Object detection and localization: an application inspired by RobotAtFactory using Machine Learning

Leonardo Pilarski

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Engenharia Industrial.

Work oriented by:

Professor José Luis Sousa Magalhães Lima

Professor Alberto Yoshihiro Nakano

Prof. João Afonso Braun

Bragança

2023

Object detection and localization: an application inspired by RobotAtFactory using Machine Learning

Leonardo Pilarski

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master Degree in Engenharia Industrial.

Work oriented by:

Professor José Luis Sousa Magalhães Lima

Professor Alberto Yoshihiro Nakano

Prof. João Afonso Braun

Bragança

2023

Dedication

I dedicate this work to everyone who somehow helped me and made this possible. Mainly to my parents, Janete and Miguel, for all the support, encouragement, motivation, education and love. To my siblings for all the moments of relaxation, charges, and for being the reason for making me a better person every day, for inspiring them. To my grandparents, for being very important in my personal growth, giving me a vast personal and professional knowledge in many areas, as well as a great affection, reciprocal, shown to me by them. To my second family, Wilson and Sônia, with whom I spent most of my life, who provided me with knowledge, experiences, and affection, helping me to become the person I am today. To my undergraduate colleagues, especially Luiz E. M. Luiz, who besides being a great advisor in the academic field, through opinions and discussions in the development of this work, provided me with lessons about life, being an example of person and professional to be inspired....

Acknowledgement

I would like to thank professors José Luis Sousa Magalhães Lima and Alberto Yoshihiro Nakano for all the guidance, advice, and experience given to me, and also for the consideration and commitment to the work developed, where they gave their time for it. I also thank the guidance and advice of Professor João Afonso Braun, member of CeDRI, for giving his time to help and advise me. I would also like to remember and thank, all the professors with whom I had the pleasure of learning and being inspired within engineering, as well as all those who were part of my academic path. Still in the academic scope, I thank the educational institutions UTFPR and IPB, for allowing my participation in the double degree program. I also thank the friends I made in Portugal for helping me, not only in the academic field, but also in life experiences. Special thanks to Lucas Colla, David Valente, and Joao Armando Fortes Lima.

Abstract

The evolution of artificial intelligence and digital cameras has made the transformation of the real world into its digital image version more accessible and widely used. In this way, the analysis of information can be carried out with the use of algorithms. The detection and localization of objects is a crucial task in several applications, such as surveillance, autonomous robotics, intelligent transportation systems, and others. Based on this, this work aims to implement a system that can find objects and estimate their location (distance and angle), through the acquisition and analysis of images. Having as motivation the possible problems that can be introduced in the robotics competition, RobotAtFactory Lite, in future versions. As an example, the obstruction of the path developed through the printed lines, requiring the robot to deviate, and/or the positioning of the boxes in different places of the initial warehouses, being positioned so that the robot does not know its previous location, having to find it somehow. For this, different methods were analyzed, based on machine leraning, for object detection using feature extraction and neural networks, as well as object localization, based on the Pinhole model and triangulation. By compiling these techniques through python programming in the module, based on a Raspberry Pi Model B and a Raspi Cam Rev 1.3, the goal of the work is achieved. Thus, it was possible to find the objects and obtain an estimate of their relative position. In the future, in a possible implementation together with a robot, this data can be used to find objects and perform tasks.

Keywords: Machine Learning, neural networks, object detection, localization, robotics competition.

Resumo

A evolução da inteligência artificial e das câmeras digitais, tornou mais acessível e amplamente utilizada a transformação do mundo real, para sua versão em imagem digital. Dessa maneira, a análise das informações pode ser efetuada com a utilização de algoritmos. A deteção e localização de objetos é uma tarefa crucial em diversas aplicações, tais como vigilância, robótica autônoma, sistemas de transporte inteligente, entre outras. Baseado nisso, este trabalho tem como objetivo implementar um sistema que consiga encontrar objetos e estimar sua localização (distância e ângulo), através da aquisição e análise de imagens. Tendo como motivação os possíveis problemas que possam ser introduzidos na competição de robótica, Robot@Factory Lite, em versões futuras. Podendo ser citados como exemplo a obstrução do caminho desenvolvido através das linhas impressas, requerendo que o robô desvie, e/ou o posicionamento das caixas em locais diferentes dos armazéns iniciais, sendo posicionadas de modo que o robô não saiba sua localização prévia, devendo encontra-las de alguma maneira. Para isso, foram analisados diferentes métodos, baseadas em machine leraning, para deteção de objetos utilizando extração de características e redes neurais, bem como a localização de objetos, baseada no modelo de Pinhole e triangulação. Compilando essas técnicas através da programação em python, no módulo, baseado em um Raspberry Pi Model B e um Raspi Cam Rev 1.3, o objetivo do trabalho é alcançado. Assim, foi possível encontrar os objetos e obter uma estimativa da sua posição relativa. Futuramente, em uma possível implementação junta a um robô, esses dados podem ser utilizados para encontrar objetos e executar tarefas.

