Cloud technologies are increasingly used by businesses because they are fast and their scope is growing. Companies tend to increasingly deploy data and studies into the Cloud, which enables data sharing for production systems. According to Microsoft Azure [46], Cloud computing is the provisioning of computing services, including servers, storage, databases, networking, software, analytics and intelligence, over the Internet - "Cloud" - to more rapidly enable innovation, flexible resources and sizing savings. According to CompuWorks, the benefits are numerous [47]:

1. **cost reduction and optimization of IT investment** - Cloud computing eliminates the costs associated with the purchase and maintenance of hardware and software, as well as the installation and running of local data centers;
2. **improved productivity and flexibility** - resources associated with the company's activity are stored on external servers, even if there is a problem in the physical infrastructure there is no risk of losing information. Additionally, employees can access the company's information and systems from any location and at any time, facilitating work and paving the way for remote work;
3. **increased security** - the company's information becomes centralized. The backups are automatic and redundant, ensuring more security in information storage;
4. **system scalability** - the Cloud system is easily scalable if the company needs more resources, storage space or processing speed.

After researching the various Cloud services available for this purpose is present on Table 2.3 a comparison of the main differences between the main Clouds used today.

Table 2.3 - Cloud services comparison.

Cloud Service	Authentication	Communication Protocols	Security	Data Analytics
Google IoT Core [48]	public/private key pairs and JSON Web Tokens	HTTP and MQTT	TLS	Google Big, Cloud Dataflow, BigQuery, and more
AWS [49]	public/private key pairs and Web Tokens	HTTP, MQTT and WebSockets	TLS	Basic data management (filtering and transforming)
Microsoft Azure [50]	Public/private key and Web Tokens	HTTP, MQTT, AMQP and WebSockets	TLS	Azure Synapse Analytics, Blob storage, Analysis Services and more

2.7 Image processing

Image processing is a way to convert an image to a digital form and perform certain functions on it [51], in order to get an enhanced image or extract other useful information from it. This takes as input an image, such as a frame or video image, and the output can be an image or features associated with that image. There are many types of image processing, such as:

- **Recognition** - distinguish or detect objects in the image;
- **Sharpening and restoration** - create an enhanced image from the original image;
- **Pattern recognition** - measure various patterns around the objects in the image;
- **Morphological** - set of processing operations for morphing images based on their shapes;
- **Segmentation** - partitioning an image into its constitutive parts or objects;
- **Character recognition** – conversion of an image to a computer-readable text.

Data processing and image acquisition, through the Python language, allows the use of specialized libraries for this purpose that facilitate the implementation of complex algorithms, developed over the years by Internet users, making such documentation available to the entire community. Among the existing ones, the SciPy [52], Python Imaging Library (PIL) [53], and Open Source Computer Vision (OpenCV) [54] libraries can be highlighted due to their great importance in assisting in image processing operations.

Computer vision is a tool that allows the analysis of the image content of a mechanical measuring device, without physical contact with that device. Essential measurement information can be extracted, which can range from measuring the contour area of an object in the image, to, for example, automatically sensing whether certain

manufactured parts match the tolerance stipulated by the manufacturer in the quality control of the process, or, as in this case, finding the gauge dial and calculating its position relative to the scale marks.

From the articles for reading analog displays presented here, different techniques were used to solve this problem. It is important to keep in mind that the requirements of each application influence the method used to obtain better results. The quality of the image to be processed is a factor to be considered. In 2018, Kucherukin, presented an article on processing low quality images, the detection of the pointer is accomplished by extracting the skeletal of the dial, and determined the rotation angle to compute the reading [55]. In conditions of varying illumination and camera angles some specific approach is required, for example, Zheng, et al., found a useful way to enhance color images which suffer from lacking brightness by using the Multi-Scale Retinex with color restoration. To correct the images angle differences a perspective transformation was applied, and then, the pointer was detected using a Hough transform [56].

Paulo Sousa, implemented a system for pointers detection [57]. Here a comparison was made between the skeleton extraction method, Hough transform for lines and template-matching method, achieving better results by applying the Hough transform. For this Paulo used Otsu methods to separate the Region of Interest (ROI) from the background. next used a Sobel filter applied to the binary image giving better contours of the objects, essential for the proper functioning of the Hough transform.

In these works [55]–[57], only the detection of the needle was addressed, and user input was required to finalize the computation of the value represented by the needle. Chi, et al., used an improved central projection method to determine the circular scale region and detected the scale marks and then used the border detection and Hough transform to obtain the pointer direction, the reading of the gauge was obtained by comparing the location of the pointer with the scale marks [58].

One of the common methods for detecting and localizing the pointer is the Hough transform method as described in [55]–[61]. It can be used to detect the straight line that represents the pointer on the analog dial image. The least squares fitting method has also been used in a number of pointer location works by some researchers in [60], [62]. It finds the symmetric central line of the pointer by fitting a regression line through the cluster of pixels representing the pointer. The Convolutional Neural Network (CNN) on object detection is also used for this purpose [63]–[65]. Also common in image processing are optical character recognition (OCR) engines, which automatically detects characters found in an image. There are open-source codes available, tested in published articles such as the one by Hegghammer, et al., where a comparison is made between Tesseract [66], Amazon Textract and Google Document AI.

3. Prototype

This section describes the selection of the microcontroller, lens, lights, and communication to be used. A detailed description of the experimental procedure used to define the measurements needed to build the final component is also given on this chapter. Figure 3.1 shows the block diagram of the proposed wireless analog dial reader system. The input of the device is the captured images from the industrial floor gauges. The camera captures images that are transmitted to a Cloud service, instead of performing tasks locally, on the edge device. Third party services will be used to provide a Cloud-based management, analytics, and storage services. The proposed system will perform dial detection remotely, on the Cloud services. This section lists and explains each component used.

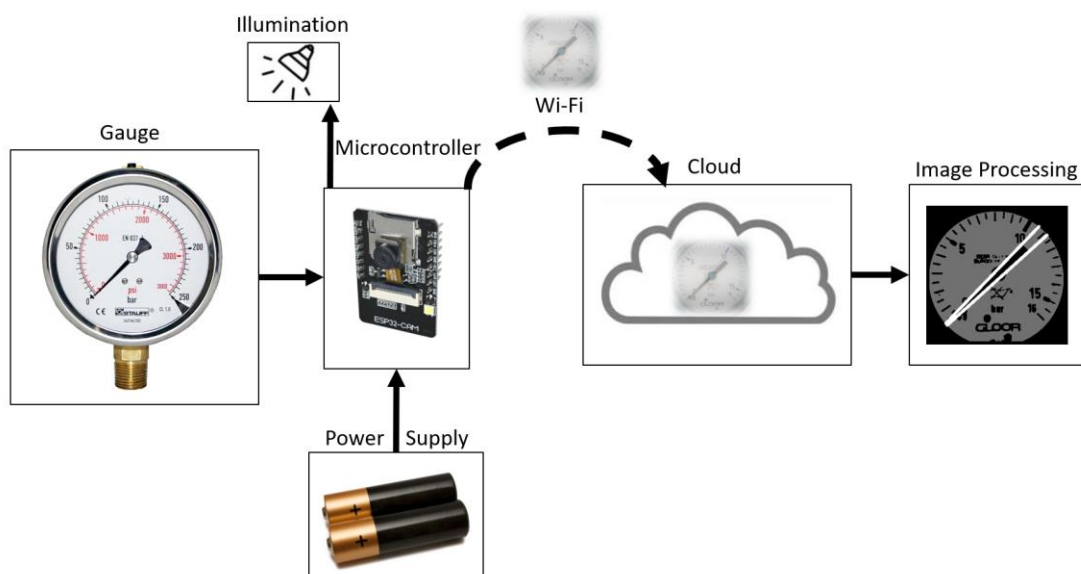


Figure 3.1 - Proposed architecture of the prototype under development.

3.1 Microcontroller

Given the designed architecture, three different paths can be taken that are important for the proper choice of a microcontroller:

- a) capture the image, local image process and send the measured value to the Cloud;
- b) capture the image, send it to the Cloud and then process the image and extract the measured value.
- c) capture the image, local image process, with the possibility of sending the image in case of an alarm or processing error.

In this case it was chosen not to process the image in the microcontroller. So, the image is captured and sent to the Cloud, via Wi-Fi, being then processed and interpreted. This is done for four reasons:

1. Limitations on the part of the microcontroller in processing capacity and the image processing on it would lead to higher power consumptions;
2. Image processing is done in the Cloud, so it allows any change to be made to the code when desired, and such a change would be applied to any analog dial reader that is being used;
3. Sending the images to the Cloud allows the creation of a large database for further implementation of machine learning algorithms;
4. Comparison of the values calculated with the received image and check whether the image processing performed correctly.

It is then necessary to choose a microcontroller with a camera capable of a quality image and sending it to the Cloud, via Wi-Fi. After analyzing different documents and surveys available in the literature, it is possible to list some devices that are able to satisfy the requirements needed for this work. These devices are listed below:

- Samsung ARTIK 055s with 320MHz clock speed, 128KB Random Access Memory (RAM), 8MB on-board storage, Wi-Fi, 3.3V for 9,73 dollars [5], but it does not allow a camera sensor;
- Omion Omega2 with 580MHz clock speed, 64MB RAM, 16MB on-board storage, Wi-Fi, 3,3V for 5 dollars [5], requiring a camera module expansion;
- ESP8266 with 80MHz clock speed, 160KB RAM, 512KB to 16MB on board storage, Wi-Fi, 3,3V for 7 dollars [5], requiring a camera module expansion;
- ESP-32 with 240MHz clock speed, 520KB RAM, Wi-Fi, 3,3V for 11 dollars [3], Espressif [67] produced a new version with an ESP-32-S SoC with a 2MP camera module on-board, the ESP-32-CAM [68].

The ESP-32, shown in Figure 3.2, was chosen by the fact that it already has a built-in camera module, Wi-Fi connection, and I/O pins for future connection with buttons or sensors.



Figure 3.2 - Microcontroller ESP-32 CAM [69].

The ESP-32 camera module is used in this project to capture an image from the gauge. It is a small-sized module that can operate independently as a minimum system with a footprint of only 27mm x 40.5mm x 4.5mm. It consists of two parts which are the main board that is an ESP-32 module mounted on a larger board and a separate camera module. The module is a programmable microcontroller with built-in Wi-Fi and Bluetooth, with an additional 4MB of external RAM.

The board is equipped with a memory card slot and a camera connector that can be connected to an OV2640 camera module. However, the memory card is not used in this project development. In this work, once a new image is captured, the previous captured image will be deleted. Table 3.1 shows ESP-32 camera specifications.

Table 3.1 - ESP-32 CAM Specifications.

Module model	ESP-32 CAM
Size [mm]	27 x 40.5 x 4.5
SPI flash	32 Mbit
RAM	520KB SRAM + 4M PSRAM
Bluetooth	4.2 BR/EDR and BLE
Wi-Fi	802.11 b/g/n/e/i
I/O port	9

The use of IoT cameras has also increased due to the high demand for image data. However, the image sensor is unsuitable as an energy-constrained edge device for IoT due to its high-power consumption. Therefore, periodic on-off scheduling of IoT cameras is a promising approach since video recording using image sensors is energy intensive.

ESP-32 is capable of light sleep and deep sleep power saving modes [70]. In light sleep mode, digital peripherals, most of the RAM, and Central Processing Units (CPUs) are clock-gated, and supply current is reduced. Upon exit from light sleep, peripherals and CPUs resume operation, their internal state is preserved. In deep sleep mode, CPUs, most of the RAM, and some digital peripherals are powered off. Wakeup from deep and light sleep modes can be done using several sources, using a timer, waking up after a specified time, or by an external input signal, through an interruption mechanism.

The ESP-32 usually requires about 75mA in normal operation and about 240mA while transmitting data over Wi-Fi. When in deep sleep the chip is expected to require around 0.15mA to 10 μ A [15], [71]. There is also the possibility to use sensor measurements to wakes up the main system.

3.2 Camera Lens

For this analog dial reader, it is necessary to detect features in the captured image equal or greater to 0.5 mm, which is the thickness of the scale numbers from the smallest manometer used on this project. It is also known the maximum FOV, 80 mm. It was decided to use this value since it includes the vast majority of the manometers present in the industry. Knowing this it is now possible to solve Equation 2.1, obtaining a minimum image resolution needed of 320 x 320 pixels. That means that a camera with a resolution of 640 x 480 pixels will work. It is now safe to say that the camera module OV2640, included on the ESP-32, is of sufficient quality because it can deliver images of resolutions up to 1600 x 1200 pixels. It is now determined the sensor size of this camera. Sensor size refers to the physical size of the sensor and is typically not noted on specification sheets. The best way to determine sensor size is to look at the pixel size on the sensor and multiply by the resolution. According to the camera user manual [72], the OV2640 module has a pixel size of 2.2 x 2.2 μ m and a resolution of 1600 x 1200 pixels. The size of the sensor can now be determined by the following calculation:

$$Height = (2.2 \times 1200) \times 10^{-3} = 2.64 \text{ mm} \quad (3.1)$$

$$Width = (2.2 \times 1600) \times 10^{-3} = 3.52 \text{ mm} \quad (3.2)$$

Lenses are manufactured with a limited number of standard focal lengths, the camera module OV2640 has a standard lens with a focal length of 4.8 mm. The minimum working distance can be determined for the OV2640 module by solving Equation 2.2.

$$4.8 \times 80 = 2.64 \times WD \quad (3.3)$$

$$WD = 145.5 \text{ mm} \quad (3.4)$$

This value is clearly high, and therefore it is necessary to test lenses with different AFOV and focal lengths. The use of lenses with a wider aperture range allows the inclusion of the entire object to be viewed from a shorter distance. However, lenses that

are not appropriate may present some distortion if used at distances shorter than those for which they are intended.

In this phase two camera modules were tested with lenses with 66.5° and 160° , as shown in Figure 3.3 a) and b). Also, two lenses with an AFOV of 180° and 235° were tested by placing them over the OV2640 camera module, with an AFOV of 66.5° , shown on Figure 3.3 c) and d). This last configuration requires special attention since it is being used one lens overlapping the other. This leads to some errors, given the difficulty in ensuring the correct alignment of the two lenses, as well as the fact that the focal length of the lens is being increased.



Figure 3.3 - Lenses used in the experimental tests, a) 66.5° lens, b) 160° lens, c) 180° lens, d) 235° lens.

Based on the Equation 2.2 and Equation 2.3, it is now possible to calculate the approximated values of the WD for each lens. These calculations are based on approximations and should be used for reference only. An experimental analysis follows where the WD for each lens is determined, for a FOV of 80 mm, WD illustrated in Figure 3.4 a). For this, the following structure was designed and dimensioned to allow the determination of this distance, shown in Figure 3.4 b). This piece allows the adjustment of the distance from the camera lens to the analog dial with increments of 5 mm.

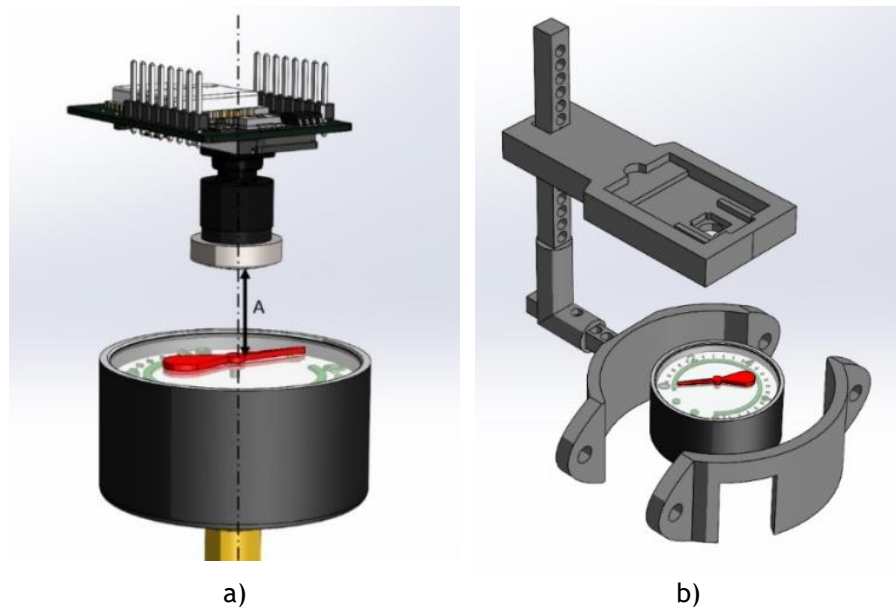


Figure 3.4 - Rehearsal performed lens choice, a) Working distance illustration, b) Support designed for testing.

After performing these tests, the following results were obtained for each lens. (lens identification based on the Figure 2.4)

- a) 70 mm
- b) 30 mm
- c) 40 mm
- d) 25 mm

It is important to note that lenses (c) and (d), given their dimensions, require a greater distance between the microcontroller and the gauge, meaning that lens (b) and (d), although they have identical WD, with lens (d) the microcontroller will have to be further away from the gauge than with lens (b), which implies an increase in the dimensions of the structure to be developed. In order to illustrate the differences observed with the different lenses, a picture was taken with all four lenses at a WD of 30 mm, as shown in Figure 3.5.

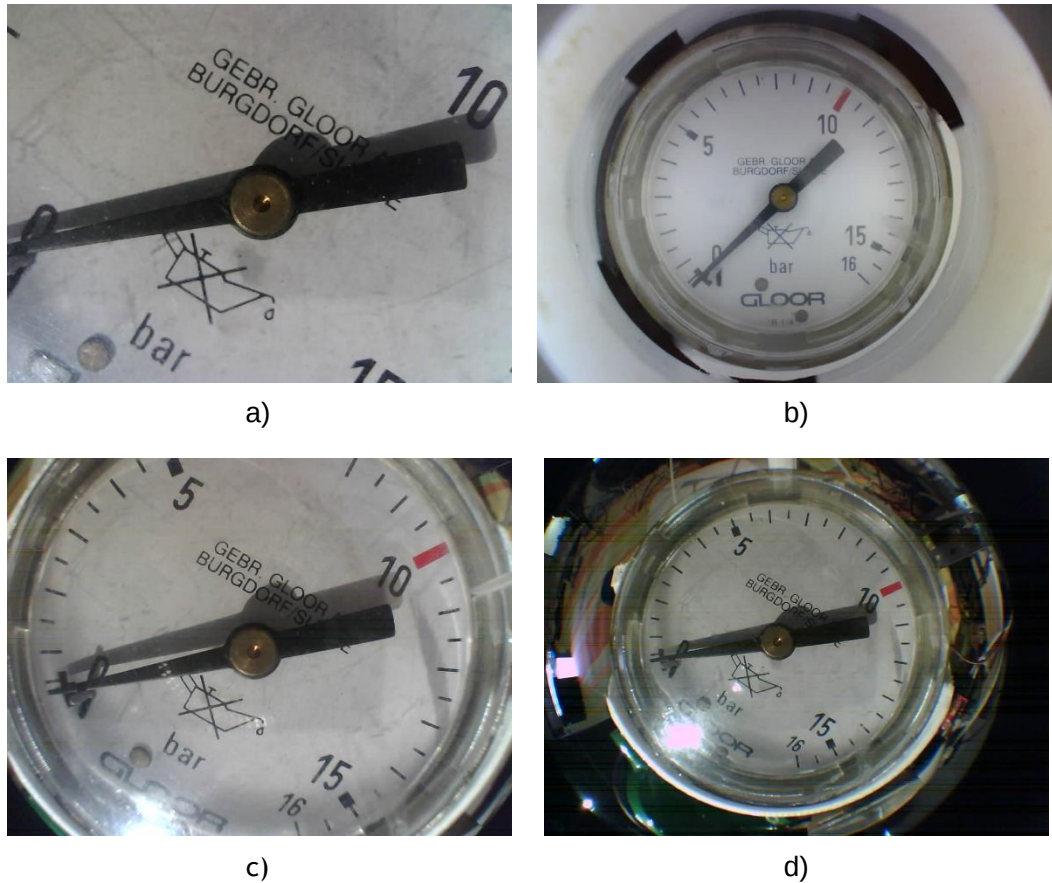


Figure 3.5 - Lenses used in the experimental tests, a) 66.5° lens, a) 160° lens, a) 180° lens, d) 235° lens.

After observing the previous image it can be seen that the image taken with lens (d) presents dark areas on the outer edge of the image. This effect is called vignetting and results from the use of lenses with a smaller format than the sensor format. This means that lens (d) is not suitable for this camera sensor. At this stage it is now safe to say that using the OV2640 camera with a lens with a 160° AFOV is the most correct one to use for this application.

3.3 Illumination

The presence of ambient light input can have a tremendous impact on the quality and consistency of inspections, particularly when using a multi-spectral source, such as white light. There are 3 active methods for dealing with ambient light – high power strobing with short duration pulses, pass filter, and a physical enclosure. For the purpose of producing a product with an acceptable reading and consistence quality it is opted for constructing a prototype that encloses the analog dial. Based on chapter 2.5.1 analysis LED is chosen as the light source.

At this point, and having defined the working distance, it is now necessary to test different illumination techniques, with the aim of obtaining an image without the existence of glare or reflections of light, thus causing loss of information. The first test done was using a single LED. In this phase the LED was moved, in two directions, as illustrated in Figure 3.6, in order to verify if it would be possible to remove the glare captured by the camera, caused by light reflecting on the gauge glass.

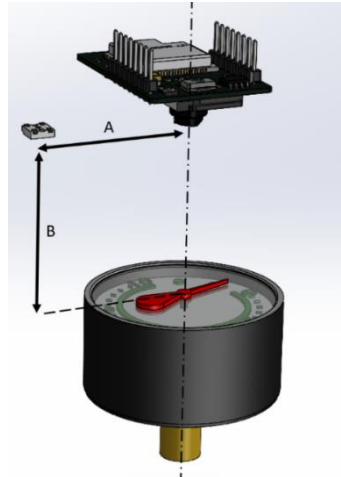


Figure 3.6 - Rehearsal performed for light choice.

Different tests were performed with this configuration by moving the LED to different positions, using techniques such as those referenced on chapter 2, bright field and dark field illumination. After testing different distances to A, horizontal distance from LED to gauge, and B, vertical distance from LED to gauge, the result images are shown in Figure 3.7.



Figure 3.7 - Light technics testing, a) bright field, b) dark field.

After the tests performed it was found that it was not possible to completely remove the reflection from the LED. It was found that for a bright field illumination, shown in Figure 3.7 a), information was eliminated from the image that would be important for

further image processing. On the other hand, using the dark field technique, shown in Figure 3.7 b), the image was too dark, and was also considered not acceptable for the purpose of this work. With these results, it was decided to test placing different diffusers over the LED. The function of these diffusers is to scatter the light rays as they pass over their surface. More detailed specifications of these diffusers used can be found at the end of the document in Annex A. Three different diffusers were tested, as shown in Figure 3.8.

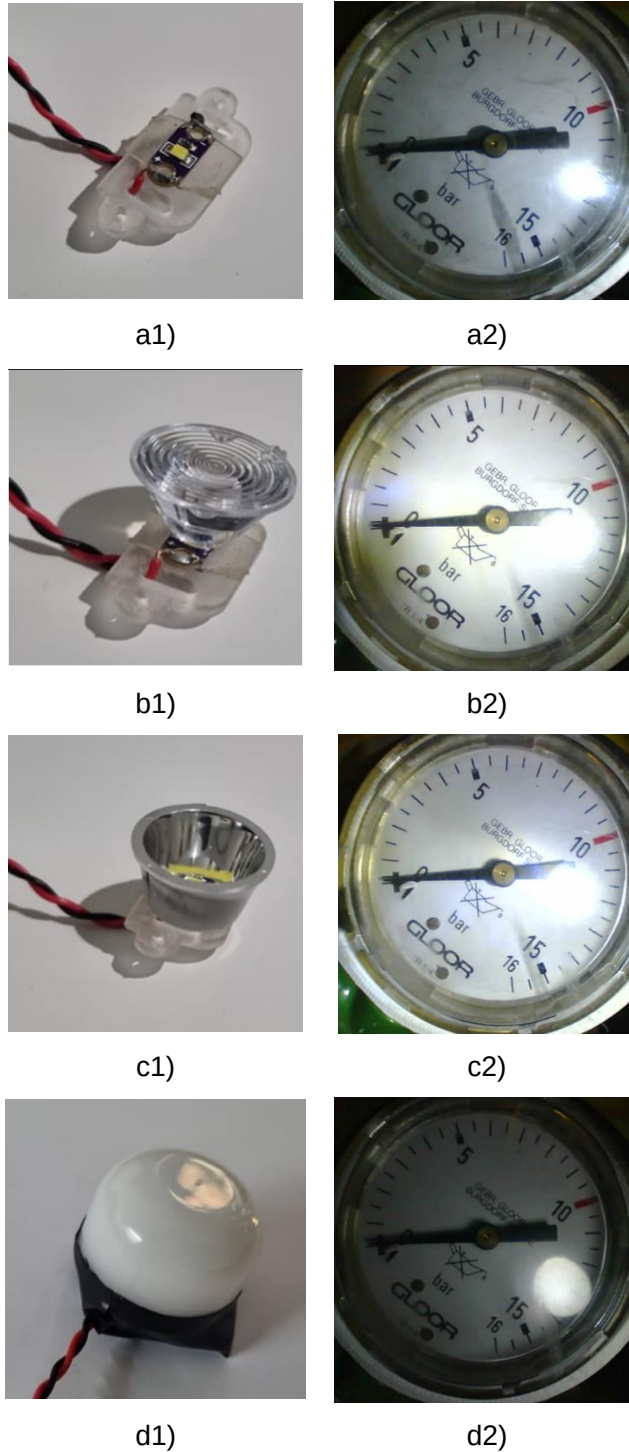


Figure 3.8 - LED diffusers experimental evaluation.

As can be seen from the analysis of the previous images shown so far, in all methods the illumination causes glare to the point of eliminating information from the image that is vital for future image processing. This glare is enough to eliminate the scale value or even eliminate the contours of the needle, making its subsequent detection more difficult. For this reason, new lighting methods need to be tested, such as a dome diffuse light and flat dome illumination.

These techniques are used for cases where a reflecting surface, or a curved surface must be illuminated, such is the case with industrial gauges analog dials. However, it will be necessary to take into consideration in advance the existence of spatial constraints. Such techniques require the use of a specialized structure for their function. It was then studied the dimensioning of such structures and their dimensions. However, in order to impose a dimensional limit, a configuration of manometers very common in the industry was taken into account, as the one shown in Figure 3.9.



Figure 3.9 - Industrial gauge system.

In this situation the distance between the center of the two gauges, represented in the image by "A" is 100 mm. This measurement will be considered as the maximum limit of the diameter for future dimensioning of the required parts.

For the dome diffuse light technique, the following variables (A, B and C) are considered for further testing, as illustrated in Figure 3.10. Due to dimensions containments explained above, dome diffuse light is a space-intensive technique. With this configuration, the bottom aperture opening (B) must be bigger or equal to the gauge diameter to observe, this means that to read an 80 mm diameter gauge, it is needed to have a near or above than 100 mm dome structure which is considered too big for this project.