Palavras-chave: Machine Learning, redes neurais, deteção de objetos, localização, competição de robótica.

Contents

List of Tables

List of Figures

Acronyms

AP	Average Precision.
API	Application Programming Interface.
BB	Bounding Box.
CNN	Convolutional Neural Network.
DNN	Deconvolutional Neural Network.
FC	Fully-Connected.
FPS	Frames per Second.
fps	frames per second.
GB	Gigabyte.
GPU	Graphics Processing Unit.
hFoV	Horizontal Field of View.
IDE	Integrated Development Environment.
k	Anchor.
mAP	mean average precision.

MP megapixels.

NMS Nonmaximum Suppression.

OpenCV Open Source Computer Vision Library.

OS Operational System.

pixels picture element.

PLA polylactic acid.

Python programming language.

RoI Region ff Interest.

RPN Region Proposal Networks.

SD Secure Digital.

SPM Spatial Pyramid Matching.

SSD Single Shot Multibox Detector.

SVM Support Vector Machine.

vFoV Vertical Field of View.

YOLO You Only Look Once.

Chapter 1

Introduction

Artificial intelligence seeks to create techniques that mimic human intelligence. Computer vision [1] is one of them and can be understood as a set of mathematical and computational techniques to process images and extract relevant information. Although some algorithms attempt to mimic human vision, they are not limited to that [2]. Most focus on extracting specific features from images, such as colors or shapes, analyzed in a personalized way, such as in object recognition [3].

This insight plays a key role in robotics [4], particularly in mobile robotics, involving data acquisition and its use to perform tasks. This multidisciplinary research area has a wide variety of applications, such as space exploration [5], autonomous transportation [6], and flexible manufacturing systems [7]. Advances in this area have enabled robots to perceive and understand their surroundings, using cameras and other sensors to collect visual information about the terrain. This information, robots to plan their trajectories and make real-time decisions to move efficiently and safely [8].

Robotics competitions are a crucial setting for the inception of technological development [9]. These can be considered the starting point for research and development in many areas, such as science and technology, as they seek to solve competition problems with many innovative ideas [10]. One notable example of such competitions is RobotAtFactory [11], which aims to simulate a factory environment and transportation objects. Over the years, it has evolved regarding technology development and complexity. Its most recent version [12] had changed regarding its competition area, replacing continuous lines by spaced markers printed on the floor.

The analysis of RobotAtFactory's evolution, based on the Lite version of the competition in 2022 and the certified paper[13] in appendix A, raises questions about its upcoming changes. Regarding this, some assumptions can be made. One of them would be distributing the boxes randomly in the competition scenario, unlike the current way, where they all have their initial location in the entry warehouse,thus, forcing the compet it ors to develop systems capable of locating these objects. Linked to this, besides locating the boxes, this system would serve to identify possible obstacles, which are obstructing the route to be performed by the robot.

Object detection aims to identify all occurrences of elements from one or more recognized categories. Usually, a small number of elements are present in the image, however there are a vast number of potential locations and sizes, which must be analyzed [14]. There are several methods for detecting objects and obtaining their location in the image. All identifications are recorded with information that ranges from the object's simple position to its size, using a bounding box or a clipping mask. In some cases, entering information about the linear or non-linear transformation results in a more precise location in the image [15].

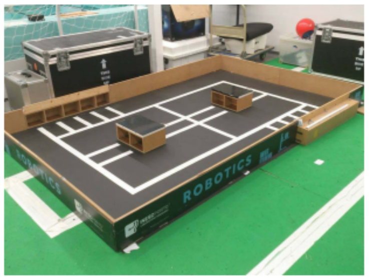
The detection information estimate the distance between the camera and the detected object in the image using its dimension information. For this, one or more cameras [16] used in a process is known as depth or distance estimation. Several techniques that can be used to perform this estimation, including geometric models and image feature analysis [17]. One such technique is the Pinhole model [18], based on the relationship between the sizes of the object in the real world and its produced image. Distance estimation is essential in many applications, such as autonomous vehicles [19] and virtual reality [20], among others, where the accuracy of object location is critical for correct decision making.