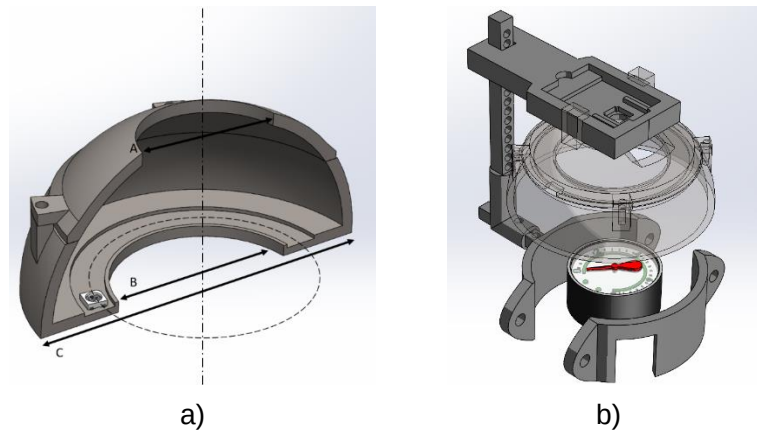


Figure 3.10 - Diffuse dome light technique experiment (A - aperture for lens, B - opening for light, C - dome diameter).

The study continued with the testing of a flat dome lighting technique. For the flat dome, configurations with outer diameters of 80 mm and 100 mm were tested. Diameters smaller than 80 mm were considered too small since a microcontroller, LEDs and a power supply need to be included in the structure. As such, a structure was dimensioned, and the respective results were shown in Figure 3.11.

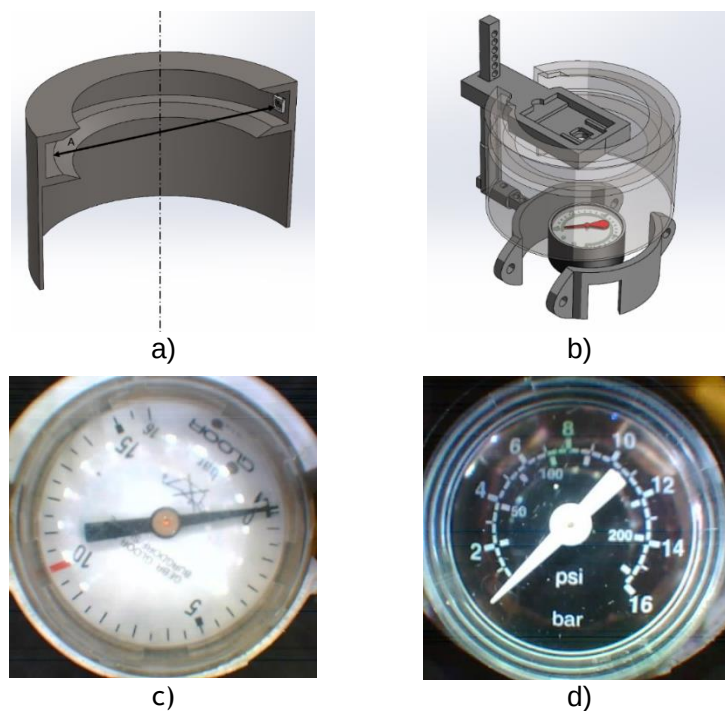


Figure 3.11 - Diffuse flat dome light technique experiment (A - distance from LEDs).

It was found that this technique was also not sufficient to remove the glare resulting from the LED illumination. It was thus decided to test a lighting technique using the previously developed support, placing a filter between the LED and the manometer, as shown in Figure 3.12.

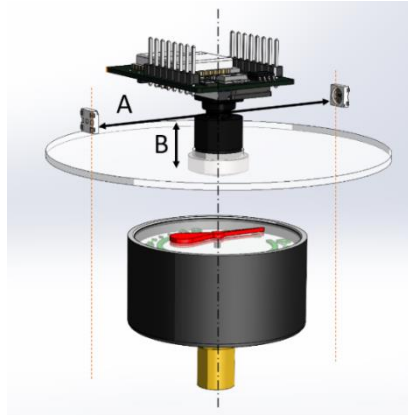


Figure 3.12 - Light filter illustration. (A - distance from LEDs, B - distance from LED to light filter).

Different materials were tested as light filters, in order to determine which one produces the best image for analog dial readding. Of the different materials tested, opalescent acrylic and a white fabric produced images with complete elimination of image glare, as presented in Figure 3.13. Configurations where the LED was overlaid with other materials such as cotton, tracing paper and a frosted acrylic were also tested, however the result images were not of such good quality as the two materials mentioned above.



a)



b)



c)



d)

Figure 3.13 - Results from filtering LED light, a) and b) with fabric filter, c) and d) with acrylic filter.

From the result of the images in Figure 3.13, it is concluded that the use of acrylic produces considerably better quality images. It can also be seen that given the properties of the acrylic the light disperses better, illuminating the manometer uniformly. It can be seen that with the use of fabric as a filter there is a greater difference in luminosity in the center of the image, where the lens is, from the rest of the manometer. The use of acrylic as filter also has the advantage of creating a better separation between the camera system and the environment, compared to fabric, thus helping to protect the most critical parts of the device, such as the camera, from adverse conditions like rain or others. That said, it is then opted to use acrylic as a light filter.

It is now necessary to determine the distance from the LED to the acrylic. The LED does not have the ability to disperse the light when restricted to a short space. It is also important to decide the angle of incidence of the LED to the acrylic, i.e. whether the LED will point directly at the acrylic, or whether it will point forward, illuminating the acrylic by reflecting the light on the structure. For this, tests were made with the LED pointed directly at the acrylic to verify the minimum distance for the light to be evenly distributed throughout the acrylic. Results shown in Figure 3.14.

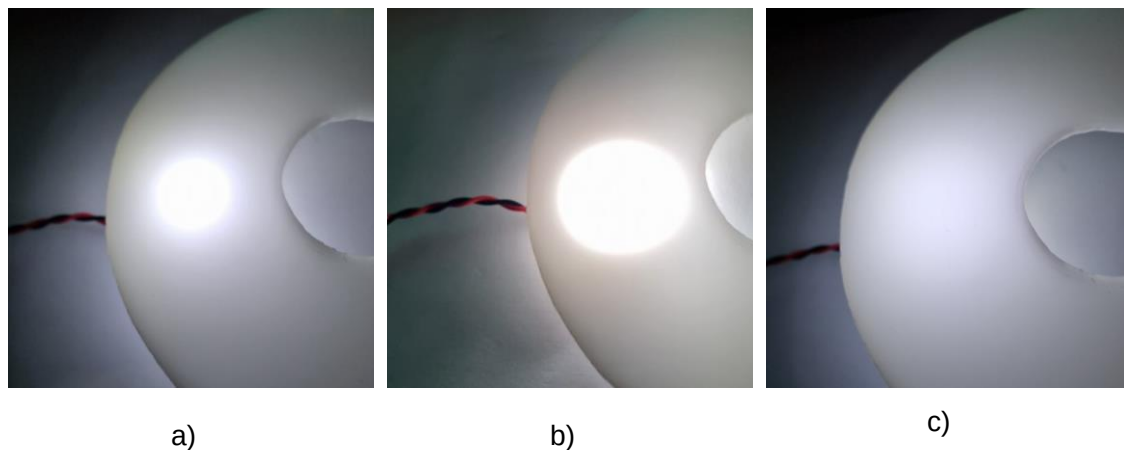


Figure 3.14 - Distance from LED to acrylic experiment, a) 10mm, b) 20mm, c) 35mm.

With this it was found that to achieve a more uniform light distribution, having the LED pointed directly at the acrylic, the LED would have to be at a distance greater than or equal to 35 mm. In order to try to reduce this distance it was tested a configuration where the LED is pointing against the structure, illuminating the acrylic by reflection of the light. For this situation it was also verified that painting the structure with white color produced better results, since it allows a greater reflection of the light rays. For this reason, the positioning of the LED was tested for distances of 5, 10, 15 and 20 mm. This variable was tested on images taken from a manometer with a black background. In this manometer it was verified that it was more visible the differences in illumination with the LEDs at different distances from the acrylic, having obtained the following results shown in Figure 3.15.

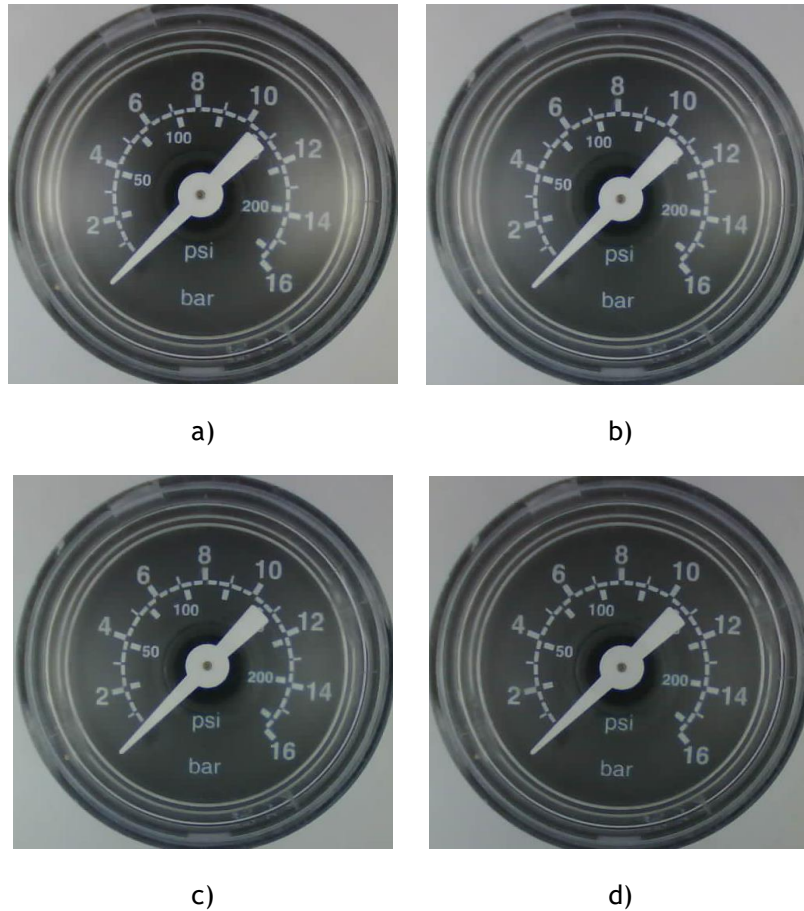


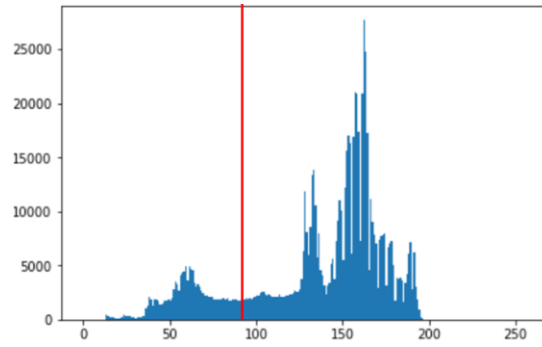
Figure 3.15 - Distance from LED to acrylic experiment, a) 5 mm, b) 10 mm, c) 15 mm, c) 20mm.

As can be seen, the image corresponding to the LED at a distance of 20 mm shows the best uniformity of illumination of the gauge. With this configuration it was possible to reduce the minimum necessary distance from the LED to the acrylic by almost half, while still being able to sufficiently illuminate the manometer. This will then be the value chosen for the distance from the LED to the acrylic for the dimensioning of the prototype.

To better illustrate this result, a binarization of the four images was made by applying a threshold value common to all four images and equal to 90, this value was chosen just for the purpose of highlighting the difference in brightness in the analog dial. The following image shows the binarized image, respectively to Figure 3.15, and its respective histogram of the image after grayscale shifting. This allows to see the number of pixels for each shade of gray, ranging from [0, 255]. Results shown in Figure 3.16



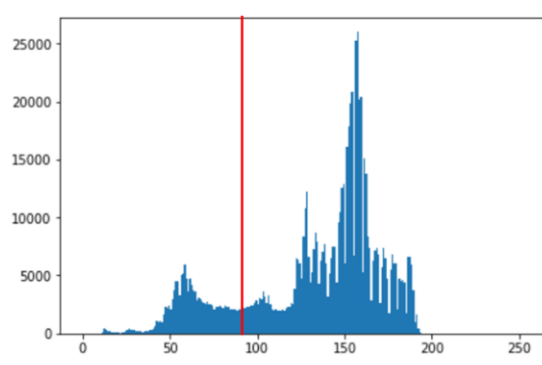
a1)



a2)



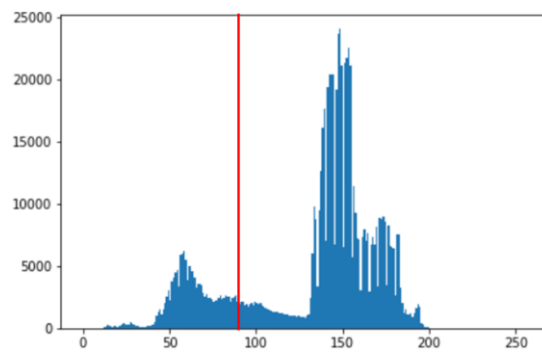
b1)



b2)



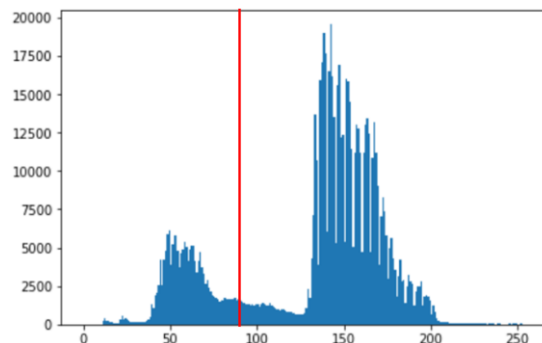
c1)



c2)



d1)



d2)

Figure 3.16 - Binarization with threshold = 90 of Figure 3.15 images, 1) image result, 2) histogram graph.

3.4 Communication

For the development of the code it was decided to use the Google Research product named Colaboratory [73], which allows developers to write and execute Python code through Google computing resources such as graphics processing units (GPUs). Google Colab allows to have multiple developers on the same project, and also provide the ability to share projects more conveniently.

Google Colab integrates with Google Drive and in order for everything to be in the same ecosystem, the microcontroller will send the image captured to the Google Drive, through HTTP protocol.

4. Algorithm development

This chapter will present the development of the algorithm capable of determining the value represented in an analog dial image, with presentation of the respective results that were obtained. The tests conducted here used real images, taken by the ESP-32 and uploaded to Google Drive, using the prototype that was developed above. The images used can be shown in Figure 4.1.



Figure 4.1 - Gauge images used to develop algorithm, a) white background gauge result, b) black background gauge result.

This algorithm was developed using these two manometers since they are significantly different from each other. This avoids tuning the procedure to a particular type of images. The main differences between them are presented on Table 4.1.

Table 4.1 - Characteristics of the gauges used to develop the algorithm.

Features	a)	b)
Background color	White	Black
Scale color	Black	White
Diameter [mm]	62.5	39
Scale marks	Spaced	Continuos
Value position relative to scale	Interior	Exterior
Value height [mm]	4	2.5

The methodology developed is based on a sequence of procedures that, from the initial image of the gauge, determines the center and angular position of the pointer, the start and end point of the scale with its corresponding values and computes the value represented on the image. The flowchart presented in Figure 4.2 summarizes the steps implemented in this work.

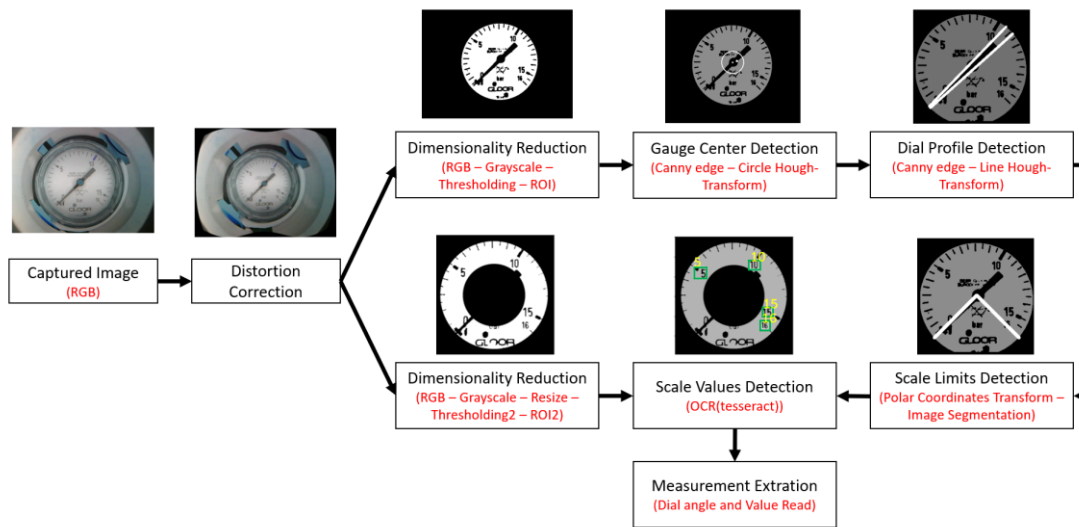


Figure 4.2 - Implemented workflow for detection of features present in an analog dial.

The detailed description of the algorithm is accompanied by pictures to better describe the implemented tasks. Initially the image received from the ESP-32 comes with a resolution of 800 x 600 pixels, and this resolution is kept constant for this developed prototype, thus ensuring a uniform size of all images being processed by this algorithm, allowing the dimensions of the filters to be defined by absolute values. As explained, the image was taken with a lens with an AFOV of 160° and as such, the image processing will start with calibrating the image received in the Cloud.

4.1 Calibration

As explained in chapter 2, the purpose of the calibration is to obtain the camera matrix (3×3) (See Equation 2.4) and the distortion coefficients (1×5) (See Equations 2.5 to 2.8). For this an OpenCV [61] implementation based on Zhang's method [74] will be used. Currently OpenCV supports three types of objects for calibration:

- Classical black-white chessboard
- Symmetrical circle pattern
- Asymmetrical circle pattern

For this case in particular, the classic chessboard pattern will be used. It is then necessary to take several shots of this pattern with the camera in use and run the calibration function in order to determine the necessary parameters. To solve the equations, it is needed at least a predetermined number of pattern snapshots to form a well-posed equation system. For example, in theory the chessboard pattern requires at least two snapshots. However, this results in a significant amount of noise present in the input images, so for good results it will be needed at least 10 good snapshots of the input pattern in different positions, based on Zhang's documentation [74].

Figure 4.3 presents a photo taken of the checkerboard pattern (a), an image in which the pattern determination is illustrated (b), as well as the corresponding corrected image (c), taken with the camera and lens chosen on chapter 3.2.

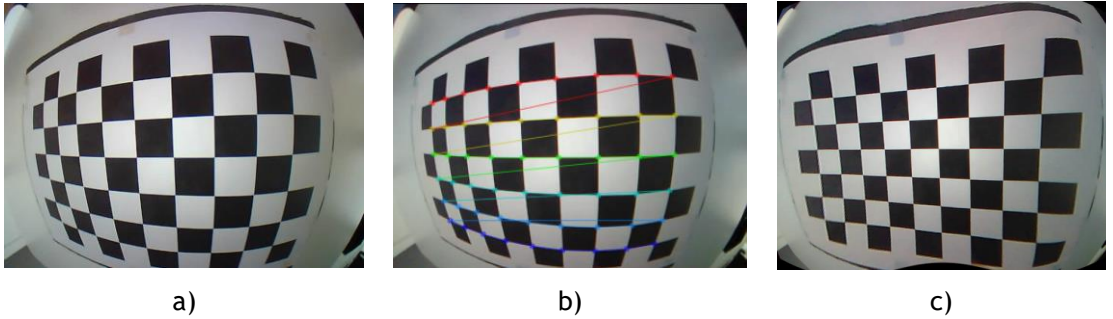


Figure 4.3 - Image calibration process, a) Image to calibrate, b) Checkerboard pattern recognition, c) Image calibrated.

After running the OpenCV calibration code, the respective parameters were determined, and the following results were obtained. Results are presented on Equation 4.1 and 4.2.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 251.93 & 0 & 384.19 \\ 0 & 251.50 & 309.02 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (4.1)$$

$$Distortion\ coeff. = [-0.096 \quad -0.0072 \quad -0.0024 \quad 0.00087 \quad 0.0033] \quad (4.2)$$

Once the characteristic variables of the camera and lens have been determined, it is no longer necessary to perform this calibration determination, so once these were determined, they were saved. The calibration code and its storage of the variables needed to perform the image calibration are presented in Annex B.

4.2 Dimensionality Reduction

The initial image comes in three color planes, red, green and blue (RGB), so the next step is to convert a standard 3 channel 24-bit (RGB), into a single channel 8-bit per pixel grayscale image ($Width \times Height$), which results in an image with only shades of gray. This sufficiently preserves the required visual features, edges, and contours, with required luminance levels minus the chromatic noise, significantly reducing the size of information contained in the image. For this prototype it was used the standard OpenCV *BGR2GRAY* function [75].

Conversion to (*black/white*) binary image is also required to completely suppress unwanted information from the Image. Binary images are usually obtained by applying thresholding to a grayscale image, which essentially replaces all pixel values greater than the threshold to white, (value = 255) and others to black, (value = 0). For this, at this stage the histogram is calculated with the distribution of the intensities of the pixels in the image. Considering that there must be two groups of elements, ROI, and background of the image, to obtain the optimal threshold of separation between the two groups of elements of the image was applied the Otsu's algorithm [76], from OpenCV. Results from this operation, applied to both images from Figure 4.1, are shown in Figure 4.4.

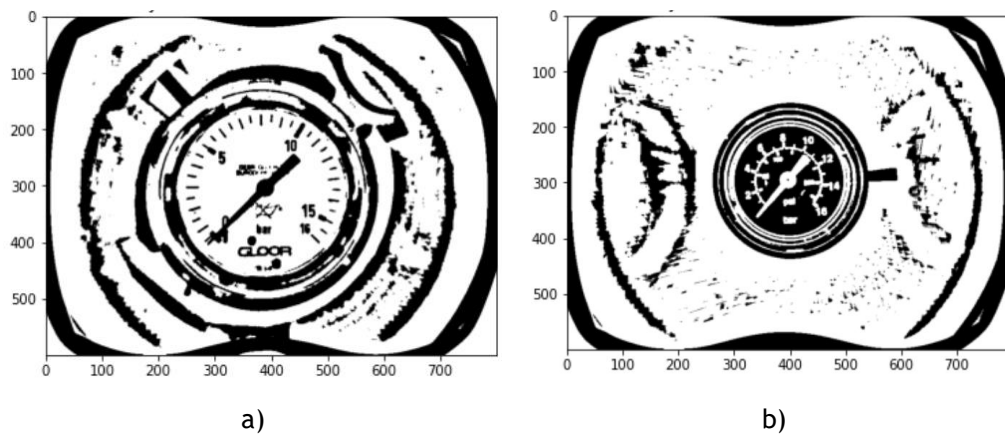


Figure 4.4 - Binarization result images, a) white background gauge's results b) black background gauge's results.

4.3 Region of Interest detection

Once the image has been binarized, follows the determination of the ROI. In this case it consists of detecting the circle that contains the dial and scale of the manometer.

This step is central to any task involving image processing because keeping any unwanted element can lead to erroneous results. On the other hand, eliminating objects of interest can result in the inability to perform the intended action. For this task it is used the previous binary image, and on this a Sobel filter is applied to obtain the contours of the image, followed by the Hough transform for circles, with increasing radius until half of the smallest dimension of the image. The Sobel filter is a derivative filter that applied to the binary image gives its contours [77]. The purpose of applying it in this case is also to avoid excess of possible circles in the response of the transform. The images in the Figure 4.5 shows the results obtained when determining the ROI of the previous example image.

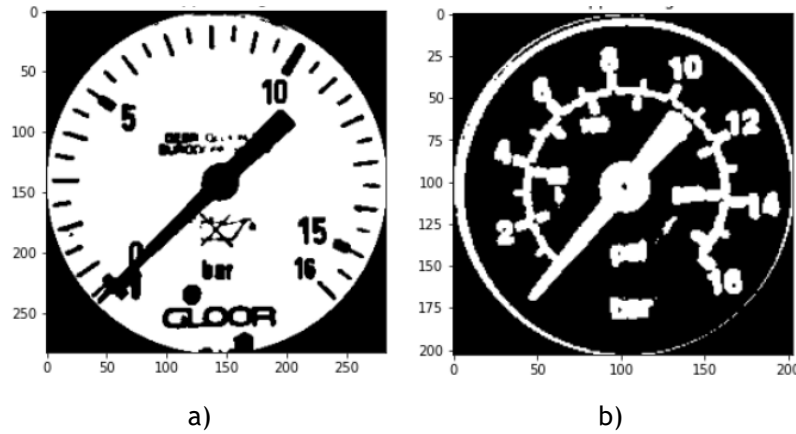


Figure 4.5 - ROI result images, a) white background gauge's results b) black background gauge's results.

4.4 Center detection

The determination of the ROI produced satisfactory results, however, the detected circle did not have its center properly aligned with the center of the gauge. Therefore, another function was used to detect the center of the gauge, taking advantage of the fact that the gauge needle has a small circle on which the needle rotates. For this the Hough transform function was parameterized to detect small circles in the center of the image and applied to the previous ROI image. A condition was also added in order to improve the result obtained. Taking advantage of the fact that the camera is axially aligned with the center of the gauge, a condition was imposed to choose, from the detected circles, the one whose center has the smallest distance to the real center of the image (400, 300). The distance between two points was calculated based on Equation 4.3.

$$distance = \sqrt{(x_1 - 400)^2 + (y_1 - 300)^2} \quad (4.3)$$

where (x_1, y_1) is the point of the detected center.

In order to illustrate the difference found between the centers obtained in the ROI detection and those obtained through the code implemented here the Figure 4.6 is presented.

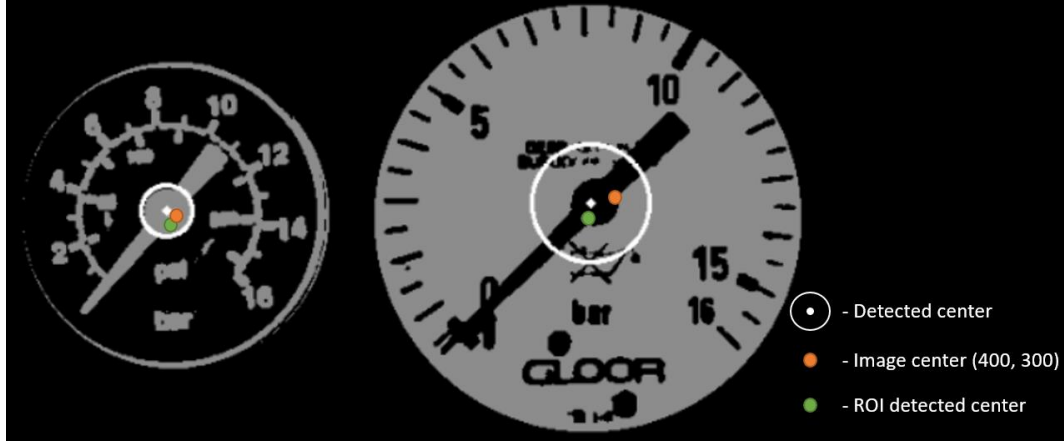


Figure 4.6 - Detected centers comparison results, left - black background gauge's results, right - white background gauge's results.

4.5 Dial detection

The next step is to determine the angular position of the pointer. At this stage, a Canny edge detector was applied, and a Probabilistic Hough Line Transform [78] was implemented on top of it.

To ensure that only two straight lines are detected, a variable of the Hough transform function for lines was used, accumulator threshold parameter. This variable ensures that only those lines that get enough votes are returned. This means that the higher the threshold value, the fewer the number of straight lines are detected. Knowing this, a code was implemented that starts the search for the straight lines with a sufficiently low threshold value to ensure that on the first time the loop runs more than two straight lines are detected, even for the smallest gauge. This threshold value is then increased in a loop that stops when only two straight lines are found.

The Hough transform used generates two points that define each detected straight line. The goal now is to calculate the point of intersection of these two lines, which, by connecting this point to the center of the gauge, detected earlier, allows us to define the angle of the needle. To do this it is first necessary to determine the slope of each straight line, defined by two points each, (x_1, y_1) and (x_2, y_2) , using Equation 4.4.

$$slope = m = \frac{y_2 - y_1}{x_2 - x_1} \quad (4.4)$$

Having the respective slope of each line the point of intersection with y-axis, b , is determined, thus getting each straight line defined by the expression, $y = m \times x + b$, it is now possible to compute the point of intersection, (x_0, y_0) , of these two lines through the Equation 4.5.

$$\begin{cases} x_0 = \frac{b_2 - b_1}{m_1 - m_2} \\ y_0 = m_1 \times x_0 + b_1 \end{cases} \quad (4.5)$$

As said before, since the center of the gauge has already been determined, it is now possible to compute the detected needle angle, by joining the interception point, from Equation 4.5, to the center. Figure 4.7 shows the result of the two lines detected using the method explained above.

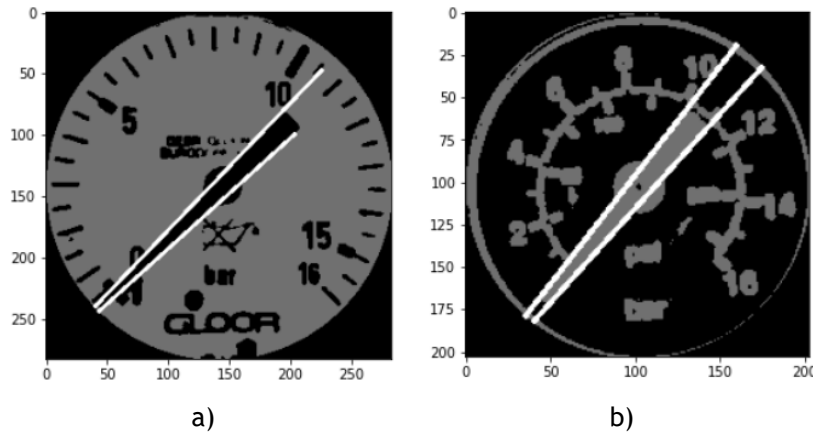


Figure 4.7 - Dial detection illustration, a) white background gauge's results b) black background gauge's results.

4.6 Scale limits detection

After a detailed analysis of different manometers in the literature, it was found that the smallest amplitude between the beginning and end of the scale is defined by empty space at its limits which is commonly observed in manometers. In order to automatically detect the measured value without any user interaction, it was decided to create a function to determine the start angle and end angle of the scale, and then run an OCR program to determine automatically its respective values. For this a polar transform of the image is performed.

Once a correct and adjusted detection of the manometer ROI has been verified, it is now possible to define a strip that includes the manometer scale, defined from the limit radius detected by the ROI, with a fixed width (w), illustrated in Figure 4.8 a). By analyzing the images of the manometers tested here and by experimental tests, it was concluded, empirically, that a width of 60 pixels was sufficient to include the scale of the manometer on the strip where the polar transform will be performed. Once this width is

defined, the polar transform of the strip mentioned above is then performed as shown in Figure 4.8 b) and c).

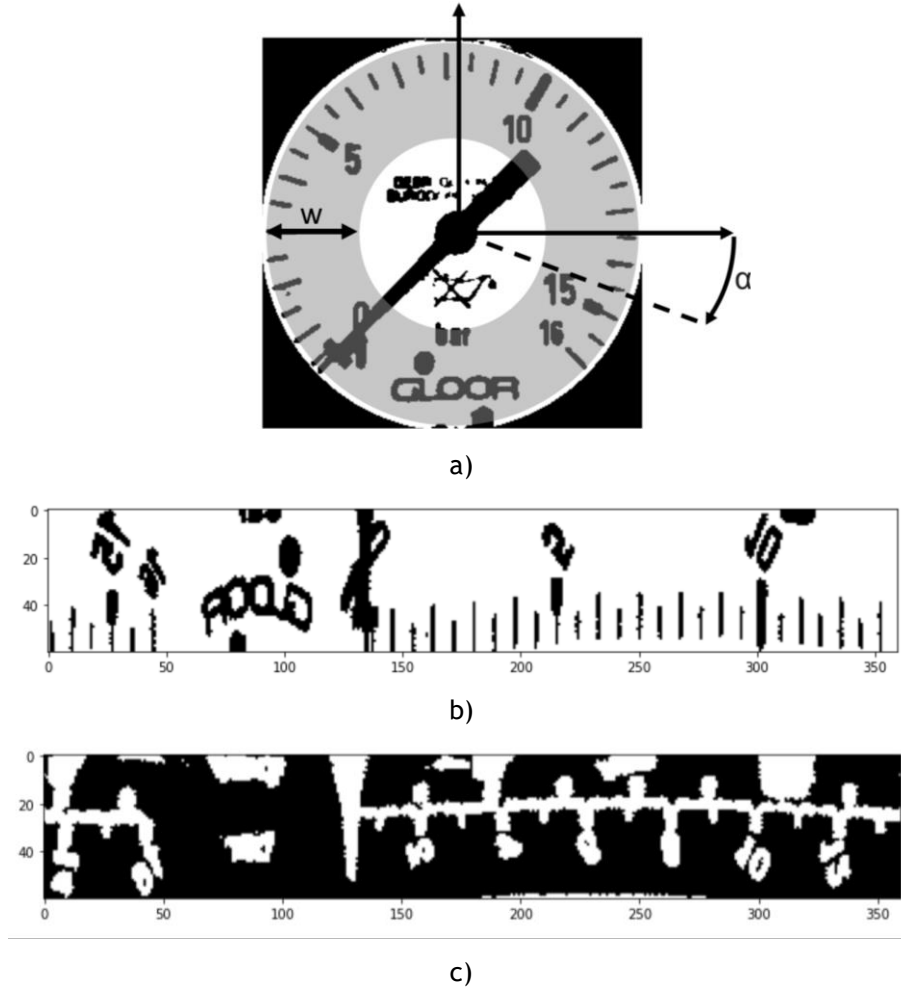


Figure 4.8 - Image polar coordinate transform, a) illustration of the area of interest, b) white background gauge polar transform c) black background gauge polar transform.

On this image it is necessary to determine whether the gauge has a white background with black scale or a black background with white scale. Knowing this it will be then possible to count the number of pixels that correspond, respectively, to the analog dial scale.

At this stage a rectangle ($d \times w$) is defined, where (w) is the width defined previously, representing in this case the height of the rectangle, and (d) is the variable to be determined, representing the width of the rectangle. Once this rectangle has been defined, the resulting image from the polar transformation is scanned, from the left to right, of a portion of the image defined by the rectangle to be used, in this portion of the image the number of pixels corresponding to the gauge scale will be counted in order to find at least two areas where the pixel count is minimal or null, corresponding to the beginning and end of the scale. If the gauge has a white background, it counts the black pixels, if it has a black background, it counts the white pixels.

In order to determine the variable (d) that produces the best results, two manometers will be used. These manometers were chosen since they feature quite different scales, the white manometer features a scale with radial strokes with spacing between each stroke of the scale, and the black manometer features radial strokes joined by a straight line. For this trial it will be performed two step size areas over the image: (d) by (d), shown in Figure 4.9 a) and c), and ($d/2$) by ($d/2$), shown in Figure 4.9 b) and d)).

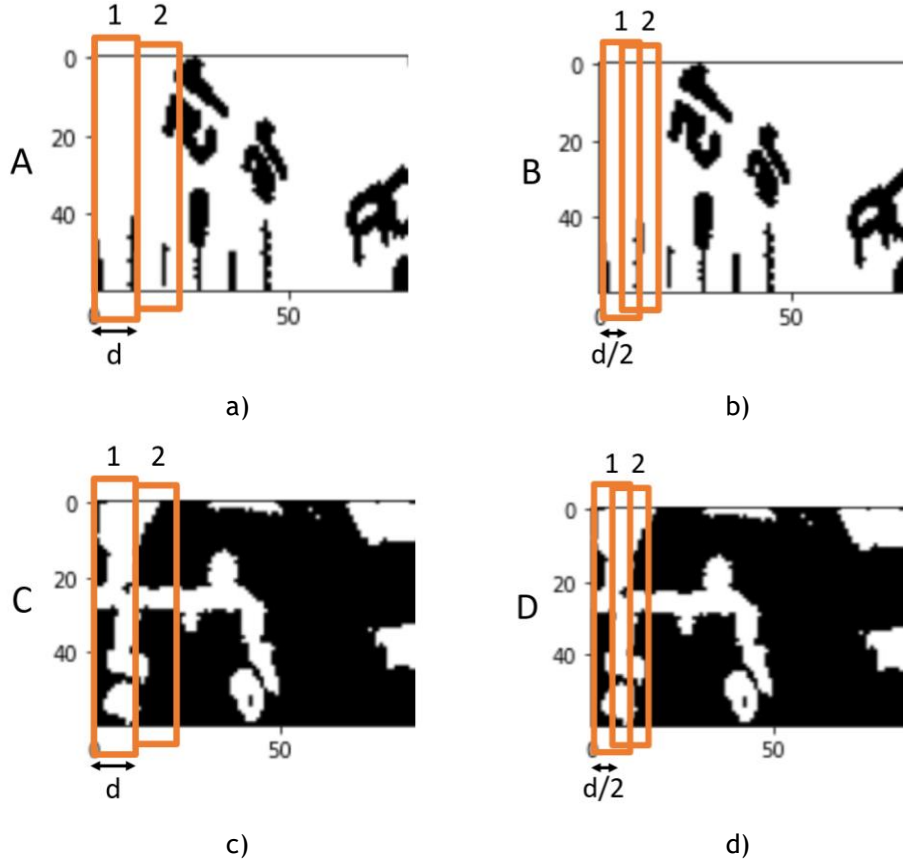


Figure 4.9 - Illustration of different scans performed to image, orange rectangle ($d \times 60$), a) white background gauge scan with d advances, b) white background gauge scan with $d/2$ advances, c) black background gauge scan with d advances, d) black background gauge scan with $d/2$ advances.

As explained, this scan will be done to determine the two zones of the image without pixels corresponding to the scale. To determine which of the zones corresponds to the beginning of the scale and to the end of the scale two factors must be taken into account: (i) the manometers have increasing scale values in the clockwise direction; and (ii) the polar transform performed starts from the right horizontal axis and follows the clockwise direction, as represented in Figure 4.8 a). The result of the transform comes with the x-axis in degrees from 0 to 360.

Knowing this, it is then possible to impose the condition that the detection with the highest value, in degrees, corresponds to the beginning of the scale and the detection with the lowest value to the end of the scale. However, for the case where the beginning of scale is in the first quadrant and the end of scale is in the fourth quadrant the above condition cannot be applied, as exemplified in Figure 4.10.

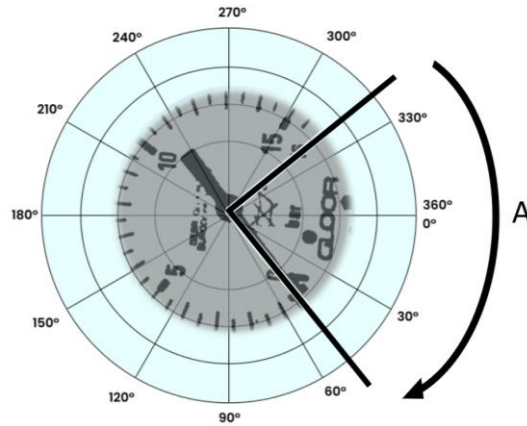


Figure 4.10 - Gauge position relative to polar coordinate system.

It is however possible to circumvent this situation by assuming that the smallest amplitude between the scale limits, represented in Figure 4.10, by the letter A, will be less than or equal to 180° . Therefore, it is now possible to impose a condition that, if the difference between the two angles obtained are greater than or equal to 180° , the detection with the highest value corresponds to the end of the scale and the detection with the lowest value to the beginning of the scale. After this, the solution presented was tested for different values of (d) to determine the one that allows the correct reading for these two manometers. The results are presented on Table 4.2 and the methods are named based in Figure 4.9.

Table 4.2 - Image scan results - (N.D. = Non Detected, N.A. = Non Acceptable).

Method	Increment	d [°]	Scale start detected [°]	Scale start error [°]	Scale end detected [°]	Scale end error [°]	Overall assessment
A	d	5	-	.	-	-	N.D.
		10	60	15	-	-	N.D.
		15	-	-	-	-	N.D.
B	d/2	5	57	12	-	-	N.D.
		10	55	10	130	5	OK
		15	56	11	-	-	N.D.
C	d	5	55	10	125	5	OK
		10	60	15	130	0	N.A.
		15	-	-	130	0	N.D.
D	d/2	5	54	9	117	13	OK
		10	55	5	130	0	OK
		15	56	8	127	7	OK

Through the results obtained it was found that scanning the image with 5 by 5 degrees advances with a rectangle width, d , of 10 degrees, allows an acceptable reading corresponding to the gauge's beginning and end of scale, so these parameters are used in the algorithm developed for this prototype. Such results can be seen in Figure 4.11. An scan's example performed by the code developed here is presented in Annex C.

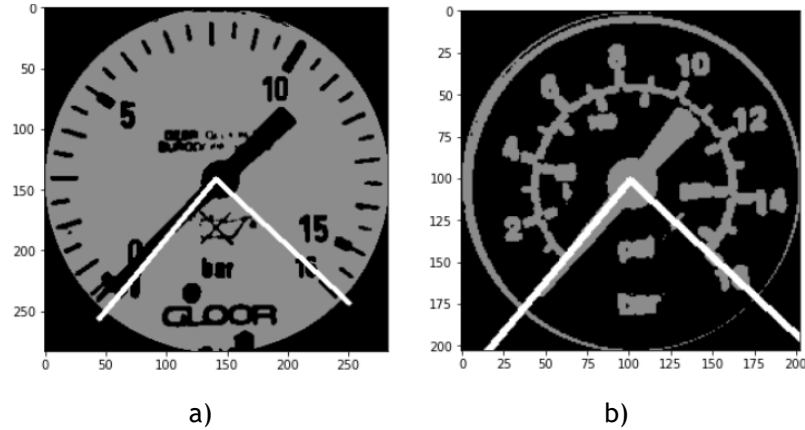


Figure 4.11 - Scale limits detection illustration, a) white background gauge's results b) black background gauge's results.

4.7 Scale values detection

In order to be able to determine all the manometer's parameters, it is necessary to determine the scale values. For this it will be used an optical character recognition (OCR) algorithm. Several codes have been developed to date, however, open-source algorithms available in the literature will be used for this project. Among the ones available, it was decided to test the Tesseract [66] algorithm at this stage. To detect only the digits belonging to the scale, a new ROI was applied to the previously binarized image, and the OCR algorithm was run on it. After analyzing the results, it was found that it was not possible to detect the digits of the manometer with the black background.

After a detailed analysis of the tesseract documentation, it was found that this OCR only detects black digits, so, since it is already determined from previous processing, whether the gauge has a black or white background, if this condition is met, the colors of the image are inverted. After this, the OCR algorithm was run again, result in Figure 4.12. The next series of tests will be performed on the smallest gauge, which has the smallest digits and is the one on which the OCR algorithm has the most difficulty in detecting the digits.

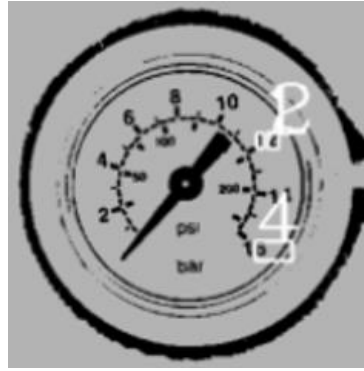


Figure 4.12 - OCR Tesseract result on inverted binary image, black background gauge.

As it shows, in Figure 4.12, although some digits have already been found, the detection is considered insufficient. Once again it was done a review of the documentation presented by Tesseract where it is described a list of advisable procedures for the treatment of the image before carrying out the detection of the digits through this OCR algorithm. Among the procedures mentioned, it was pointed out that rescaling the image can improve the detection of digits, which is due to the fact that this algorithm needs a minimum number of pixels to define the number to be detected [66]. Based on the Tesseract documentation, in order to obtain a zero-error detection with this algorithm, it is necessary for the character to be detected to be defined, at its height, by 30 to 33 pixels, as shown in Figure 4.13.

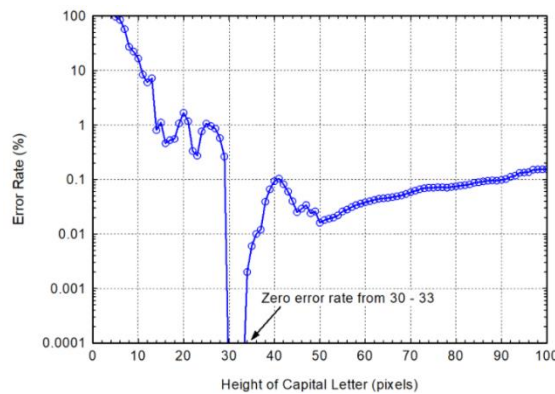


Figure 4.13 -Tesseract Error rate function of Letter pixel height graph [66].

One solution to increase the number of pixels defining each digit of the image would be to increase the resolution of the image captured by the ESP-32. This solution was tested, however and to try to keep the resolution as low as possible it was also tested performing an image rescaling from the previously used image. Given these results, it turns out that the difference between using a higher resolution image compared to a 600 x 800 resolution image with resize is not significant. It was then decided, for this work, to use an image with a resolution of 600 x 800 and perform a resize of 200%. A mask was also applied to the image, creating a new ROI for the number detection.

From the study carried out in this phase it was concluded that, for a manometer reading with 39 mm diameter, where each digit has a height of 2 mm, the tesseract algorithm can detect 5 of the 8 digits represented, 62.5%. On the other hand, for a manometer with a diameter of 61.5 mm, with each digit having a height of 4 mm, the detection was 100%, results shown in Figure 4.14.

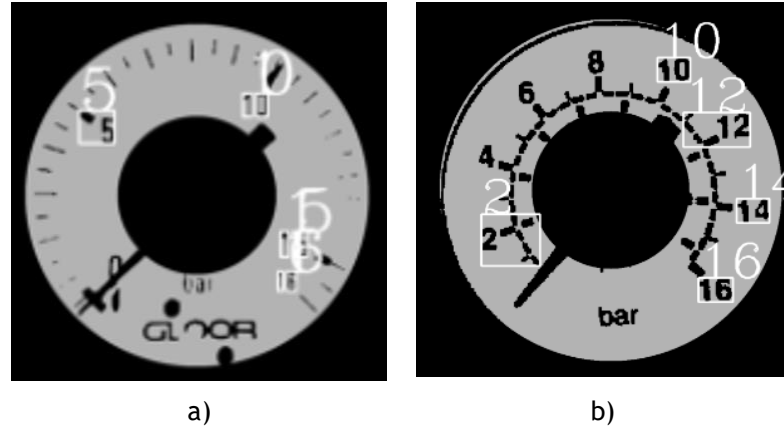


Figure 4.14 - OCR detection, a) white background gauge, b) black background gauge.

Since the scale limits are already set, if all the gauge's digits are not detected and the gauge is symmetrical, the detection of the value represented on the gauge is guaranteed without the need for any user input. If it is not symmetrical, 3 user inputs will be required, explained on the 4.9 section. If all digits of the gauge are detected, then automatic detection for symmetrical and asymmetrical gauges is ensured by the methodology proposed here.

4.8 Symmetry detection

Since, under the worst-case scenario, for the 39 mm diameter analog dial, 62.5% of the digits are detected, it is then possible to automatically check whether the gauge is symmetrical or not. Once the OCR algorithm used detects the value and its position, it is possible to determine the angular amplitude between consecutive digits, as illustrated in Figure 4.15. To do this, the interior angle between the center and two consecutive digits position was calculated and stored the value on *ang* vector.



Figure 4.15 - Illustration of angular amplitude between detected values.

Knowing the angular amplitude between the detected digits, the angular resolution between consecutive values can be determined, according to Equation 4.6.

$$\text{Angular Resolution} = \frac{val[i+1]-val[i]}{ang[i]} \quad (4.6)$$

where *val* is a vector where the detected scale values are stored, in ascending order and *ang* is a vector where the calculated values of the angle between consecutive scale numbers are stored.

To determine whether or not the manometer is symmetrical, the average of the angular resolutions is determined, and the difference between each angular resolution and the average obtained was determined. The condition can be imposed that if any of the differences calculated is equal to or greater than 30%, obtained empirically, of the average, the manometer is considered asymmetric. Otherwise, the conclusion is that it is symmetric.

4.9 Value computation

Once the beginning and end of the scale, center of the gauge, the point that defines the needle, and the symmetry of the gauge are identified, the value indicated on the gauge by the needle can be calculated.

The calculation begins with the determination of the angle of the needle in relation to the beginning of the scale and then determining the value of the beginning and end of scale. Assuming that the manometers analyzed here have a 0 value at the beginning of scale, the developed algorithm will have to determine the end-of-scale value automatically. If the gauge is asymmetric, it requires user input of the end-of-scale value, as well as the angle from which the scale takes another resolution and the respective value corresponding to that angle. Having now all the necessary variables the value represented on the manometer can then be calculated, the code developed here is presented in Annex D.

5. Results

In this chapter the end result of the developed prototype is presented. Here the structure of the prototype and its constraints are described in detail. The results obtained after performing a series of tests on the algorithm will also be presented, and some of its limitations will be analyzed afterwards.

5.1 Prototype design

For the design of the prototype some precautions had to be considered. Since the parts were produced using the 3D printing manufacturing process, the lack of precision associated with the 3D printer had to be taken into account, as well as the fact that plastic shrinks when it cools. Therefore for the development of the prototype the necessary tolerancing was taken into account. Another limitation of this manufacturing process is the creation of threaded surfaces for screws. This impossibility led to the use of a screw and nut assembly for attaching the acrylic, as well as for attaching the mount for the camera and the power supply.

5.1.1 Structure design

Having all the dimension variables determined from chapter 3, it is now possible to design a prototype that guarantees all the conditions that have been mentioned throughout the work. In order to better understand the needs of the prototype to be dimensioned, a summary of the main conditions that needed to be ensured are presented here.

I. Positioning of the ESP-32 and camera lens relative to the manometer.

For the correct positioning of the ESP-32 microcontroller, a custom part was designed. This piece ensures that not only the ESP-32 is centrally positioned in relation to the manometer, but also that the microcontroller is correctly fixed without allowing it to rotate or move away from the structure, illustrated in Figure 5.1.

It is also ensured that the camera lens is at 30 mm from the manometer, this value is a system requirement, as shown in section 3.2. This distance will

be used for reading any manometer, thus allowing the universality of the structure.

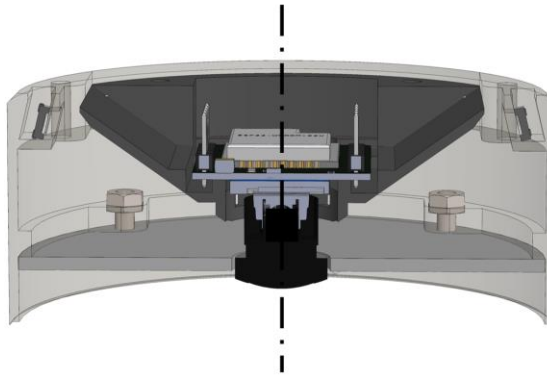


Figure 5.1 - Detailed view of ESP-32 on developed structure.

II. Positioning of two LEDs pointing at the structure.

According to the studies demonstrated on 3.3 section, it was verified that two LEDs pointed at the structure of the prototype allowed a correct illumination of the gauge, for this a structure was designed as illustrated in Figure 5.2.

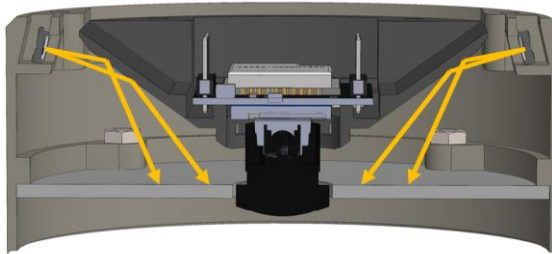


Figure 5.2 - Illustration of the LED lighting on the structure.

III. Positioning of two LEDs at a distance of 20 mm from the acrylic.

The developed structure allows the acrylic to be fixed at a distance of 20 mm from the two LEDs, this value is a system requirement, and was determined in section 3.3. The fixing of the acrylic is ensured through the use of four sets of bolt and nut, shown in detail in Figure 5.3.

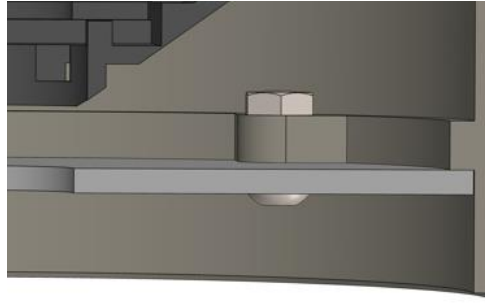


Figure 5.3 - Detailed view of the acrylic fixation.

IV. Possibility of visually read the manometer.

Another requirement of the product to be developed is that the operator needs to be able to read the gauge visually, if desired. For that, the structure was divided into two main parts. The support for the camera, acrylic, LEDs, and battery and the support that will attach to the manometer. These two pieces are then joined together using a hinge, as shown in Figure 5.4 the prototype on open position.

The possibility of axially rotating the top support to the side was also studied, however, and for reasons such as the one exemplified in Figure 3.9, such a solution would pose problems for cases where the gauges are placed laterally and close to each other. Therefore, it was considered that the use of a hinge for this purpose would be a more viable solution.



Figure 5.4 - Product view in open position, using a hinge.

V. Positioning of the power supply

Knowing that this product should have its own power supply, a part was designed to allow its attachment to the structure. Knowing also at this point

that the use of three to four 1.5 V batteries each are sufficient for the correct power supply of the ESP-33, the following piece was designed to be positioned on top of the camera, as shown in Figure 5.5.

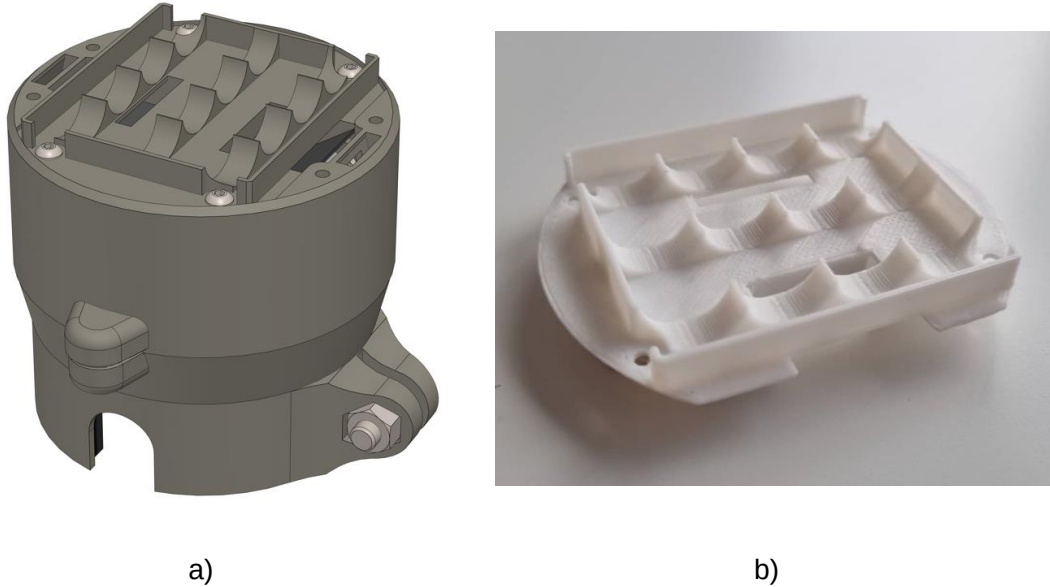


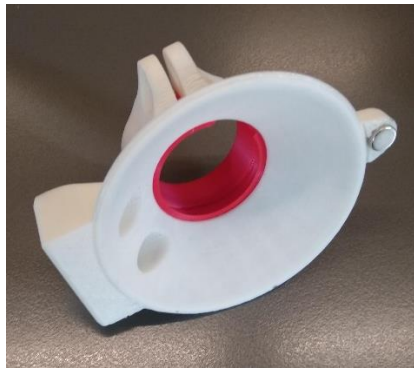
Figure 5.5 - a) Top view of prototype without upper cover, b) real image batteries support

VI. Attachment to manometers with diameters from 30 mm to 80 mm.

One of the conditions of this prototype is that it can be used for any circular manometer with diameters ranging from 30 to 80 mm. The development of five manometer mounting brackets with diameters of 40, 50, 60, 70, and 80 mm, as presented in Figure 5.6 a) and b), was considered as a solution.

In order to allow the attachment to a manometer whose diameter is between two of these consecutive values, three filling rings were developed for each of the supports, with a thickness of 1.5 mm each, shown in Figure 5.6 c).

The tightening of the support to the manometer is ensured by the use of a screw and nut set, and the support also has a strip without material of 3 mm width that allows a better adjustment of it, as illustrated in Figure 5.6 d). The filling rings were designed with a system of projections and indentations in order to prevent the relative movement of the ring in relation to the support, as well as to an inner ring to its outer ring, shown in Figure 5.6 e).



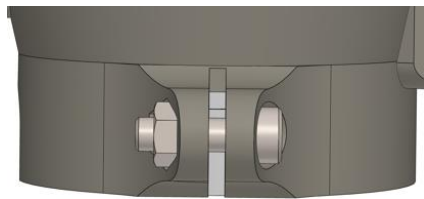
a)



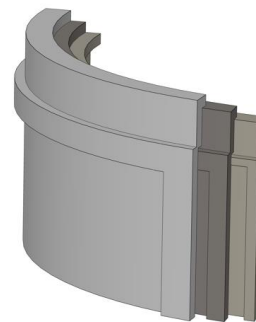
b)



c)



d)



e)

Figure 5.6 - Mounting details, a) real image from 40 mm support, b) real image from 70 mm support, c) exploded view of filling rims, d) tightening detail, e) detail view of filling rim fittings.

Having explained in detail the conditions that are necessary to ensure in the construction of this prototype, a picture of the complete part is now presented, followed by a sectional view, as well as an exploded view in Figure 5.7. The technical drawings of every part developed here can be found in Annex H.

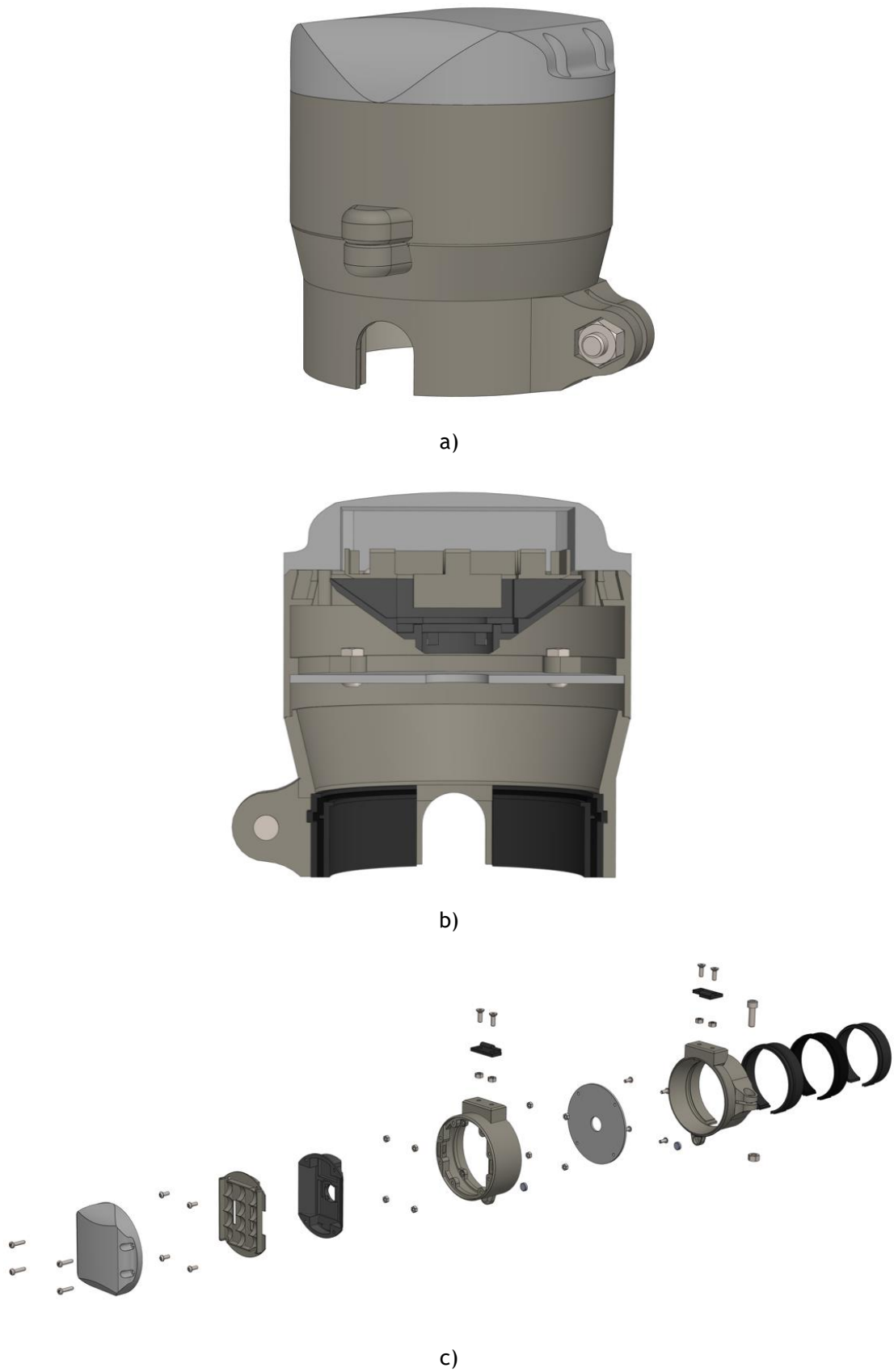


Figure 5.7 - Developed Prototype, a) complete view, b) sectional view, c) exploded view.

5.2 Algorithm validation and limitations

Having developed the algorithm, it was then tested resorting to a set of 10 manometer images. A diverse set of images was used to verify whether the code is biased for a particular type of gauge. The results obtained showed identical performance between them. Note that eight of the images used are from the internet and therefore do not have a constant lighting or centered orientation. For these images all the procedures described on chapter 4 were tested, except for the detection of the scale values, since the images have a different resolution from each other, such analysis would be irrelevant.

It can be seen in Annex E, illustration of the detections obtained in the 10 gauges going through the steps of the algorithm. The results obtained are shown in the Table 5.1. In these results, since the scale value detection algorithm was not run, the beginning of the scale is considered to have a value of 0 and the end of the scale is assigned a value of 100, so the computed value is given as a percentage. All the analog dials tests were done with the dials on the zero position, so it is expected to obtain a computed value of zero, any number different than that is considered to be an error of equal amplitude of the value computed.

Table 5.1 - Algorithm validation results.

Manometer	1	2	3	4	5	6	7	8	9	10
Binarization	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
ROI detection	OK	OK	OK	OK	OK	OK	OK	OK	OK	FAIL
Center detection	OK	OK	OK	OK	OK	OK	OK	OK	OK	FAIL
Dial detection	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
Scale limits detection	(1)	OK	OK	(1)	OK	(1)	OK	OK	OK	OK
Compute Value [%]	1.52	0	0.41	1.62	0.34	1.21	0.17	0.42	0.23	0.40

Manometers 1, 4 and 6 shown an error larger than 1%. This is due to the fact that the detection of the beginning of the scale does not coincide with the manometer's 0 value. Some manometers have a needle holder, and if it has the same color as the scale, the algorithm developed for scale limit detection thus assumes that the needle holder is part of the scale, as shown in detail the dial holder Figure 5.8.

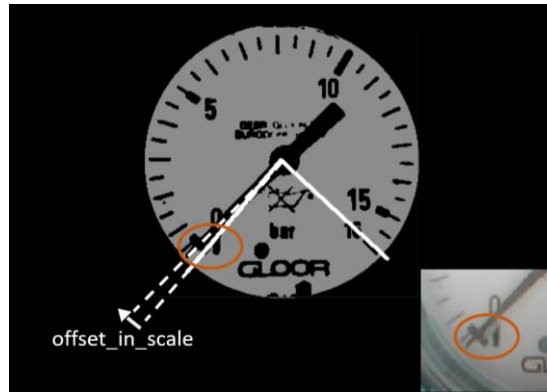


Figure 5.8 - Beginning of scale correction.

In order to resolve this problem, it was decided to make available in the code a variable where the user has the option to assign a value to it. This value causes the advance of the beginning of the scale clockwise, thus being able to adjust the scale detection and refine the computed value from this algorithm. It was also found that the detection of ROI and center of gauge 10 were not the most accurate. For this, two more variables were also made available in the code for the user to be able to adjust the parameters. These variables are explained in detail in the Annex F.

The Tesseract OCR algorithm allowed to detect 62.5% of the scale values for the 39 mm diameter gauge and 100% for the 62.5 mm gauge. For both, the accuracy of the detected values was 100%. These values are considered acceptable for the project developed here, it allowed an automatic detection of the gauge symmetry, since it detects more than half of the scale values. This allowed the automatic computation of the value without any user input, if the gauge is symmetrical. It can be concluded that the difficulty in detecting the digits on the 39 mm diameter manometer is due to the small size of its digits.

When detecting the scale values, it was also found that the configuration function of the tesseract algorithm presented a flaw. This configuration limits the OCR code to detecting only digits in the image, ignoring any symbol or letter found. Such a flaw brought implications for the code developed here, since there were cases where upon detecting a value, it also detected a symbol next to that value. The implication of this is that the contour generated when looking for the scale values may be larger than it should be, as illustrated in Figure 5.9.

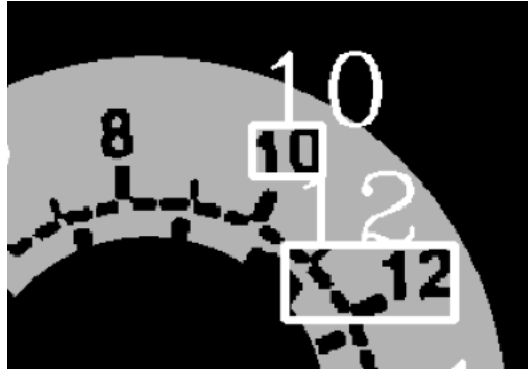


Figure 5.9 - OCR error illustration.

For the case shown in Figure 5.9, the tesseract algorithm detected a quotation mark and joined it to the digit 12 also detected, thus creating a larger contour than it was supposed to. Since to know the position of the digit the center point of the generated rectangle is determined, when this case happens, the position of the digit is induced in an error. This has implications for the detection of gauge symmetry and is one of the reasons given for the 30% threshold value presented on the 4.8 section.

It was found that the developed algorithm takes 25 seconds, to detect all the parameters, using the Google computing resources provided by Google Colab. However, if it is ensured that after applying the device, it is not rotated on the gauge, then it is possible to save certain parameters to be used later. Thus, the algorithm when it runs for the second time, can process the image in less than 1 second. For this, it is then necessary to save the following variables detected by the algorithm:

- center (x, y)
- radius (float)
- scale starting point (x, y)
- scale end point (x, y)
- symmetry (bool)
- scale starting value (int)
- scale end value (int)

Having these variables stored, a code was developed in which it only needs to perform the calibration of the image it receives, binarization and needle detection, shown in Annex G. This way it achieves a much less time-consuming processing. However, in the case where the gauge reading is done in periods longer than the calculation time, this is not necessary, but it helps to reduce the computer's workload in calculating its value.

6. Conclusion and Future Works

6.1 Conclusion

With the continuous development of computer technology and image processing technology, the automatic reading technology for analog dials has emerged. This technology improves the level of automation of dial identification and will be extensively used instead of the conventional industrial dial reading method. The analog dial automatic reading system refers to a system that analyzes the pointer dial collected by the camera through image processing methods such as machine vision and automatically recognizes the dial indication. Its main purpose is to replace manual readings, reduce workload, and improve work efficiency.

This work presents a methodology for creating a mechanism capable of capturing an image and sending it to the Cloud as well as the implementation of an algorithm for reading automatically the value represented in an analog dial. The goal was to create a product with enough versatility to be used in several types of industrial gauges with diameters ranging from [30, 80] mm, with a non-invasive installation. For the code was taken into account not to use proprietary tools, for that, it was decided to develop the algorithm on the Google Colaboratory platform, in Python, using libraries specific to the area under study. Another goal was to create a product with an open platform, that is, capable of being used with any desired Cloud service. As a proof of concept, the image captured by the ESP-32 was sent to Google Drive, through the HTTP protocol. This microcontroller can also establish different types of communications allowing connection with various Cloud services available on the market today, thus having the versatility desired for this work.

The results obtained were considered satisfactory and of sufficient quality to proceed with further developments. The device created can capture an image, with low power consumption, send the image via Wi-Fi to the Cloud, and extract fundamental aspects of the analog dial such as its center, radius, scale limits and respective values. It has the ability to automatically detect the value represented by the analog dial needle, without any input, in cases where the gauge is symmetrical.

The work done here has shown that for achieving proper illumination of a curved reflective surface requires a careful approach, for which it became necessary to perform

several experimental tests either by making use of more specialized techniques such as the use of specific structures to diffuse light or by using everyday materials, such as fabrics, to filter the light.

A validation of the algorithm was done in this work using 10 different manometers, having successfully detected 87.5% of its fundamental characteristics for the correct computation of its value. The determination of the represented value, without any user input, obtained a maximum error of 1.6%. For these cases, there is the possibility of parameter adjustment by the user, reducing the error of the value calculated.

6.2 Future work

The capabilities of the project developed here were demonstrated, and as such, some procedures to be taken in the future are now suggested in order to improve and enhance the final product.

It is therefore proposed to make an implementation that detects from which value the scale changes its resolution, this will allow automatic detection for symmetrical or asymmetrical analog dials, without the need for user input. To do this it is advisable to improve the OCR detection to ensure that all values of the scale are always detected. It is also advisable to test different open-source OCR algorithms available.

The purpose of this product is to read the value represented on the manometer periodically. The periodicity with which the microcontroller captures the analog dial image and sends it can be remotely controlled with, for example, an MQTT communication, thus being able to establish different periodicities for each product applied in the factory manometers. Given the I/O inputs provided by the microcontroller, it is also possible to add, for example, piezoelectric accelerometers that sense vibration. This would give the option to send an error message to the Cloud service if there were vibrations on the analog dial that would disturb the capture of a quality image.

In order to ensure a better insulation of the system created, the use of seals should be studied, allowing in this way the use of the device in outdoor areas without problems.

References

- [1] “Welcome to Python.org.” <https://www.python.org/> (accessed Mar. 02, 2022).
- [2] U. Isikdag, *Internet of things: Single-board computers*, vol. 0, no. 9783319218. 2015. doi: 10.1007/978-3-319-21825-0_4.
- [3] K. J. Singh and D. S. Kapoor, “Create Your Own Internet of Things: A survey of IoT platforms,” *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 57-68, 2017. doi: 10.1109/MCE.2016.2640718.
- [4] F. Karray, M. W. Jmal, A. Garcia-Ortiz, M. Abid, and A. M. Obeid, “A comprehensive survey on wireless sensor node hardware platforms,” *Computer Networks*, vol. 144, pp. 89-110, 2018, doi: 10.1016/j.comnet.2018.05.010.
- [5] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, “A Review of Low-End, Middle-End, and High-End IoT Devices,” *IEEE Access*, vol. 6, pp. 70528-70554, 2018, doi: 10.1109/ACCESS.2018.2879615.
- [6] A. Polianytsia, O. Starkova, and K. Herasymenko, “Survey of hardware IoT platforms,” *2016 3rd International Scientific-Practical Conference Problems of Infocommunications Science and Technology, PIC S and T 2016 - Proceedings*, pp. 152-153, Apr. 2017, doi: 10.1109/INFOCOMMST.2016.7905364.
- [7] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking, “A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT),” *2015 Internet Technologies and Applications, ITA 2015 - Proceedings of the 6th International Conference*, pp. 219-224, Nov. 2015, doi: 10.1109/ITECHA.2015.7317398.
- [8] “Why it is called Internet of Things: Definition, history, disambiguation.” <https://iot-analytics.com/internet-of-things-definition/> (accessed Jan. 10, 2022).
- [9] A. Floris and L. Atzori, “Managing the Quality of Experience in the Multimedia Internet of Things: A layered-based approach,” *Sensors (Switzerland)*, vol. 16, no. 12, Dec. 2016, doi: 10.3390/S16122057.
- [10] “GSMA Intelligence - Source Of Mobile Industry Insights.” <https://www.gsmainelligence.com/> (accessed Jan. 20, 2022).

- [11] E. Power Research Insitute, "CONFIGURATION AND DEMONSTRATION OF 433-MHZ WIRELESS GAUGE READER," 2012.
- [12] "Wireless Gauge Reader from Honeywell | Fierce Electronics."
<https://www.fierceelectronics.com/components/wireless-gauge-reader-from-honeywell> (accessed Mar. 02, 2022).
- [13] H. Dietz and P. Eberhart, "An ultra-low-cost large-format wireless IoT camera," in *IS and T International Symposium on Electronic Imaging Science and Technology*, 2021, vol. 2021, no. 7. doi: 10.2352/ISSN.2470-1173.2021.7.ISS-070.
- [14] P. Rai and M. Rehman, "ESP32 based smart surveillance system," 2019. doi: 10.1109/ICOMET.2019.8673463.
- [15] E. Gatial, Z. Balogh, and L. Hluchy, "Concept of Energy Efficient ESP32 Chip for Industrial Wireless Sensor Network," 2020. doi: 10.1109/INES49302.2020.9147189.
- [16] C. G. C. Carducci, A. Monti, M. H. Schraven, M. Schumacher, and D. Mueller, "Enabling ESP32-based IoT Applications in Building Automation Systems," 2019. doi: 10.1109/METROI4.2019.8792852.
- [17] P. Foltýnek, M. Babiuch, and P. Šuránek, "Measurement and data processing from Internet of Things modules by dual-core application using ESP32 board," *Measurement and Control (United Kingdom)*, vol. 52, no. 7-8, pp. 970-984, Sep. 2019, doi: 10.1177/0020294019857748.
- [18] "ESP32 based Industrial Controllers - NORVI Industrial Arduino."
<https://norvi.lk/esp32-based-industrial-controllers/> (accessed Dec. 21, 2021).
- [19] "Moduino X Series - Industrial IoT Controller based on ESP32."
<https://moduino.techbase.eu/> (accessed Dec. 21, 2021).
- [20] "Calculating Camera Sensor Resolution and Lens Focal Length - NI."
<https://www.ni.com/pt-pt/support/documentation/supplemental/18/calculating-camera-sensor-resolution-and-lens-focal-length.html> (accessed Jan. 22, 2022).
- [21] "What Is Camera Calibration? - MATLAB & Simulink."
<https://www.mathworks.com/help/vision/ug/camera-calibration.html> (accessed Jan. 24, 2022).
- [22] "OpenCV: Camera Calibration."
https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html (accessed Jan. 17, 2022).
- [23] C. Amsler *et al.*, "Review of Particle Physics*," *Physics Letters B*, vol. 667, no. 1-5, p. 1, Sep. 2008, doi: 10.1016/j.physletb.2008.07.018.
- [24] S. Zhou, T. Zhu, K. Shi, Y. Li, W. Zheng, and J. Yong, "Review of light field technologies," *Visual Computing for Industry, Biomedicine, and Art*, vol. 4, no. 1, Dec. 2021, doi: 10.1186/S42492-021-00096-8.

- [25] "A Practical Guide to Machine Vision Lighting - NI." <https://www.ni.com/pt-pt/innovations/white-papers/12/a-practical-guide-to-machine-vision-lighting.html> (accessed Feb. 24, 2022).
- [26] J.-J. Kim, S. Chae, and K.-H. Jeong, "Micropatterned single lens for wide-angle light-emitting diodes," *Optics Letters*, vol. 35, no. 6, 2010, doi: 10.1364/ol.35.000823.
- [27] B. Sun, J. Fan, X. Fan, G. Zhang, and G. Zhang, "A reliability prediction methodology for LED arrays," *IEEE Access*, vol. 7, pp. 8127-8134, 2019, doi: 10.1109/ACCESS.2018.2887252.
- [28] C. C. Sun *et al.*, "Illuminance and starting distance of the far field of led-array luminaire operated at short working distance," *Crystals*, vol. 10, no. 5, May 2020, doi: 10.3390/cryst10050360.
- [29] I. Moreno, C. C. Sun, and R. Ivanov, "Far-field condition for light-emitting diode arrays," *Applied Optics*, vol. 48, no. 6, pp. 1190-1197, Feb. 2009, doi: 10.1364/AO.48.001190.
- [30] "Bright Field and Dark Field Lighting." <https://www.microscan.com/en-us/resources/know-your-tech/bright-field-and-dark-field-lighting> (accessed Feb. 24, 2022).
- [31] D. Martin, M. Sales, and S. Manager, "A Practical Guide to Machine Vision Lighting," 2013.
- [32] Z. Wang, Q. Xing, L. Fu, and H. Sun, "Realtime vision-based surface defect inspection of steel balls," *Transactions of Tianjin University*, vol. 21, no. 1, 2015, doi: 10.1007/s12209-015-2452-6.
- [33] "Flat dome illumination | STEMMER IMAGING." <https://www.stemmer-imaging.com/en/knowledge-base/flat-dome-illumination/> (accessed Dec. 21, 2021).
- [34] H. Cho, S. Lee, and B. Lee, "Optical design of photometric techniques for specular reflection," *Optics and Lasers in Engineering*, vol. 151, 2022, doi: 10.1016/j.optlaseng.2021.106882.
- [35] "Dome illumination | STEMMER IMAGING." <https://www.stemmer-imaging.com/en/knowledge-base/dome-illumination/> (accessed Dec. 21, 2021).
- [36] "Bright Field vs. Dark Field Lighting Techniques | Ai." <https://www.advancedillumination.com/lighting-education/bright-field-dark-field-lighting/> (accessed Jan. 20, 2022).
- [37] Z. M. Zhu, X. L. Jin, H. Yang, and L. S. Zhong, "Design of diffuse reflection freeform surface for uniform illumination," *IEEE/OSA Journal of Display Technology*, vol. 10, no. 1, 2014, doi: 10.1109/JDT.2013.2273073.
- [38] Z. Zhu, X. Xu, X. Sun, and X. Wang, "Establishment and Solution for a Mathematical Model of Multiple Diffuse Reflection," *IEEE Photonics Journal*, vol. 12, no. 4, 2020, doi: 10.1109/JPHOT.2020.3003543.

- [39] “Using Polarizing Filters Techniques | Machine Vision.” <https://www.advancedillumination.com/lighting-education/using-polarizing-filters-in-machine-vision/> (accessed Dec. 21, 2021).
- [40] E. Ferro and F. Potorti, “Bluetooth and Wi-Fi wireless protocols: A survey and a comparison,” *IEEE Wireless Communications*, vol. 12, no. 1, pp. 12-26, 2005, doi: 10.1109/MWC.2005.1404569.
- [41] Y. Chen and T. Kunz, “Performance evaluation of IoT protocols under a constrained wireless access network,” 2016. doi: 10.1109/MoWNet.2016.7496622.
- [42] T. Moraes, B. Nogueira, V. Lira, and E. Tavares, “Performance comparison of iot communication protocols,” in *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 2019, vol. 2019-Octob, pp. 3249-3254. doi: 10.1109/SMC.2019.8914552.
- [43] “Hypertext Transfer Protocol Version 3 (HTTP/3).” <https://quicwg.org/base-drafts/draft-ietf-quic-http.html> (accessed Feb. 20, 2022).
- [44] “MQTT Version 5.0.” <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> (accessed Jan. 04, 2022).
- [45] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” 2014. Accessed: Jan. 04, 2022. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc7252>
- [46] “Serviços de Computação na Cloud | Microsoft Azure.” <https://azure.microsoft.com/pt-pt/> (accessed Mar. 02, 2022).
- [47] “O que é cloud computing e como pode beneficiar a sua empresa.” <https://www.compuworks.pt/o-que-e-cloud-computing-e-como-pode-beneficiar-a-sua-empresa/> (accessed Mar. 02, 2022).
- [48] “Cloud IoT Core | Google Cloud.” <https://cloud.google.com/iot-core> (accessed Feb. 20, 2022).
- [49] “AWS Documentation.” https://docs.aws.amazon.com/index.html?nc2=h_ql_doc_do (accessed Feb. 20, 2022).
- [50] “Serviços de Computação na Cloud | Microsoft Azure.” <https://azure.microsoft.com/pt-pt/> (accessed Feb. 20, 2022).
- [51] P. Chakravorty, “What is a Signal? [Lecture Notes],” *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 175-177, Sep. 2018, doi: 10.1109/MSP.2018.2832195.
- [52] “Numpy and Scipy Documentation – Numpy and Scipy documentation.” <https://docs.scipy.org/doc/> (accessed Mar. 02, 2022).
- [53] “Python Pillow.” <https://python-pillow.org/> (accessed Mar. 02, 2022).
- [54] “Home - OpenCV.” <https://opencv.org/> (accessed Mar. 02, 2022).
- [55] V. Kucheruk, I. Kurytnik, P. Kulakov, R. Lishchuk, Y. Moskvichova, and A. Kulakova, “Definition of dynamic characteristics of pointer measuring devices on the basis of

- automatic indications determination,” *Archives of Control Sciences*, vol. 28, no. 3, pp. 401-418, 2018, doi: 10.24425/acs.2018.124709.
- [56] C. Zheng, S. Wang, Y. Zhang, P. Zhang, and Y. Zhao, “A robust and automatic recognition system of analog instruments in power system by using computer vision,” *Measurement: Journal of the International Measurement Confederation*, vol. 92, pp. 413-420, 2016, doi: 10.1016/j.measurement.2016.06.045.
- [57] P. E. C. de Sousa and A. A. V. T. Sousa, “Leitura e calibração de manómetros mecânicos através da visão por computador em linguagem Python,” 2018, Accessed: Jan. 05, 2022. [Online]. Available: <https://recipp.ipp.pt/handle/10400.22/14349>
- [58] J. Chi, L. Liu, J. Liu, Z. Jiang, and G. Zhang, “Machine Vision Based Automatic Detection Method of Indicating Values of a Pointer Gauge,” *Mathematical Problems in Engineering*, vol. 2015, 2015, doi: 10.1155/2015/283629.
- [59] B. Yang, G. Lin, and W. Zhang, “Auto-recognition Method for Pointer-type Meter Based on Binocular Vision,” 2014, doi: 10.4304/jcp.9.4.787-793.
- [60] S. Zhao, B. Li, X. Jia, J. Yuan, and G. Yue, “Optimized Meter Calibration Method Based on Auto Recognition,” 2004.
- [61] X. Ye, D. Xie, and S. Tao, “Automatic Value Identification of Pointer-Type Pressure Gauge Based on Machine Vision,” 2013, doi: 10.4304/jcp.8.5.1309-1314.
- [62] Y. wei, H. Jun da, and Z. Wen bin, “An auto-recognition system for pointer-meter based on TLS analytic resolution,” 2009, doi: 10.1109/ICNDS.2009.101.
- [63] C. Li, D. Zheng, L. Liu, and X. Zheng, “A UAV-Based Machine Vision Algorithm for Industrial Gauge Detecting and Display Reading,” *2020 5th Asia-Pacific Conference on Intelligent Robot Systems, ACIRS 2020*, pp. 109-115, Jul. 2020, doi: 10.1109/ACIRS49895.2020.9162618.
- [64] Y. Liu, J. Liu, and Y. Ke, “A detection and recognition system of pointer meters in substations based on computer vision,” *Measurement*, vol. 152, p. 107333, Feb. 2020, doi: 10.1016/J.MEASUREMENT.2019.107333.
- [65] L. Zuo, P. He, C. Zhang, and Z. Zhang, “A robust approach to reading recognition of pointer meters based on improved mask-RCNN,” *Neurocomputing*, vol. 388, pp. 90-101, May 2020, doi: 10.1016/J.NEUCOM.2020.01.032.
- [66] “Tesseract documentation | Tesseract OCR.” <https://tesseract-ocr.github.io/> (accessed Feb. 17, 2022).
- [67] “Wi-Fi & Bluetooth MCUs and AIoT Solutions | Espressif Systems.” <https://www.espressif.com/> (accessed Jan. 03, 2022).
- [68] “ESP32-CAM and Other Cool Projects on RNT | Espressif Systems.” https://www.espressif.com/en/news/ESP32_CAM (accessed Jan. 03, 2022).

- [69] “ESP32-CAM WiFi and Bluetooth Module w/ Camera OV2640 2MP | Sensor C...”
<https://www.ptrobotics.com/sensor-camera/7112-esp32-cam-wifi-and-bluetooth-module-w-camera-ov2640-2mp.html> (accessed Mar. 03, 2022).
- [70] “Sleep Modes - ESP32 - — ESP-IDF Programming Guide latest documentation.”
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html (accessed Jan. 03, 2022).
- [71] “In-Depth: ESP32 Deep Sleep & Wakeup Sources | Timer, Touch & External.”
<https://lastminuteengineers.com/esp32-deep-sleep-wakeup-sources/> (accessed Jan. 03, 2022).
- [72] “OV2640 Datasheet | OmniVision Technologies - Datasheetspdf.com.”
<https://datasheetspdf.com/datasheet/OV2640.html> (accessed Mar. 03, 2022).
- [73] “Welcome To Colaboratory - Colaboratory.” <https://colab.research.google.com/> (accessed Mar. 02, 2022).
- [74] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330-1334, Nov. 2000, doi: 10.1109/34.888718.
- [75] “OpenCV: Color conversions.”
https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html (accessed Jan. 17, 2022).
- [76] “OpenCV: Image Thresholding.”
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html (accessed Mar. 05, 2022).
- [77] “scipy.ndimage.sobel — SciPy v1.8.0 Manual.”
<https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.sobel.html> (accessed Feb. 01, 2022).
- [78] “OpenCV: Hough Line Transform.”
https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html (accessed Feb. 02, 2022).

Annex A - LED diffuser characteristic graph (luminous intensity function of angle)

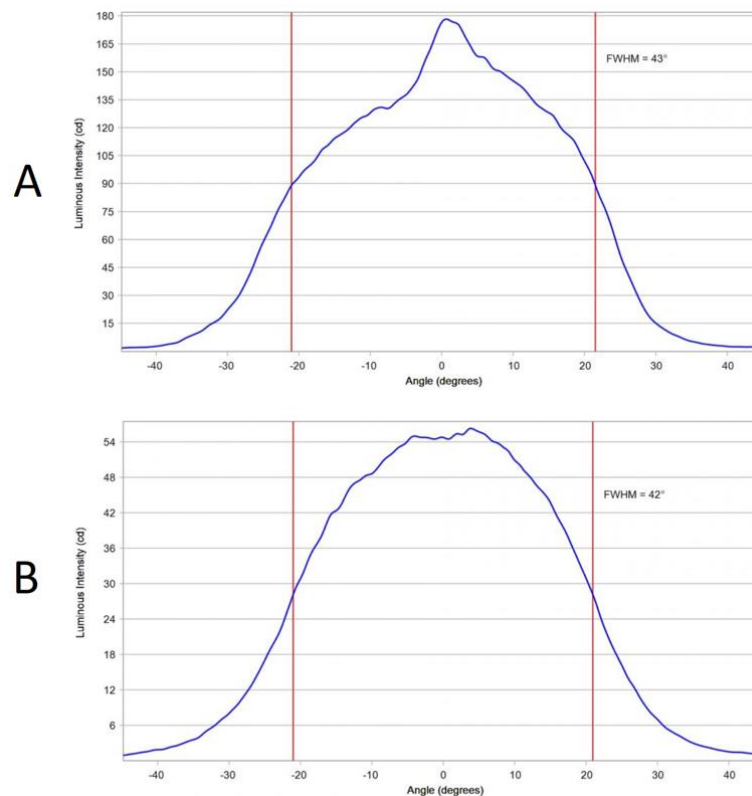


Figure 6.1 - LED diffuser characteristic's graph - luminous intensity function of angle

The graph of the LED luminous intensity as a function of the angle to the LED position, for each diffuser, is shown here. The graph shown with the letter A refers to the led diffuser presented in Figure 3.8 c1), while the graph with the letter B refers to the led diffuser shown in Figure 3.8 b1).

Annex B - Image calibration algorithm

```
1. import cv2 as cv
2. import numpy as np #vetores, matemática, etc
3. import glob
4. import matplotlib.pyplot as plt
5.
6.
7. #Montar o google drive na pasta "/content/drive"
8. from google.colab import drive
9. drive.mount('/content/drive')
10.
11. mostrarimagem=False
12.
13.
14. i=0
15. # Criterio finalização
16. criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
17.
18. # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
19. objp = np.zeros((6*9,3), np.float32)
20. objp[:, :2] = np.mgrid[0:9,0:6].T.reshape(-1,2)
21.
22. # Arrays to store object points and image points from all the images.
23. objpoints = [] # 3d point in real world space
24. imgpoints = [] # 2d points in image plane.
25. images = glob.glob('/content/drive/My Drive/Tese_Nuno_2021/SVGA.800/*.jpg')
26.
27. for fname in images:
28.     img = cv.imread(fname)
29.     gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
30.
31.     # Find the chess board corners
32.     ret, corners = cv.findChessboardCorners(gray, (9,6), None)
33.
34.     # If found, add object points, image points (after refining them)
35.     #Show the image with matplotlib
36.     if ret == True:
37.         objpoints.append(objp)
38.         corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
39.         imgpoints.append(corners)
40.
41.         # Draw and display the corners
42.         cv.drawChessboardCorners(img, (9,6), corners2, ret)
43.         if mostrarimagem:
44.             plt.figure(figsize=(40,10))
45.             plt.subplot(2,5,i+1)
46.             plt.imshow(img, cmap='gray')
47.             i=i+1
48.         plt.show()
```

```
49.  
50. ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,  
    gray.shape[:-1], None, None)  
51.  
52.  
53. #Guarda variáveis em formato .npy no Google Drive  
54. np.save('/content/drive/My Drive/Tese_Nuno_2021/SVGA.800/calibracao_mtx.npy',  
    mtx)  
55. np.save('/content/drive/My Drive/Tese_Nuno_2021/SVGA.800/calibracao_dist.npy',  
    dist)
```

Annex C - Scale limits detection's scan

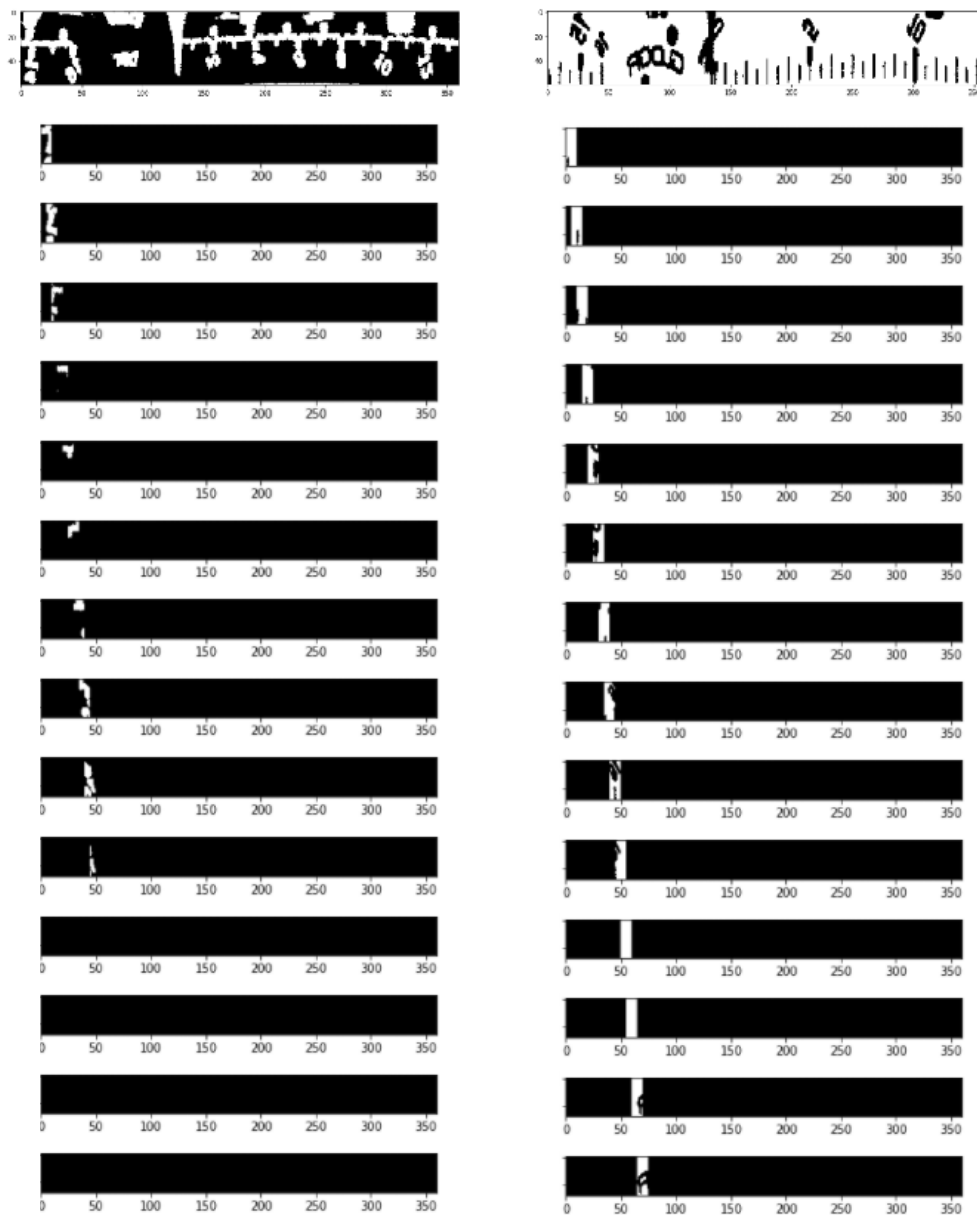


Figure 6.2 - Illustration of the 15 first scans performed on the image after polar transform

Annex D - Analog dial reader algorithm """

```
"""
Created on November 10 2021

@author: Nuno Gonçalo Vitória Bartolomeu
"""

#montar o google drive na pasta "/content/drive"
1 from google.colab import drive
drive.mount('/content/drive')

12
#Bibliotecas necessárias
14 import numpy
import cv2 as cv
15 import cv2 #openCV para imagens
import numpy as np #vetores, matemática, etc
16 from matplotlib import pyplot as plt #mostrar gráficos/imagens
import os
17 from google.colab.patches import cv2_imshow
from scipy.ndimage import filters as filter2
18 from skimage import filters, feature, morphology
from skimage.transform import import (rescale,hough_line,
19 probabilistic_hough_line, hough_circle)
import matplotlib.pyplot as plt
20 from matplotlib.figure import Figure
from pylab import *
21 import math
from math import atan2, pi #calculo e valor de pi
22 from PIL import Image
import tempfile
23 import pytesseract
import re
24
#Inicialização de variáveis
25 #####
# Mostra imagem resultante caso seja True
26 mostrarimagem=False
# Mostra resultados caso seja True
27 mostrarresultados = False
# Mostra imagens do scan feito á escala caso seja True
28 mostraescala = False
# Inserir valor de inicio de escala caso não seja feita a deteção automática
correta da simetria e dos valores de escala
29 val_in = None
# Inserir valor de fim de escala caso não seja feita a deteção automática
correta da simetria e dos valores de escala
30 val_fim = None
# Margem a aplicar á ROI para aperfeicoamento do resultado
31 margem_ROI = 40
```

```

# Variar este valor caso o centro da imagem não seja detetado corretamente.
# Rácio entre a resolução da imagem original para a matriz do acumulador.
# (valores maiores que zero) (conselho -> começar do 1 para cima)
50 ratio_resolution_CENTRO = 10
# Número mínimo de votos (intercepções nas células da rede de Hough).
# Aumentando o valor torna mais exigente a detecção, o que diminui o número de
# linhas detetadas
52 threshold_AGULHA = 83
# Variar este valor permite deslocar a faixa da imagem a fazer a transformada
# polar. Valor positivos deslocam a imagem do centro para a periferia. Valor
# negativos deslocam a imagem da periferia para o centro.
54 offset_escala_IMG2POLAR = 0
# Variar este valor permite deslocar o início de escala detetado, em graus.
# Valor positivos deslocam a posição detetada no sentido horário, valor
# negativos deslocam a posição detetada no sentido anti-horário
56 offset_inicio_ESCALA = 0
# Dimensões imagem original
58 h, w = img.shape[:2]
# Não alterar!
60 thresh_agulha_determinada = None

62
#Upload das variáveis de calibração previamente guardadas
64 mtx = np.load('/content/drive/My
Drive/Tese_Nuno_2021/SVGA.800/calibracao_mtx.npy')
dist = np.load('/content/drive/My
Drive/Tese_Nuno_2021/SVGA.800/calibracao_dist.npy')
66
def calibrar (img, mtx, dist, mostrarimagem=False):
68     """
    Calibração da imagem original.
    """
70     newcameramtx, roi = cv.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))
72
# Correção da distorção
74     mapx, mapy = cv.initUndistortRectifyMap(mtx, dist, None, newcameramtx,
(w,h), 5)
76
    dst = cv.remap(img, mapx, mapy, cv.INTER_LINEAR)
78
# Mostra as imagens resultantes
80     if mostrarimagem:
        plt.figure(figsize=(20,20))
        u1 = "Distorted image"
        plt.subplot(2,2,1).set_title(u1)
        plt.imshow(img, cmap='gray')
        u2 = "Undistorted image"
        plt.subplot(2,2,2).set_title(u2)
        plt.imshow(dst, cmap='gray')
82
    return dst
84
86
## Escala de cinzentos + blur da imagem + binarização com método OTSU
#####
88
def RBGTOBIN (img, mostrarimagem):
89     """
    Tratamento inicial da imagem.
    Conversão para escala de cinzento, "blur" da imagem, seguido de uma
    binarização segundo o método de OTSU
    """
91
93
#Inicialização de variáveis da função "RBGTOBIN"
95
#####

```

```

104 # Valor threshold do output
105 thresholding_value = 140
106 # Aqui é calculado o valor médio da intensidade de pixel em escala de
107 # cinzentos de cada sub-região 101×101 na imagem para calcular o nosso valor
108 # limiar.
109 pixel_neighborhood_size = 101
110 # Este valor permite-nos afinar o valor limiar.
111 constant_c_adjustment = 2
112 #####
113
114 # Converter para escala de cinzento.
115 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
116
117 # Blur usando 3 * 3 kernel.
118 gray_blurred = cv2.blur(gray, (3, 3))
119
120 # OTSU GAUSSIAN sobre a imagem "gray_blurred".
121 imgth = cv2.adaptiveThreshold(gray_blurred, thresholding_value,
122                               cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
123                               cv2.THRESH_BINARY,
124                               pixel_neighborhood_size, constant_c_adjustment)
125
126 if mostrarimagem:
127     plt.figure(figsize=(40,10))
128     u1 = "Grayscale"
129     plt.subplot(2,3,1).set_title(u1)
130     plt.imshow(gray, cmap='gray')
131     u2 = "Grayscale + Blur"
132     plt.subplot(2,3,2).set_title(u2)
133     plt.imshow(gray_blurred, cmap='gray')
134     u3 = "Grayscale + Blur + Otsu Binarization"
135     plt.subplot(2,3,3).set_title(u3)
136     plt.imshow(imgth, cmap='gray')
137
138 return imgth
139
140
141 ## Determinacao da ROI
142 #####
143
144 def ROI(img, margem_ROI, mostrarimagem):
145     """
146     Função para detecção da ROI, exclui informação da imagem que não seja
147     necessária para posterior processamento.
148     """
149
150     #Inicialização de variáveis da função "ROI"
151     #####
152     # Ajuste para obter melhores resultados sem ser necessário alterar variáveis
153     ajuste = 85/90
154     # Valor para gerar raios desde o raio mínimo ao raio máximo de 10 em 10
155     pesquisa_circ = 5
156     #####
157
158     # Comprimento máximo para o raio da ROI, em pixeis. (raio máximo para
159     # procurar)(h = altura imagem, em pixeis)
160     comprMax=(h/2)*ajuste
161
162     # raio mínimo para procurar = metade do raio máximo
163     comprmin=comprMax/2
164
165     # lista de raios a pesquisar na Transformada de Hough
166     raios = np.arange(comprmin, comprMax, pesquisa_circ)
167
168     # Aplicar Filtro de Sobel á imagem
169     imgsobel=filters.sobel(img)
170

```

```

# Pesquisar círculos usando Hough transform
154 res = hough_circle(imgsobel, raios)

156 # Resulta o melhor resultado dos vários círculos detetados, escolhendo o que
    tiver o valor mais elevado
    indRaio, xroi, yroi = np.unravel_index(np.argmax(res), res.shape)

158 # Aplicar uma margem para obtenção de um melhor resultado
159 raio=raios[indRaio]- margem_ROI

161 # Recriar a máscara igual ao círculo
162 x, y = np.indices(shape(img))

164 # Definir a máscara
165 mask = (x - xroi)**2 + (y - yroi)**2 < raio**2

167 # Imagem resultante da ROI
168 imgroi=mask*img

170 imgtest = cv2.circle(mask*img, (yroi,xroi), radius=0, color=(255, 0, 0),
171 thickness=5)

173 if mostrarimagem:
174     plt.figure(figsize=(40,20))
175     plt.subplot(1,2,1)
176     plt.imshow(imgroi, cmap='gray')
177     plt.subplot(1,2,2)
178     plt.imshow(imgtest, cmap='gray')
179     print("Raio =", raio)

181 return imgroi,raio, (yroi,xroi), imgtest
182
184
186 ## Determinar centro do manómetro
187 #####
189 def CENTRO(img, ratio_resolution_CENTRO, mostrarimagem):
190     """
191     Aqui é feita a determinação do centro do manómetro, com recurso á função do
    OpenCV (cv2.HoughCircles)
    Dos círculos gerados é escolhido aquele que tiver mais próximo do centro
    real da imagem (400x300)
192     O centro do círculo escolhido é guardado na variável "centro"
    """
194
196     #Inicialização de variáveis da função
197     #####
198     # This parameter controls the minimum distance between detected circles.
199     circ_dist=1
200     # Canny edge detection requires two parameters – minVal and maxVal.
    canny_param is the higher threshold of the two. The second one is set as
    Param1/2.
201     canny_param=300
202     # This is the accumulator threshold for the candidate detected circles. By
    increasing this threshold value, we can ensure that only the best circles,
    corresponding to larger accumulator values, are returned.
203     circ_threshold = 100
204     # Raio mínimo a detetar em pixels
205     min_radius = 5
206     # Raio máximo a detetar em pixels
207     max_radius = 45
208     # Metade da largura da imagem
209     xreal = w/2
210     # Metade da altura da imagem
211     yreal = h/2

```

```

# Inicialização variável i
232 i=0
# Inicialização da distancia a calcular usando um valor elevado = 500
234 d=500
#####

# Corre função transformada de Hough para círculos com os seguintes
parametros: Guarda as coordenadas do centro e respetivo raio na variável
"detected_circles"
238 detected_circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT,
ratio_resolution_CENTRO, circ_dist, param1=canny_param, param2=circ_threshold,
minRadius=min_radius, maxRadius=max_radius)

240 # Criar nova imagem para desenhar círculos
circ = np.copy(img)

242 if detected_circles is not None:
244     detected_circles = np.uint16(np.around(detected_circles))
count=len(detected_circles)

246 # Guarda os círculos detetados e acrescenta uma variável na matrix
correspondente a d (distancia do círculo detetado ao centro real da imagem)
248 detected_circles_nova = np.insert(detected_circles, 3, d, axis=2)

250 for pt in detected_circles[0, :]:
    a, b, r = pt[0], pt[1], pt[2]

252 # Determina o valor de d (distancia do círculo detetado ao centro real
da imagem)
254 d = (((xreal - a)**2) + (yreal-b)**2)**0.5

256 # Coloca o valor da distancia calculada na matrix
"detected_circles_nova"
258 detected_circles_nova[0,i,3] = d
i+=1

260 # Escolhe o círculo que tiver a menor distancia "d" calculada
posicao =
np.nonzero(detected_circles_nova[:,3]==np.min(detected_circles_nova[:,3]))

262 # Guarda a posição do centro do círculo escolhido na variável "centro"
centro = detected_circles_nova[0, posicao[1][0]]
a = centro[0]
264 b = centro[1]
r = centro[2]

266 # Desenha círculo detetado na imagem "circ"
268 cv2.circle(circ, (a, b), r, (255, 0, 0), 2)
cv2.circle(circ, (a, b), 1, (255, 0, 0), 3)
270 # Desenha o centro real da imagem ((w/2), (h/2))
cv2.circle(circ, (int(xreal), int(yreal)), 1, (255, 255, 255), 2)

272 if mostrarimagem:
274     plt.figure(figsize=(40,10))
u1 = "Center Detection Illustration"
276 plt.subplot(1,2,1).set_title(u1)
plt.imshow(circ, cmap='gray')
278 print('Centro =', (a, b))

280 else:
    a = 0
    b = 0
    circ = img

282 return (a, b), circ
284

```

```

280
281 ## Corta a imagem no limite da ROI detetada anteriormente
282 #####
283 ###
284
285 def CORTE(img, raio, centro, mostrarimagem):
286     """
287     Função para corte da imagem fornecida, sabendo o centro calculado da função
288     anterior e limites igual ao raio calculado na ROI
289     """
290
291     start_point = (int(centro[0]-raio), int(centro[1]-raio))
292     end_point = (int(centro[0]+raio), int(centro[1]+raio))
293
294     crop_img = img[start_point[1]:end_point[1], start_point[0]:end_point[0]]
295     a = crop_img.shape[0]/2
296     b = crop_img.shape[0]/2
297
298     if mostrarimagem:
299         plt.figure(figsize=(10,10))
300         u1 = "Cropped Image"
301         plt.subplot(1,2,1).set_title(u1)
302         plt.imshow(crop_img, cmap='gray')
303
304     return crop_img, (a, b)
305
306
307 ## Determinar retas limites da agulha e respetiva intercepção
308 #####
309 ###
310
311 def AGULHA(img, mostrarimagem):
312     """
313     Função para a determinação dos limites que formam o bordo da agulha, com
314     recurso á função do OpenCV (cv2.HoughLinesP)
315     Quando são detetadas duas retas é calculado o ponto resultante da sua
316     intercepção.
317     Na primeira vez que corre para um dado manómetro testa diferentes valores de
318     threshold de linhas para a transformada de Hough e guarda aquele que gerar
319     duas retas.
320     Nas passagens posteriores á primeira, determina as linhas da agulha com base
321     no melhor valor do threshold determinado previamente, caso volte a detetar
322     mais retas, testa novamente diferentes valores de threshold.
323     O ponto resultante da intercepção é guardado na variável "ponto_ag"
324     """
325
326     #Inicialização de variáveis da função
327     #####
328     # Valor mínimo do threshold para método de CANNY
329     low_threshold = 50
330     # Valor máximo do threshold para método de CANNY
331     high_threshold = 150
332     # distance resolution in pixels of the Hough grid
333     dist_resolution = 1
334     # angular resolution in radians of the Hough grid
335     ang_resolution = np.pi / 180
336     # minimum number of pixels making up a line
337     min_line_length = 100
338     # maximum gap in pixels between connectable line segments
339     max_line_gap = 40
340
341
342
343
344

```

```

#####
337
    # Determinar interseção das 2 linhas da agulha
338 def seg_intersect(a1,a2, b1,b2) :
    """
339     Função para cálculo de interseção de 2 retas definidas por 2 pontos cada.
    """
340     da = a2-a1
341     db = b2-b1
342     dp = a1-b1
343
344     dap = empty_like(da)
345     dap[0] = -da[1]
346     dap[1] = da[0]
347
348     denom = dot( dap, db)
349     num = dot( dap, dp )
350     return (num / denom.astype(float))*db + b1
351
352 #Aplicar o método de CANNY.
353 edges = cv2.Canny(img, low_threshold, high_threshold)
354
355 #Criar imagem para desenhar linhas da agulha
356 line_image = np.copy(img) * 0
357
358 # Lista de threshold a usar na Transformada de Hough da função "AGULHA"
359 threshold_automatico_AGULHA = np.arange(30, 150, 5)
360
361 for i in range(0, len(threshold_automatico_AGULHA), 1):
362     # Corre a Transformada de Hough com os valores gerados na lista
363     "threshold_automatico_AGULHA"
364     lines = cv2.HoughLinesP(edges, dist_resolution, ang_resolution,
365                             threshold_automatico_AGULHA[i], np.array([]), min_line_length, max_line_gap)
366
367     # Verifica se foram detetadas retas com a Transformada de Hough
368     if lines is None:
369         nr_linhas = 0
370     else:
371         nr_linhas = lines.shape[0]
372
373     # Corre os valores da lista "threshold_automatico_AGULHA" até que sejam
374     encontradas apenas duas retas
375     if nr_linhas > 2:
376         pass
377     if nr_linhas < 2:
378         pass #
379     if nr_linhas == 2:
380         melhor_threshold = threshold_automatico_AGULHA[i]
381         break
382
383 # Corre Hough edge detection onde resulta o output "lines" com os respetivos
384 segmentos de reta detetados, com o valor de threshold ideal, calculado
385 previamente
386 lines = cv2.HoughLinesP(edges, dist_resolution, ang_resolution,
387                         melhor_threshold, np.array([]), min_line_length, max_line_gap)
388 for line in lines:
389     for x1,y1,x2,y2 in line:
390         # Desenha as linhas detetadas com a transformada de Hough
391         cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0), 2)
392         img_line_edges = cv2.addWeighted(img, 0.8, line_image, 1, 0)
393
394 # Determinar interseção das 2 linhas da agulha e o ponto resultante na
395 variável "ponto0"
396 ponto0 = seg_intersect( lines[0,0,0:2],lines[0,0,2:4],
397                         lines[1,0,0:2],lines[1,0,2:4])
398

```

```

# Conversão do resultado para float32
395 ponto_ag = numpy.float32(ponto0)

397 # Desenha o ponto de intercepção detetado
image2 = cv2.circle(img_line_edges, tuple(ponto_ag), radius=0, color=(255,
0, 0), thickness=5)

399
if mostrarimagem:
401     plt.figure(figsize=(10,10))
    u1 = "Dial Detection"
403     plt.subplot(1,2,1).set_title(u1)
    plt.imshow(img_line_edges, cmap='gray')
405     print('Número de linhas detetadas = ', nr_linhas)
    print('Ponto intercepção agulha = ', ponto_ag)
407

409 return ponto_ag, img_line_edges, nr_linhas, melhor_threshold

411
## Converte a imagem para coordenadas Polares
412 #####
413 ###
414
415 def IMG2POLAR (img, raio, centro, offset_escala_IMG2POLAR, mostrarimagem):
    """
417     Função para conversão da imagem para coordenadas polares.
    Resulta daqui uma imagem retangular com altura (eixo das ordenadas) de 35
    pixels escalada de 0 a 360 no eixo das abcissas
    """
419
    #Inicialização de variáveis da função
421     #####
    # Raio exterior da imagem a transformar
423     final_radius = int(raio) -10 + offset_escala_IMG2POLAR
    print(final_radius)
425     # Raio interior da imagem a transformar
    initial_radius = int(raio) - 45 + offset_escala_IMG2POLAR
427     print(initial_radius)
    # Escalar imagem de 0 a 360 (eixo das abcissas em graus)
429     amplit_eixo_xx = 360
    #####
431
432     def polar2cart(r, theta, center):
        """
434         Função para conversão de cordenadas polares para cartesianas.
        """
436
        x = r * np.cos(theta) + center[0]
437         y = r * np.sin(theta) + center[1]
        return x, y
439
440     def img2polar(img, center, final_radius, initial_radius, amplit_eixo_xx):
        """
442         Função para conversão da imagem para coordenadas polares.
        """
443
        theta , R = np.meshgrid(np.linspace(0, 2*np.pi, amplit_eixo_xx),
445                                np.arange(initial_radius, final_radius))
447
        Xcart, Ycart = polar2cart(R, theta, center)
449
        Xcart = Xcart.astype(int)
        Ycart = Ycart.astype(int)
451
        polar_img = img[Ycart,Xcart]
453         polar_img = polar_img.reshape(polar_img,(final_radius-
initial_radius,amplit_eixo_xx))
455
        return polar_img

```

```

457 polar = img2polar(img, centro, final_radius, initial_radius, amplit_eixo_xx)
459
460 if mostrarimagem:
461     plt.figure(figsize=(40,10))
462     u1 = "Polar coordinates Image"
463     plt.subplot(1,1,1).set_title(u1)
464     plt.imshow(polar, cmap='gray')
465
466 return polar
467
468
469 ## Calcular início e fim de escala
470 #####
471 ###
472
473 def ESCALA (img, img2, centro, offset_inicio_ESCALA, mostrarimagem,
474            mostraescala):
475     """
476     Função para detecção de onde começa e acaba a escala.
477     Resulta daqui dois pontos que definem o início e fim da escala.
478     """
479
480     #Inicialização de variáveis da função
481     #####
482     # Inicializar variável incremental "i"
483     i = 0
484     # Inicializar variável "x" que representa o avanço do SCAN em graus
485     x = 0
486     # Largura do retângulo que faz o scan
487     w_scan = 10
488     # Altura do retângulo que faz o scan
489     h_scan = 35
490     # Percentagem de pixels admissíveis para detetar o início e fim de escala
491     percent = 97/100
492     # Criar matrix onde serão guardadas os resultados do SCAN
493     resultados_escala = []
494     #####
495
496     # Threshold de pixels para manómetros de fundo branco com escala em
497     preto
498     valor = (h_scan*w_scan)*percent
499
500     # Threshold de pixels para manómetros de fundo preto com escala em branco
501     invalor = (h_scan*w_scan) - ((h_scan*w_scan)*percent)
502
503     # Número de pixels com valor diferente de 0 na imagem "img"
504     pixels = cv2.countNonZero(img)
505
506     for i in range(0, 360, 5):
507         # Criar máscara com as dimensões pré-definidas para realizar o scan da
508         imagem
509         mask = np.zeros(img.shape,np.uint8)
510         mask[0:h_scan,x:x+w_scan] = img[0:h_scan,x:x+w_scan]
511         x=x+5
512
513         # Conta o número de pixels relativos á escala do manómetro
514         contar = cv2.countNonZero(mask)
515
516         # Verifica se é um manómetro de fundo branco com escala em preto
517         if pixels > img.size/2:
518             branco = True
519             if contar > valor:
520                 resultados_escala.insert(0,x)
521             i=i+1

```

```

    if mostraescala:
519         plt.imshow(mask, cmap='gray')
        plt.show()
521     # Verifica se é um manômetro de fundo preto com escala em branco
    else:
523         branco = False
        if contar < invalor:
525             resultados_escala.insert(0,x)
            i=i+1
527         if mostraescala:
            plt.imshow(mask, cmap='gray')
            plt.show()
529         # Verifica qual dos valores detetados corresponde ao início e qual
corresponde ao fim da escala
531         if len(resultados_escala) > 1:
            min = numpy.min(resultados_escala)
533             max = numpy.max(resultados_escala)

535         if max-min < 100:
            ang_in = (max + w_scan) + offset_inicio_ESCALA
537             ang_fim = min - 12
        else:
539             ang_in = (min + w_scan) + offset_inicio_ESCALA
            ang_fim = max - 12
541
543     # Converte coordenadas polares em coordenadas cartesianas
    def polar2cart(r, theta, center):
        x = r * np.cos(theta) + center[0]
545         y = r * np.sin(theta) + center[1]
        return x, y
547
549     # Converte theta de graus para radianos
    theta_in = ang_in * math.pi/180.0;
551
553     # Coordenadas cartesianas do ponto início de escala
    x_in, y_in = polar2cart(150, theta_in, centro)
555
557     # Converte theta de graus para radianos
    theta_fim = ang_fim * math.pi/180.0;
559
561     # Coordenadas cartesianas do ponto fim de escala
    x_fim, y_fim = polar2cart(150, theta_fim, centro)
563
565     # Desenha as linhas detetadas
    escalaimg = np.copy(img2)
    escala1 = cv2.line(escalaimg, (int(centro[0]),int(centro[1])),
567 (int(x_in),int(y_in)), (255, 255, 0), 3)
    escala2 = cv2.line(escala1, (int(centro[0]),int(centro[1])),
569 (int(x_fim),int(y_fim)), (255, 255, 0), 3)

571     # Caso não detete no mínimo dois valores na matrix "resultados_escala"
assume os seguintes valores
    else:
573         escala2 = np.copy(img2)
        x_in = 0
575         y_in = 0
        x_fim = 0
577         y_fim = 0

579     if mostrarimagem:
        plt.figure(figsize=(5,10))
        u1 = "Scale-marks Limits Detection Illustration"
        plt.subplot(1,1,1).set_title(u1)
        plt.imshow(escala2, cmap='gray')
581

```

```

581     return x_in, y_in, x_fim, y_fim, escala2, branco
582
583
584
585
586     ## Determinar valor da escala do manômetro
587     #####
588     ###
589
590 def OCR (img, branco, centro, raio, mostrarimagem):
591     """
592     Função para detecção dos valores apresentados na escala e respetiva posição.
593     Resulta daqui uma matrix com o valor detetado e respetiva posição central
594     (x, y).
595     """
596
597     #Inicialização de variáveis da função
598     #####
599     # Criar lista onde serão guardadas os resultantes do OCR
600     ocr_results = []
601     # Criar lista onde serão guardadas apenas os números resultantes do OCR
602     numbers = []
603     # Configuração utilizada na função tesseract
604     config = '--oem 3 --psm 12 outputbase digits'
605     # Valor para resize da imagem, em percentagem
606     resize_percent = 420#200
607
608     # Binarização
609     # Valor threshold do output
610     thresholding_value_branco = 180
611     thresholding_value_preto = 180
612     # Aqui é calculado o valor médio da intensidade de pixel em escala de
613     cinzentos de cada sub-região
614     ("pixel_neighborhood_size"x"pixel_neighborhood_size") na imagem para calcular
615     o nosso valor limiar. - APENAS NUMEROS IMPARES #751
616     pixel_neighborhood_size_branco = 751
617     pixel_neighborhood_size_preto = 581
618     # Este valor permite-nos afinar o valor limiar.
619     constant_c_adjustment_branco = 28
620     constant_c_adjustment_preto = -18
621
622     #raio = raio
623     #####
624
625 def RESIZE (img, resize_percent):
626     """
627     Função para fazer o "resize" da imagem.
628     """
629     width = int(img.shape[1] * resize_percent / 100)
630     height = int(img.shape[0] * resize_percent / 100)
631     dim = (width, height)
632     x0 = centro[1]*(resize_percent/100)
633     y0 = centro[0]*(resize_percent/100)
634     raio_ocr = raio*(resize_percent/100)
635     resize = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
636
637     return resize, (y0,x0), raio_ocr
638
639 def ROIRESIZE (img, resize_percent, raio_automatico):
640     """
641     Função para aplicar uma máscara para representar apenas os valores de
642     interesse (zona da escala).
643     """
644     ocrimg = np.copy(img)
645     x, y = np.indices(shape(img))
646     x0 = centro3[1]

```

```

        y0 = centro3[0]
541     raio_ocr = raio2
        z2 = 235#140#235/2 - 10 #-30
542
        # Definir a máscara exterior
543     mask_ocr1 = (x - x0)**2 + (y - y0)**2 < (raio_ocr -
        (raio_automatico))**2
544     imgroi_ocr1=mask_ocr1*ocrimg
545
        # Definir a máscara interior
546     mask_ocr2 = (x - x0)**2 + (y - y0)**2 > (raio_ocr-
        (raio_automatico+z2))**2
547     img2=mask_ocr2*imgroi_ocr1
548
        return img2
549
        # Resize da imagem (determina novo centro e raio devidamente escalado
        (centro2, raio2))
550     resize, centro2, raio2 = RESIZE(img, resize_percent)
551
        # Passagem para escala de cinzentos da imagem
552     GRAY = cv2.cvtColor(resize, cv2.COLOR_BGR2GRAY)
553
        # Corte da imagem (determina novo centro depois de contar imagem (centro3))
554     imgcorte2, centro3 = CORTE(GRAY, raio2, centro2, mostrarimagem=False)
555
        # Aplica diferente binarização caso seja manómetro com fundo branco ou preto
        if
        branco:
556             imgth = cv2.adaptiveThreshold(imgcorte2, thresholding_value_branco,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                    cv2.THRESH_BINARY,
        pixel_neighborhood_size_branco, constant_c_adjustment_branco)
557         else:
            imgth = cv2.adaptiveThreshold(imgcorte2, thresholding_value_preto,
        cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                    cv2.THRESH_BINARY_INV,
        pixel_neighborhood_size_preto, constant_c_adjustment_preto)
558
        raios_detecao_OCR = np.arange(-100, raio, 3)
559
        s = (500,2)
560
        raio_detecao_OCR_automatica = np.zeros((s), dtype=int)
561
        # Corre diferentes raios para a ROI do OCR por forma a descobrir aquele que
        gere mais valores
        for i in range(0, len(raios_detecao_OCR), 1):
562
            raio_automatico = raios_detecao_OCR[i]
563
            # ROI da imagem, aplicando mascara exterior e interior
564             imgroi = ROIRESIZE(imgth, resize_percent, raio_automatico)
565
            # Função tesseract para detecção dos números da escala, guarda valores
            detetados na variável "results"
            results = pytesseract.image_to_data(imgroi, config=config)
566
            for id, line in enumerate(results.splitlines()):
567
                if id != 0:
                    line = line.split()
568
                    if len(line) == 12:
                        # Posição do número detetado(x, y) e dimensões da envolvente do
                        número (w, h)
569                         x, y, w_ocr, h_ocr = int(line[6]), int(line[7]), int(line[8]),
                        int(line[9])

```

```

695         # Código para separa os digitos de outros simbolos ou letras
detetadas
        numbers = (re.findall('[0-9]+', line[11]))
697
        if numbers:
699             number_int = int("".join(numbers))
701
        # Guarda resultados na variável "ocr_results" com o valor detetado
e respetiva posição central do número encontrado
        ocr_results.append ([number_int,
int((x+(w_ocr/2))/(resize_percent/100)),
int((y+(h_ocr/2))/(resize_percent/100))])
703         #print(ocr_results)
        numeros_detetados = len(ocr_results)
705
        # Apaga os valores previamente detetados
707         ocr_results = []
709
        # Guarda o valor do raio a usar na ROI e a quantidade de numeros detetados
na lista "raio_detecao_OCR_automatica"
        raio_detecao_OCR_automatica[i] = raios_detecao_OCR[i], numeros_detetados
711
        # Determina o índice da lista onde se encontra o raio que conseguiu detetar
mais números da escala
        posicao_raio_OCR =
np.nonzero(raio_detecao_OCR_automatica[:,1]==np.max(raio_detecao_OCR_automatica[:,1]))
715
        # Guarda o melhor raio para a ROI na variável e respetivo numero de dígitos
detetados na variável "melhor_detecao"
        melhor_detecao = raio_detecao_OCR_automatica[posicao_raio_OCR[0][0]]
717
        # Corre novamente o OCR tesseract agora com o melhor parametro detetado
automaticamente
719         # ROI da imagem, aplicando mascara exterior e interior
        #print(melhor_detecao[0])
721         imgroi = ROIRESIZE(imgth, resize_percent, melhor_detecao[0])
723
        # Função tesseract para deteção dos números da escala, guarda valores
detetados na variável "results"
        results = pytesseract.image_to_data(imgroi, config=config)
725         print(results)
727
        for id, line in enumerate(results.splitlines()):
            if id != 0:
729                 line = line.split()
                if len(line) == 12:
731                     # Posição do número detetado(x, y) e dimensões da envolvente do número
(w, h)
                    x, y, w_ocr, h_ocr = int(line[6]), int(line[7]), int(line[8]),
int(line[9])
733
                    # Código para separa os digitos de outros simbolos ou letras detetadas
735                     numbers = (re.findall('[0-9]+', line[11]))
737
                    if numbers:
739                         number_int = int("".join(numbers))
741
                        # Guarda resultados na variável "ocr_results" com o valor detetado e
respetiva posição central do número encontrado
                        ocr_results.append ([number_int,
int((x+(w_ocr/2))/(resize_percent/100)),
int((y+(h_ocr/2))/(resize_percent/100))])
743
                        # Desenha retangulo que inclua os valores detetados

```

```

        cv2.rectangle(imgroi, (x, y), (w_ocr+x, h_ocr+y), (255, 255, 0), 2)

        # Escreve na imagem o valor detetado
        cv2.putText(imgroi, str(number_int), (x, y),
cv2.FONT_HERSHEY_COMPLEX, 1, (255, 0, 0), 1)

    if mostrarimagem:
        plt.figure(figsize=(20,10))
        u = "Image resize + Image ROI + Gray Scale + Binarization +
OCR(tesseract)"
        plt.subplot(1,1,1).set_title(u)
        plt.imshow(imgroi, cmap='gray')
        plt.show()
        print('Resultados do OCR =', ocr_results)

    return imgroi, ocr_results

## Verificação da simetria da escala do manómetro
#####
def OCRvalor (ocr_results, centro1, x_in, y_in, mostrarresultados):
    """
    Função para verificação se o manómetro tem uma escala simétrica ou
    assimétrica.
    Posteriormente é determinado o valor de inicio de escala e o valor de fim de
    escala.
    """
    #Inicialização de variáveis da função
    #####
    # Inicializar variável incremental "k"
    k = 0
    # Definição do desvio máximo aceitável para determinação de simetria da
    escala, em percentagem
    desvio_percent = 40
    # Ponto de inicio escala
    A = np.array([x_in,y_in])
    # Ponto do centro do manómetro
    B = np.array([centro1[0], centro1[1]])
    # Criar matrix onde serão guardadas os valores detetados
    valores_detetados = []
    # Criar matrix onde serão guardadas as amplitudes angulares entre os pontos
    consecutivos detetados
    amplitude_angular_entre_pontos = []
    # Criar matrix onde serão guardadas as resolucoes angulares entre os pontos
    consecutivos detetados
    resolucao_angular_entre_pontos = []
    # Criar matrix onde serão guardadas o valor da diferenca entre a resolucao
    do valor detetado e a média das resolucoes calculadas
    diferenca_entre_valores_media = []
    #####

    def angle(A, B, C):
        """
        Função para calculo de amplitude angular entre 3 pontos.
        """
        Ax, Ay = A[0]-B[0], A[1]-B[1]
        Cx, Cy = C[0]-B[0], C[1]-B[1]
        a = atan2(Ay, Ax)
        c = atan2(Cy, Cx)
        if a < 0: a += pi*2
        if c < 0: c += pi*2
        return (pi*2 + c - a) if a > c else (c - a)

```

```

801 def sortSecond(valor):
802     """
803     Função para ordenação de pontos por ordem crescente dos valores da
804     primeira coluna.
805     """
806     return valor[0]
807
808 # Ordenacao dos valores detetados por ordem crescente
809 ocr_results.sort(key = sortSecond)
810
811 # Sai da função se receber lista vazia
812 if shape(ocr_results)[0] == 0:
813     simetrico = False
814     resolucao_angular_entre_pontos = []
815     return simetrico, resolucao_angular_entre_pontos
816
817 img_resol_test = np.copy(imgcorte)
818
819 for i in range(0, shape(ocr_results)[0], 1):
820
821     img_test = cv2.line(img_resol_test, (int(centro1[0]), int(centro1[1])),
822     (ocr_results[i][1], ocr_results[i][2]), (255, 255, 0), 3)
823
824     valores_detetados.append(ocr_results[i][0])
825
826     x = ocr_results[i][1]
827     y = ocr_results[i][2]
828     D = np.array([x,y])
829
830     #Calculo de amplitude angular entre 3 pontos
831     ang = np.degrees(angle(A,B,D))
832     amplitude_angular_entre_pontos.append(ang)
833
834     # Calculo da resolução angular dos pontos detetados
835     if i == 0:
836         ang1 = amplitude_angular_entre_pontos[0]
837         resolucao_angular_entre_pontos.append(valores_detetados[0]/ang1)
838     else:
839         ang1 = amplitude_angular_entre_pontos[i]-
840         amplitude_angular_entre_pontos[i-1]
841         resolucao_angular_entre_pontos.append((valores_detetados[i]-
842         valores_detetados[i-1])/ang1)
843
844     # Definição do desvio máximo aceitável para determinação de simetria da
845     escala
846     val_toleravel = mean(resolucao_angular_entre_pontos)*(desvio_percent/100)
847
848     for z in range(0, shape(ocr_results)[0], 1):
849
850         # Calculo da diferenca entre o valor da resolucao e a média das resolucoes
851         detetadas
852         diferenca_entre_valores_media.append(resolucao_angular_entre_pontos[z]-
853         mean(resolucao_angular_entre_pontos))
854
855         # Incrementa valor de k caso o valor seja aceitável
856         if abs(resolucao_angular_entre_pontos[z]) <
857         (mean(resolucao_angular_entre_pontos)-val_toleravel):
858             k = k + 1
859         elif abs(resolucao_angular_entre_pontos[z]) <
860         (mean(resolucao_angular_entre_pontos)+val_toleravel):
861             k = k + 1
862         else:
863             pass
864
865     print("shape", resolucao_angular_entre_pontos)

```

```

858 if k == (len(ocr_results)):
859     simetrico = True
860 else:
861     simetrico = False

862 if mostrarresultados:
863     print('Valores detetados por ordem crescente, e posição = ', ocr_results)
864     print('Resolução angular entre valores consecutivos = ',
865           resolucao_angular_entre_pontos)
866     print('Média das resoluções angulares detetadas = ',
867           mean(resolucao_angular_entre_pontos))
868     print('Avaliação da simetria da escala do manómetro = ', simetrico)
869     plt.subplot(1,1,1)
870     plt.imshow(img_test, cmap='gray')
871     plt.show()

872     return simetrico, mean(resolucao_angular_entre_pontos)
873
874
875
876 ## Calcular valor lido no manómetro
877 #####
878 ###
879
880 def VALORLIDO (centro, x_in, y_in, x_fim, y_fim, pontoagulha, simetrico,
881               val_ocr, val_in, val_fim, mostrarresultados):
882     """
883     Aqui é feito o cálculo do valor lido.
884     Cálculo possível de forma automática ou após insercao dos valores
885     manualmente por parte do utilizador (val_in, val_fim)
886     """
887     #Inicialização de variáveis da função
888     #####
889     # Ponto de inicio escala
890     A = np.array([x_in,y_in])
891     # Ponto do centro do manómetro
892     B = np.array([centro1[0], centro1[1]])
893     # Ponto de inicio escala
894     C = np.array([x_fim, y_fim])
895     #####
896
897     def angle(A, B, C):
898         """
899         Função para calculo de amplitude angular entre 3 pontos.
900         """
901         Ax, Ay = A[0]-B[0], A[1]-B[1]
902         Cx, Cy = C[0]-B[0], C[1]-B[1]
903         a = atan2(Ay, Ax)
904         c = atan2(Cy, Cx)
905         if a < 0: a += pi*2
906         if c < 0: c += pi*2
907         return (pi*2 + c - a) if a > c else (c - a)
908
909     # Amplitude angular desde o início ao fim da escala
910     total_ang = np.degrees(angle(A,B,C))
911
912     # Ponto da agulha
913     D = np.array(pontoagulha)
914
915     #Angulo da agulha em relação ao início de escala (clockwise)
916     angagulha = np.degrees(angle(A,B,D))
917
918     #Se o manómetro tiver uma escala simétrica
919     if simetrico:
920         #Se for utilizado os valor detetados pelo OCR

```

```

910     if val_in == None:
911         val_1 = 0
912         val_2 = int(total_ang*val_ocr)
913
914         if angagulha < total_ang + 5:
915             # Calculo do valor lido no manómetro
916             valor = (angagulha * ((val_2 - val_1) / total_ang))
917         else:
918             valor = 0
919             #Se for utilizado os valor inseridos pelo utilizador!
920         else:
921             val_1 = val_in
922             val_2 = val_fim
923             if angagulha < total_ang + 5:
924                 # Calculo do valor lido no manómetro
925                 valor = val_1 + (angagulha * ((val_2 - val_1) / total_ang))
926             else:
927                 valor = val_1
928
929         #Se o manómetro tiver uma escala assimétrica
930         else:
931             print("Manómetro Assimétrico: Adicionar escala manualmente!")
932             pass
933
934     if mostrarresultados:
935         print('Amplitude angular desde o início ao fim de escala = ', total_ang)
936         print('Angulo detetado da agulha = ', angagulha)
937         print('Valor de início de escala = ', val_1)
938         print('Valor de fim de escala = ', val_2)
939         print('Valor lido = ', valor)
940
941     return valor, val_1, val_2
942
943
944 #####
945 # Código para deteção do manómetro analógico
946 #####
947 # Tarefa 0 - Inicialização da imagem
948 img = cv.imread('/content/drive/My Drive/Tese_Nuno_2021/preto/2.jpg')
949
950 # Tarefa 1 - Calibração da imagem --- (com 'Matriz da camera (mtx)' e
951 #                                     'Coeficientes de distorção (dist)'
952 guardados
953 imgdst = calibrar (img, mtx, dist, mostrarimagem)
954
955 # Tarefa 2 - Processamento imagem (RGB para Escala de cinzento para imagem
956 binária)
957 imgth = RBGTOBIN(imgdst, mostrarimagem)
958
959 # Tarefa 3 - Roi da imagem
960 imgroi, raio, precentro, imgtest = ROI(imgth, margem_ROI, mostrarimagem)
961
962 # Tarefa 4 - Deteção do centro da imagem
963 centro, imgcentro = CENTRO(imgroi, ratio_resolution_CENTRO, mostrarimagem)
964
965 # Tarefa 5 - Máscara da imagem
966 img_mask = np.copy(imgth)
967 xmask, ymask = np.indices(shape(imgth))
968 mascara = (xmask - centro[1])**2 + (ymask - centro[0])**2 <
969 (raio)**2
970 imgroi2=mascara*img_mask
971
972 # Tarefa 6 - Corte da imagem
973 imgcorte, centro1 = CORTE(imgroi2, raio, centro, mostrarimagem)
974

```

```
# Tarefa 7 - Detecção da agulha na imagem
985 pontoagulha, imgagulha, nr_linhas, thresh_agulha_determinada =
    AGULHA(imgcorte, mostrarimagem)

986 # Tarefa 8 - Detecção dos limites da escala do manómetro
    imgpolar = IMG2POLAR(imgcorte,raio,centro1, offset_escal_a_IMG2POLAR,
        mostrarimagem)
987 x_in, y_in, x_fim, y_fim, imgescala, branco = ESCALA(imgpolar, imgcorte,
    centro1, offset_inicio_ESCALA, mostrarimagem, mostraescala)

988 # Tarefa 9 - Detecção dos valores da escala do manómetro
    imgOCR, resultados = OCR (imgdst, branco, centro, raio, mostrarimagem)
989

    # Tarefa 10 - Verificação da simetria do manómetro
990 simetrico, val_ocr = OCRvalor(resultados, centro1, x_in, y_in,
    mostrarresultados)

991
    # Tarefa 11 - Computação do valor final
    valor, valor1, valor2 = VALORLIDO (centro1, x_in, y_in, x_fim, y_fim,
    pontoagulha, simetrico, val_ocr, valor1, valor2, mostrarresultados)
992

    #plt.figure(figsize=(20,10))
993 #u = "Image resize + Image ROI + Gray Scale + Binarization + OCR(tesseract)"
    #plt.subplot(1,1,1).set_title(u)
1000 #plt.imshow(imgagulha, cmap='gray')
    #plt.show()
1001 print("Valor calculado =", valor)
```

Annex E - Algorithm experimental results

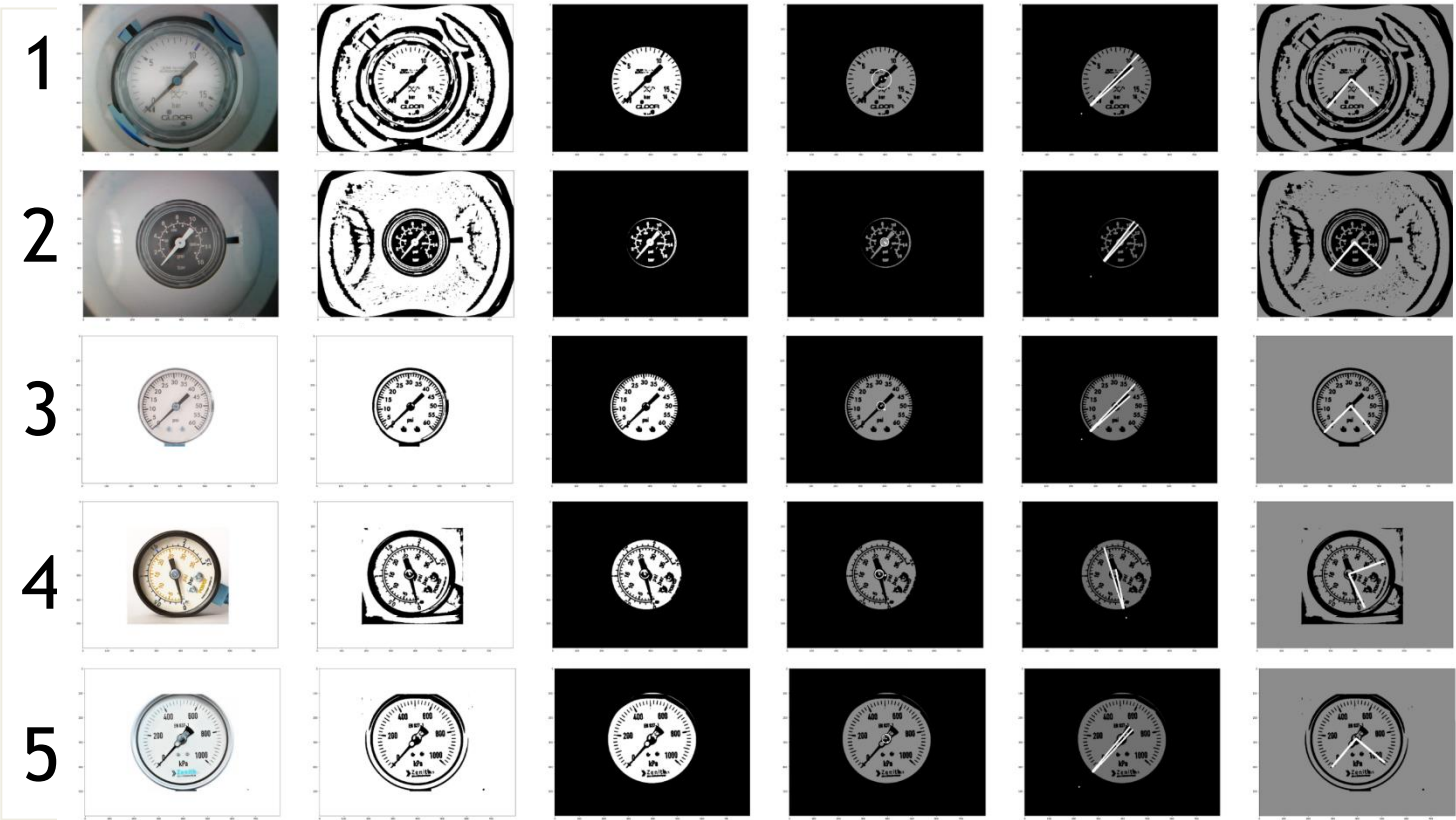


Figure 6.3 - Illustration of the results of manometers 1 to 5 used in the validation of the algorithm

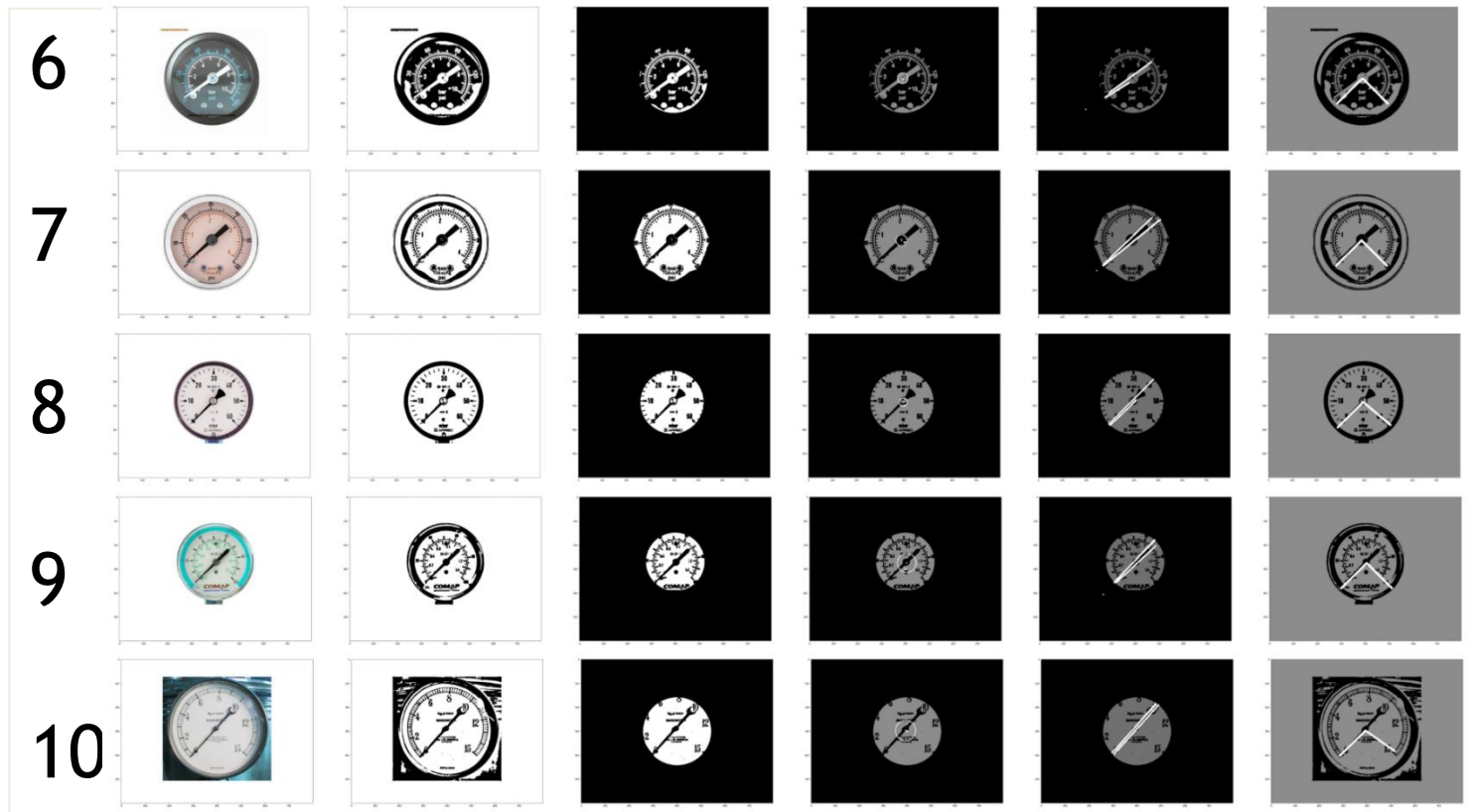


Figure 6.4 - Illustration of the results of manometers 6 to 10 used in the validation of the algorithm

The images presented in Figure 6.3 and Figure 6.4 show the results obtained after running the algorithm developed in this work. Ten gauges were tested, according to the numbering presented in the images, and for each of the gauges the original image is shown, followed by the image after binarization, followed by the result image after determining the ROI, followed by an illustration of the detection of the center, presented by a point and a circle in white, followed by the illustration of the detection of the border of the needle and finally an illustration of the detection of the limits of the gauge.

Annex F - Algorithm variables to adjust the calculated value

In order to allow the user to adjust certain parameters to get a better detection of the analog dial characteristics, three variables were created, as explained below:

- ***offset_in_SCALE*** - allows adjust the scale detection and refine the computed value. This value comes in degrees and causes the advance of the beginning of the scale clockwise;
- ***ROI_margin*** - allows tuning the ROI detection, where decreasing the value allows the radius resulting from ROI detection to be increased, and vice versa;
- ***resolution_ratio_CENTER*** - allows to adjust the detection of the center, must have values between [1, 20] and the user must try different values to obtain the best possible detection of the center.

In order to facilitate future configurations that may be necessary, a code was implemented that through a question/answer system allows to configure the 3 necessary parameters. For this the algorithm asks the user, step by step, if the detection was correct or not, if not, it asks the user to insert new parameters. The result from this new parameter is shown to the user and, when it reaches an acceptable result, the parameters introduced are saved for later readings. Such a scheme can be seen in Figure 6.5, where if the answer is “Yes” the algorithm moves on to the next calculation, if it is “No”, then the user enters new parameters and visualizes the result.

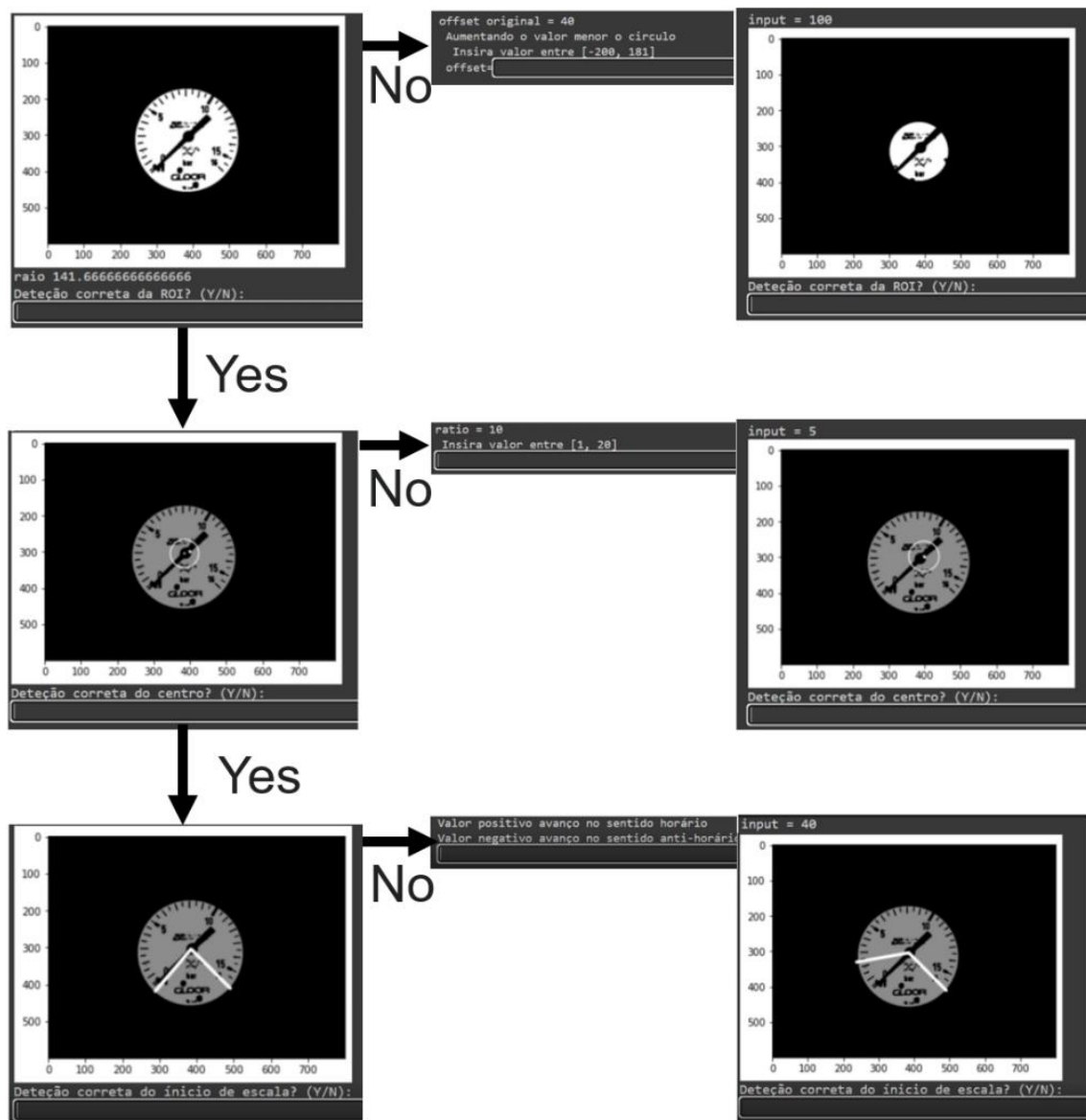


Figure 6.5 - Scheme for interactive parameter setting.

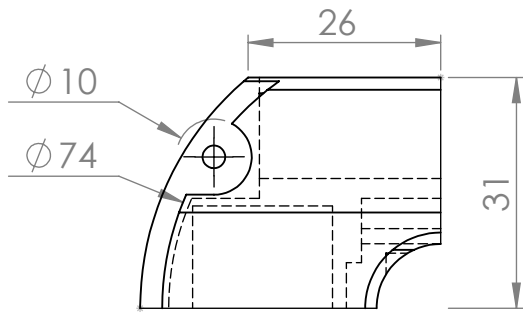
Annex G - Analog dial reader algorithm, 2nd reading

```

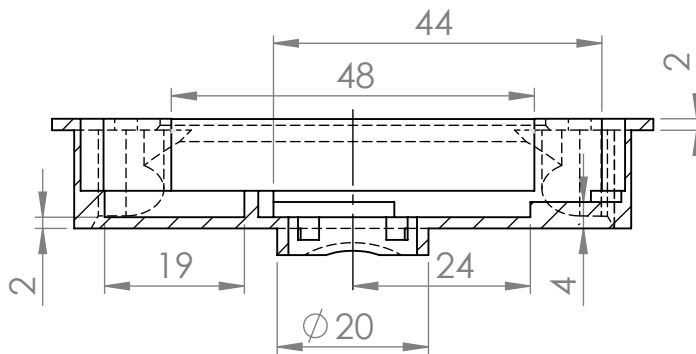
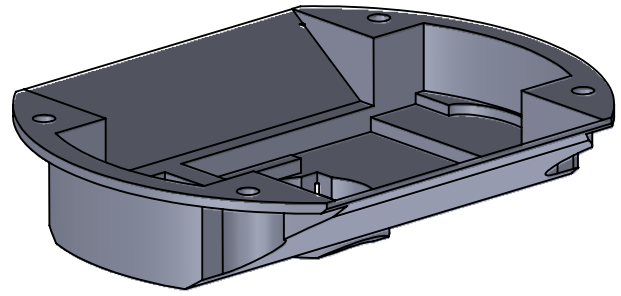
1. # Tarefa 0 - Inicialização da imagem
2. img = cv.imread('/content/drive/My Drive/Tese_Nuno_2021/preto/1.jpg')
3.
4. # Tarefa 1 - Calibração da imagem      --- (com 'Matriz da camera (mtx)' e
5. #                                     'Coeficientes de distorção (dist)'
   guardados)
6. imgdst = calibrar (img, mtx, dist, mostrarimagem=False)
7.
8. # Tarefa 2 - Processamento imagem (RGB para Escala de cinzento para imagem
   binária)
9. imgth = RBGTObin(imgdst, mostrarimagem=False)
10.
11. # Tarefa 3 - Máscara da imagem ROI      --- (com '(centro)' e
12. #                                     '(raio)' guardados)
13. img_mask = np.copy(imgth)
14. x, y = np.indices(shape(imgth))
15. mascara = (x - centro[1])**2 + (y - centro[0])**2 < (raio)**2
16. imgroi = mascara * img_mask
17.
18.
19. # Tarefa 4 - Corte da imagem            --- (com 'centro' e
20. #                                     '(raio)' guardados)
21. imgcorte, centro1 = CORTE(imgroi, raio, centro, mostrarimagem=False)
22.
23. # Tarefa 5 - Detecção da agulha na imagem --- (com 'thresh_agulha_determinada'
   guardados em "_____")
24. pontoagulha, imgagulha, nr_linhas, thresh_agulha_determinada =
   AGULHA(imgcorte, mostrarimagem=False)
25.
26. # Tarefa 6 - Computação do valor final   --- (com 'Ponto início escala (x_in,
   y_in)',
27. #                                     'Ponto fim escala (x_fim, y_fim)',
28. #                                     'Valor início de escala (valor1)',
29. #                                     'Valor fim de escala (valor2)',
30. #                                     'Simetria (simetrico)' e
31. #                                     'Resolução da escala (val_ocr)'
   guardados)
32. valor, valor1, valor2 = VALORLIDO (centro1, x_in, y_in, x_fim, y_fim,
   pontoagulha, simetrico, val_ocr, valor1, valor2, mostrarresultados=False)
33.
34. print("Valor calculado =", valor)

```

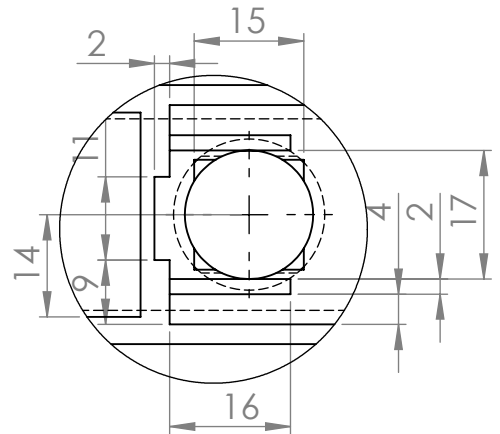
Annex H - Prototype technical drawings



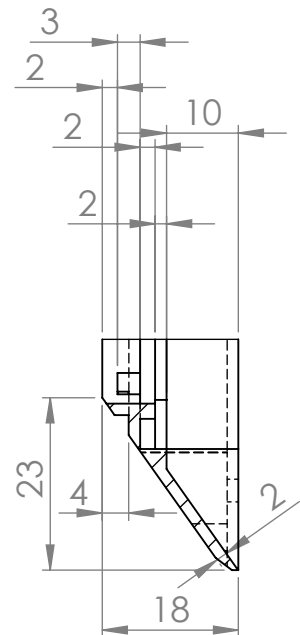
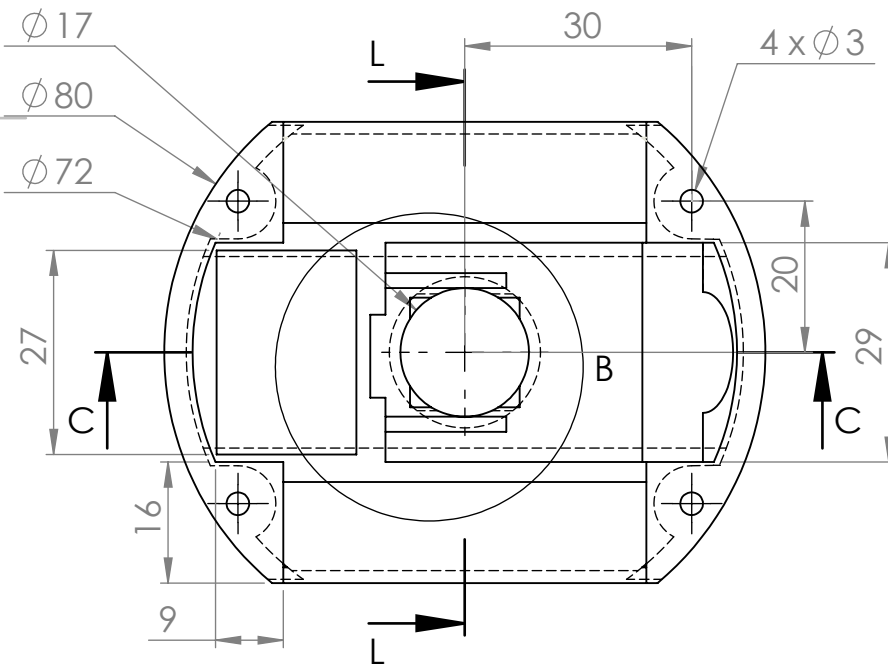
BOTTOM VIEW
SCALE 1 : 1



SECTION C-C
SCALE 1 : 1



DETAIL B
SCALE 1 : 1



SECTION L-L
SCALE 1 : 1

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND
BREAK SHARP
EDGES

DO NOT SCALE DRAWING

REVISION

NAME	SIGNATURE	DATE
DRAWN Nuno Bartolomeu		09/02/22
CHK'D		
APPV'D		
MFG		
Q.A		

TITLE:
ESP-32 support

DWG NO.

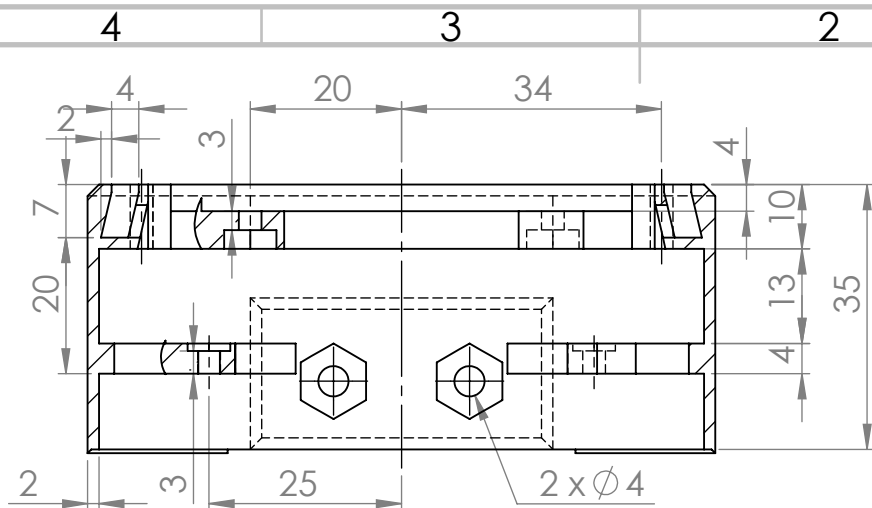
2

A4

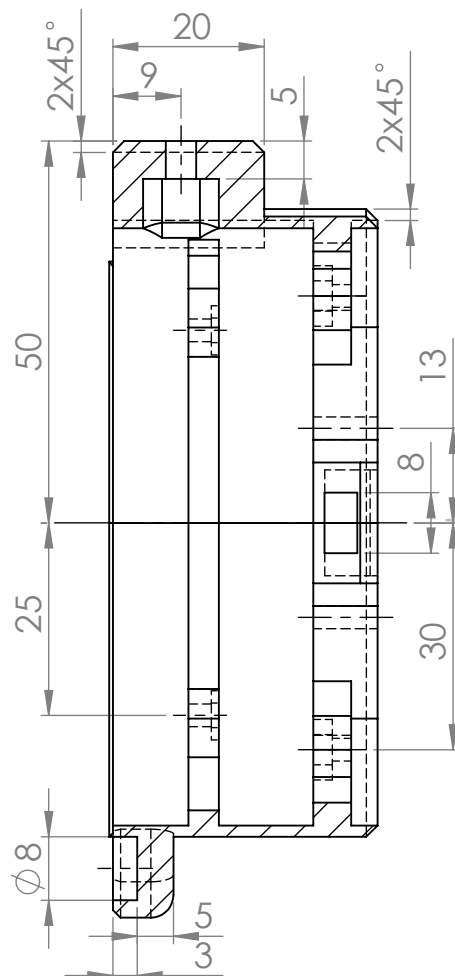
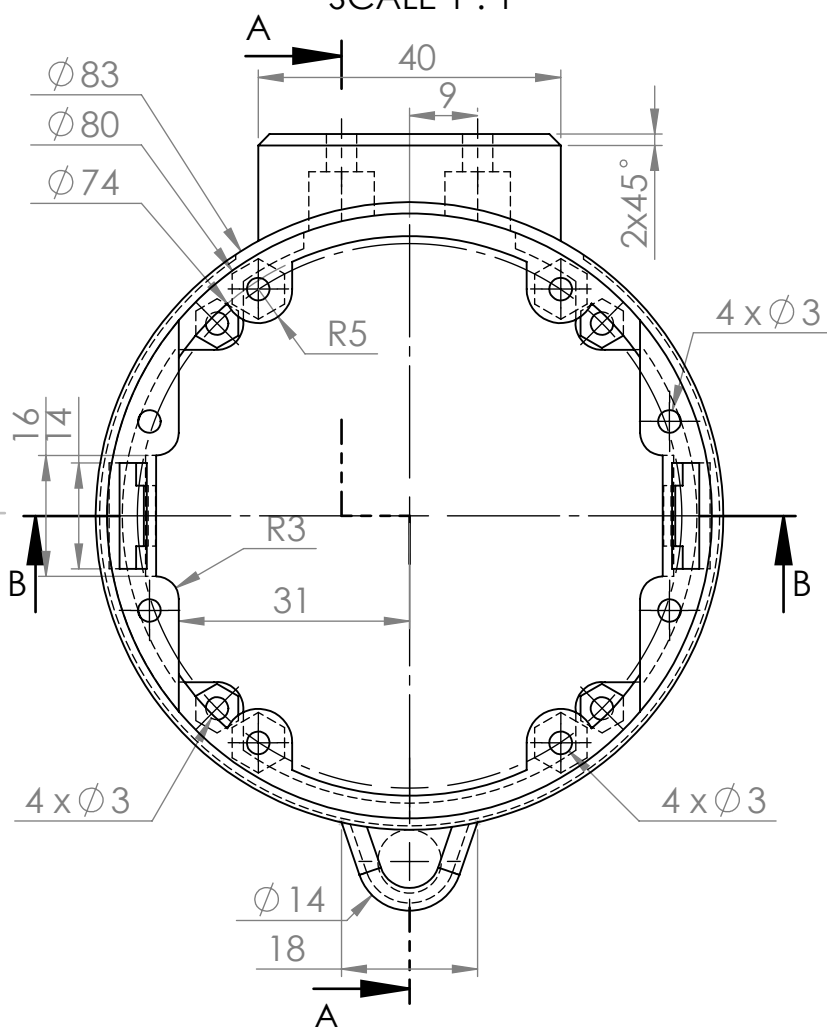
WEIGHT:

SCALE:1:1

SHEET 1 OF 1



SECTION B-B
SCALE 1 : 1



SECTION A-A
SCALE 1 : 1

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND
BREAK SHARP
EDGES

DO NOT SCALE DRAWING

REVISION

NAME	SIGNATURE	DATE		
DRAWN Nuno Bartolomeu		09/02/22		
CHK'D				
APPV'D				
MFG				
Q.A				
		MATERIAL:		
		WEIGHT:		

TITLE:

Superior support

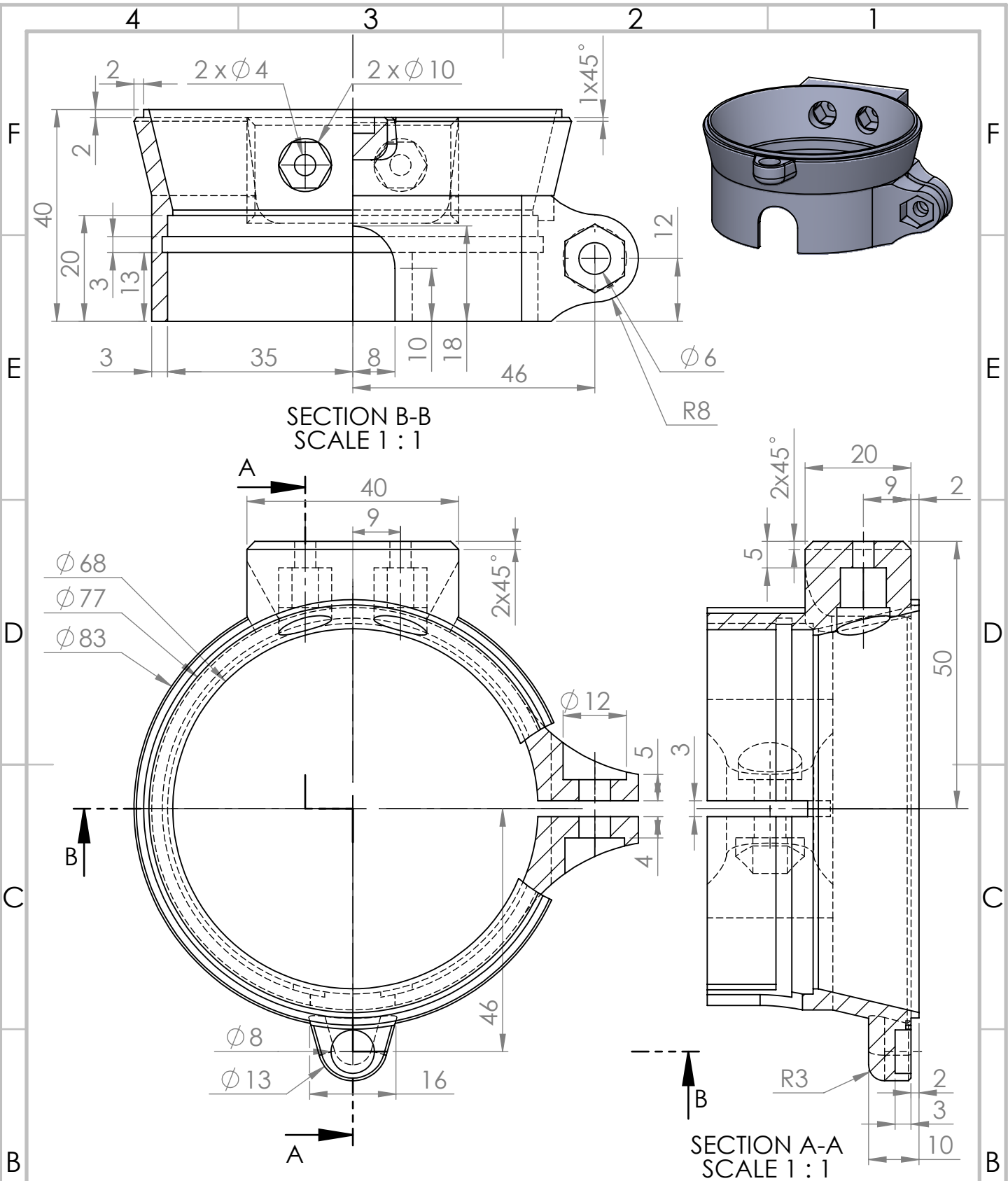
DWG NO.

3

A4

SCALE:1:1

SHEET 1 OF 1



UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND
BREAK SHARP
EDGES

DO NOT SCALE DRAWING

REVISION

NAME	SIGNATURE	DATE		
DRAWN Nuno Bartolomeu		09/02/22		
CHK'D				
APPV'D				
MFG				
Q.A				
MATERIAL:				
WEIGHT:				

TITLE:
**70 mm gauge
support**

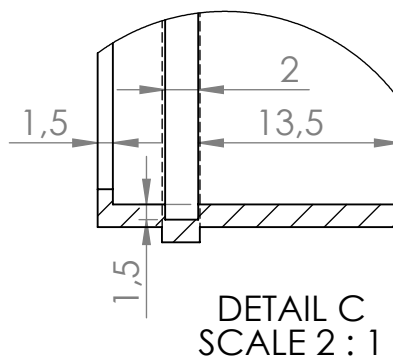
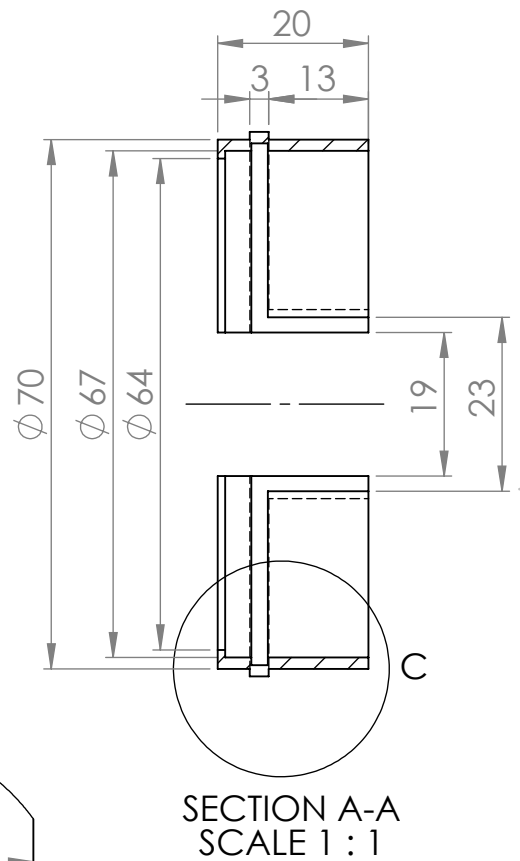
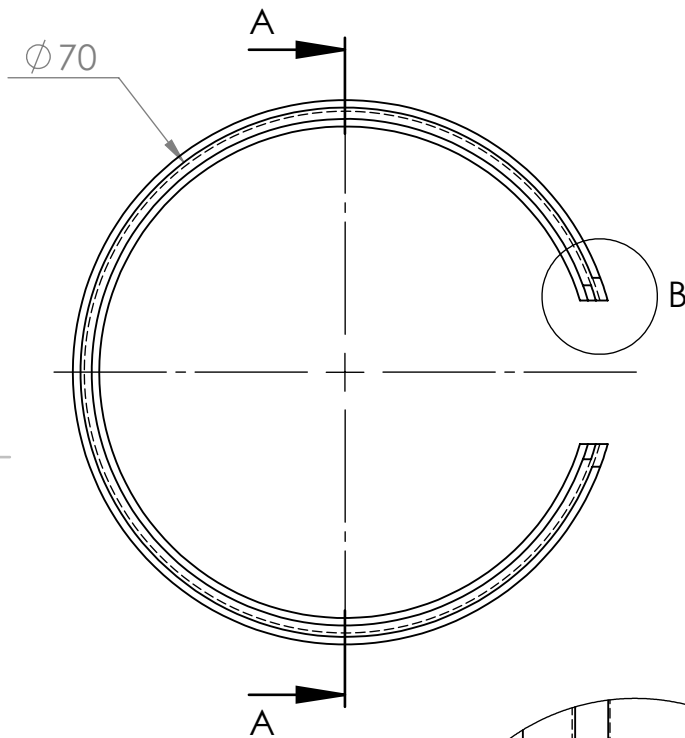
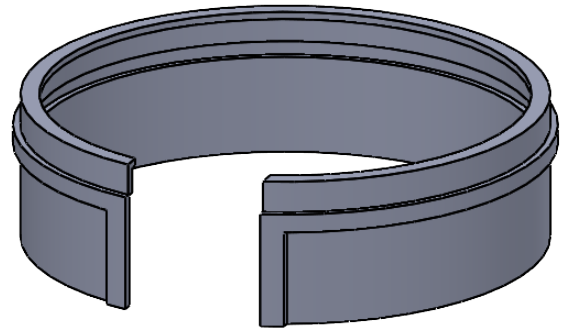
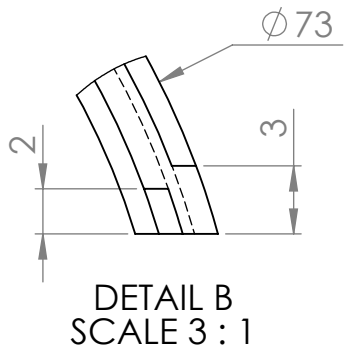
DWG NO.

4

A4

SCALE:1:1

SHEET 1 OF 1



UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND
BREAK SHARP
EDGES

DO NOT SCALE DRAWING

REVISION

NAME	SIGNATURE	DATE			
DRAWN Nuno Bartolomeu		09/02/22			
CHK'D					
APPV'D					
MFG					
Q.A					

TITLE:

70mm rim 1

DWG NO.

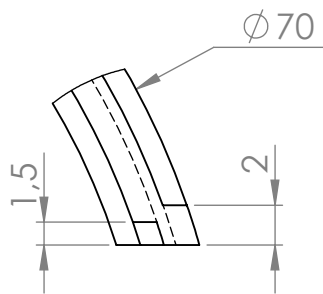
5

A4

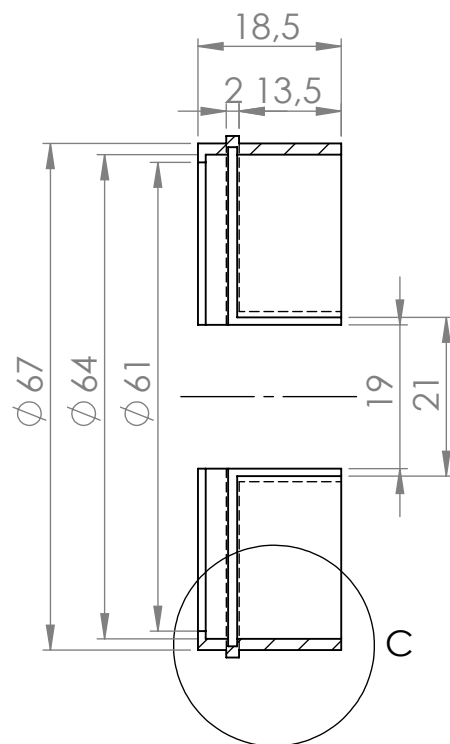
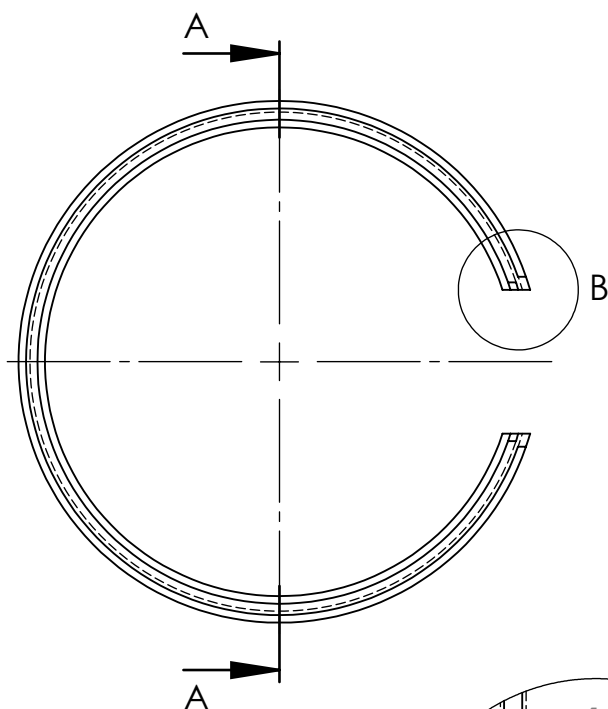
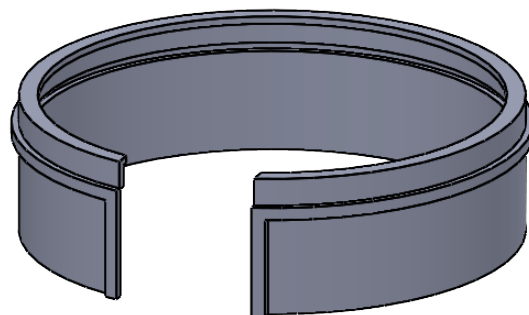
WEIGHT:

SCALE:1:1

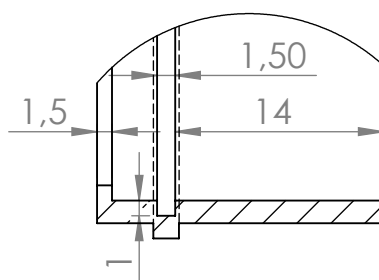
SHEET 1 OF 1



DETAIL B
SCALE 3 : 1



SECTION A-A
SCALE 1 : 1



DETAIL C
SCALE 2 : 1

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND
BREAK SHARP
EDGES

DO NOT SCALE DRAWING

REVISION

NAME	SIGNATURE	DATE			
DRAWN Nuno Bartolomeu		09/02/22			
CHK'D					
APPV'D					
MFG					
Q.A					

TITLE:

70mm rim2

DWG NO.

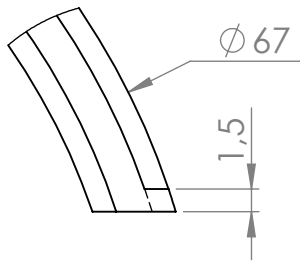
6

A4

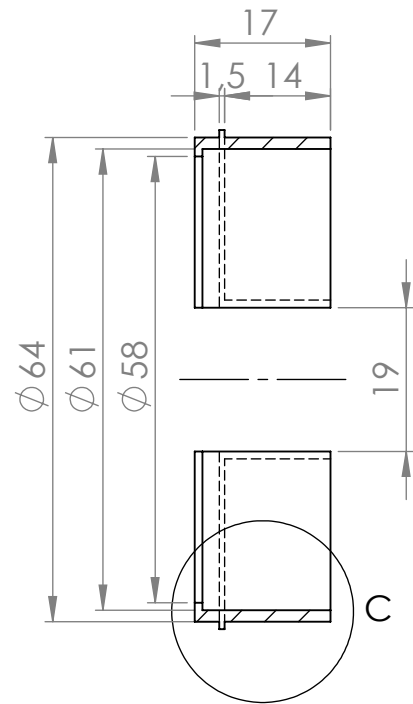
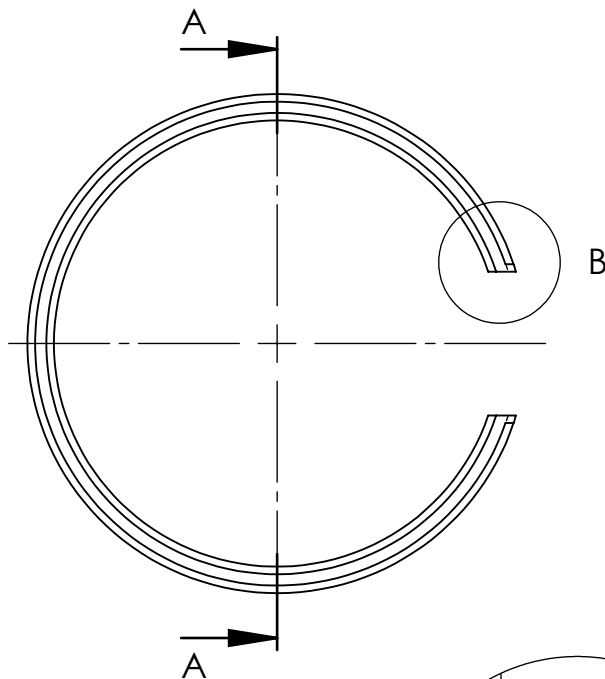
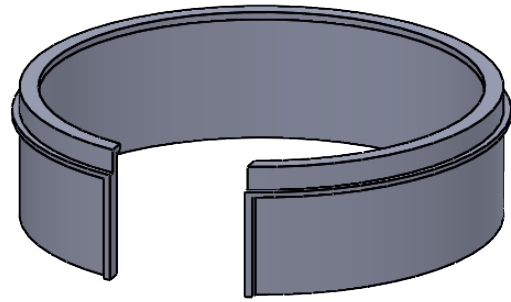
WEIGHT:

SCALE:1:1

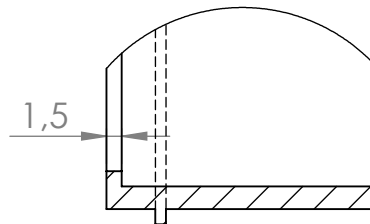
SHEET 1 OF 1



DETAIL B
SCALE 3 : 1



SECTION A-A
SCALE 1 : 1



DETAIL C
SCALE 2 : 1

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND
BREAK SHARP
EDGES

DO NOT SCALE DRAWING

REVISION

NAME	SIGNATURE	DATE			
DRAWN Nuno Bartolomeu		10/02/22			
CHK'D					
APPV'D					
MFG					
Q.A					

TITLE:

70mm rim3

DWG NO.

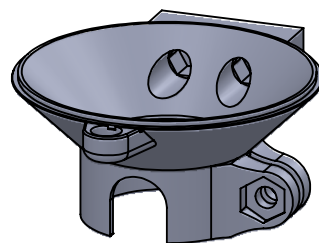
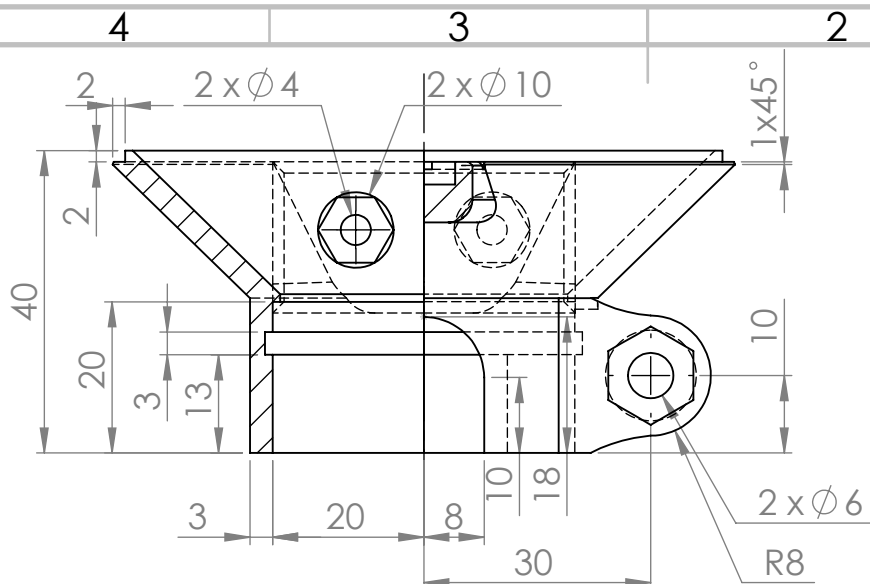
7

A4

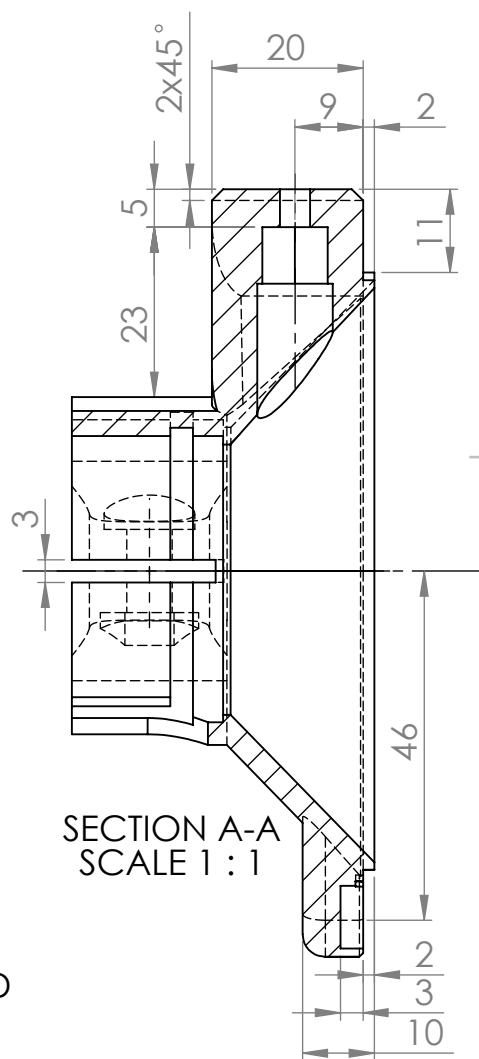
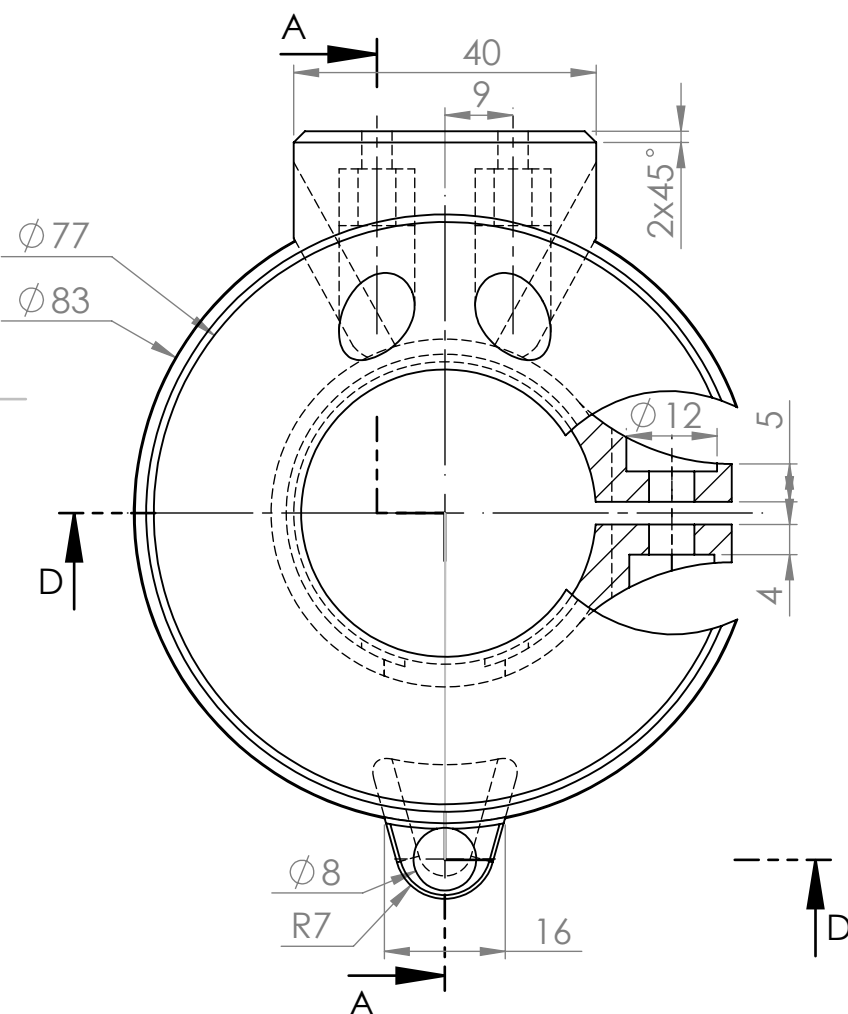
WEIGHT:

SCALE:1:1

SHEET 1 OF 1



SECTION D-D
SCALE 1 : 1



SECTION A-A
SCALE 1 : 1

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

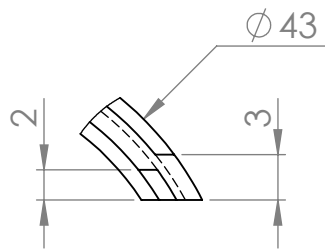
DEBURR AND
BREAK SHARP
EDGES

DO NOT SCALE DRAWING

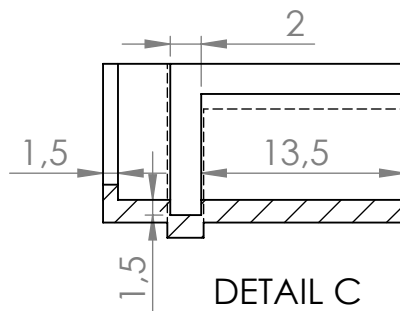
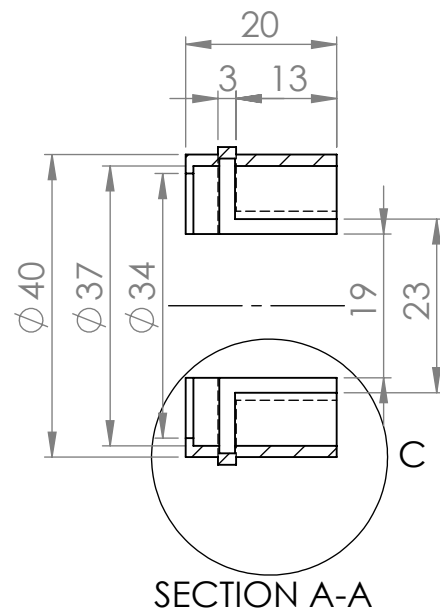
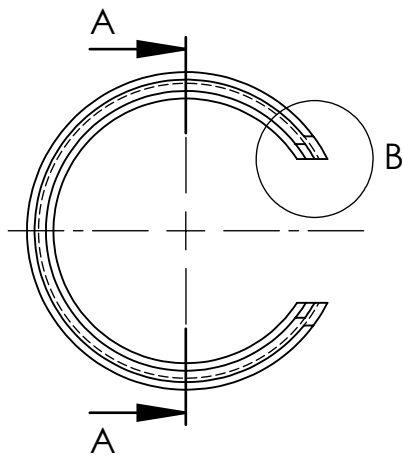
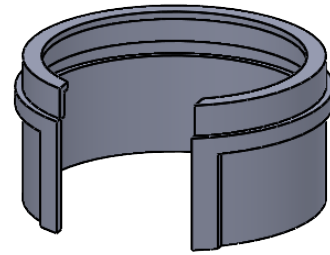
REVISION

NAME	SIGNATURE	DATE		
DRAWN Nuno Bartolomeu		09/02/22		
CHK'D				
APPV'D				
MFG				
Q.A				
		MATERIAL:		
		WEIGHT:		

TITLE:	40mm gauge support	
DWG NO.	8	A4
SCALE:1:1	SHEET 1 OF 1	



DETAIL B
SCALE 2 : 1



DETAIL C
SCALE 2 : 1

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND
BREAK SHARP
EDGES

DO NOT SCALE DRAWING

REVISION

NAME	SIGNATURE	DATE			
DRAWN Nuno Bartolomeu		09/02/22			
CHK'D					
APPV'D					
MFG					
Q.A					

TITLE:

40mm rim 1

DWG NO.

9

A4

WEIGHT:

SCALE:1:1

SHEET 1 OF 1