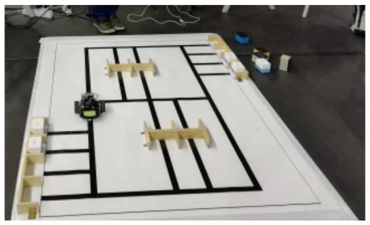
1.1 RobotAtFactory Competition

The first edition took place in Lisbon in 2011 at the Robotics Festival [21]. The competition sought to recreate problems autonomous robots face in a factory environment. Its constitution was based on three parts, the initial warehouse, eight processing machines and the final warehouse. The robots' task was to get items from the initial warehouse to the final one quickly, and sometimes to pass through the machines first. They should have to collect the boxes, transport and allocate them in each stage, simultaneously locating and navigating the environment, avoiding obstacles and reaching the goal. The competition track was formed by white lines on a black background, connecting warehouses and machines. The competition was divided into three sleeves, with increasing difficulty, introducing different types of items at each stage, differentiated through different LEDs, related to their type of processing.

In 2019, the race evolved to its Lite version [11]. Based on the rules of the first version, it featured hardware and software simplifications, attracting smaller controlled robots. Materials could now be dragged, using an electromagnet for coupling, decreasing the requirement for picking materials and focusing on path performance. The shop floor changes to two A0 sheets, with black line printing on a white background. The complexity of the sleeves remains the same, changing only the identification of the different items, at this moment being done through RFID identification.

The 2021 confinement brought restrictions for competition. At that time, the competition was played virtually, through the SIMTWO simulator. The rules were similar to the face-to-face version, however, in this competition the information about the different subjects, with no possibility of RFID identification, was done by sending information through a simulator or variable. In 2022, a transition started from RFID material identtification and WIFI use and also presented the evolution up to version 4.0 of the test. The rules are based on its previous editions, with complexity changes related to location and navigation [12]. Instead of lines, ArUco Markers are printed on the ground, possibly using reflectors in the track corners.


(a)


(b)

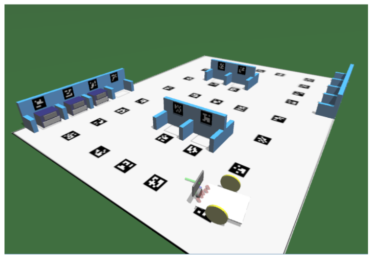

(c)

Figure 1.1: Evolution of RobotAtFactory competition arena (a) RobotAtFactory [22], (b) RobotAtFactory Lite [23] and (c) RobotAtFactory 4.0 virtual view [12]

1.2 Motivation

Considering the proximity to the academic environment, robotics competitions offer an initial path for students to experience problem-solving first-hand. A good example is the Robot@Factory Lite competition, which bases its competition area on a factory floor with printed lines and machines. The main product worked on in this race is 3D printed boxes, transported from one warehouse to another, using the lines to trace their route. However, if the lines are obstructed, the robot's path is affected, and may even lead to a collision. Looking further ahead, if the test were to evolve in the future and the boxes were scattered randomly in the environment without any prior information about their location, or if a box were to get lost in the middle of the route, it would need to be located somehow. Based on these possibilities, the robot must have a system that first detects these objects, estimates their position in the real world, and then performs the other test requirements.

1.3 Objective

Based on this motivation, this work aims to develop an object detection and localization system. It can be used to identify possible obstacles in a pre-defined path or even locate objects in an unknown environment. Also, without suffering interference, it can operate in an indoor environment.

The scope of the objective encompasses:

- Using the Raspberry Pi Cam Rev 1.3 coupled with a Raspberry Pi 4 Model B - 4 Gigabyte (GB) RAM;
- Obtaining images of the environment;
- Using neural networks;
- Train a custom object detector;
- Pass training data to the detection model;
- Locate the objects in the image;
- Parameterize its dimensions in the image and the real world;
- Calculate the distance to the object; It is
- Calculate the angular orientation to the object.

1.4 Document Structure

The document structure is presented as follows:

- Chapter 1 presents an introduction to the concepts related to the proposed work. It begins by providing an overview of the topic and discussing of the research motivation and objectives. The chapter concludes with a presentation of the structure of the document;
- Chapter 2 reviews many studies on object detection and localization methods. It begins by presenting proposed region-based methods, which divide the image into smaller images and analyze each region to detect objects. Next, regression-based methods are presented, which get a faster response by predicting the location of objects in the image with the help of neural networks. Next, a review of machine

vision applied to localization is presented. Reviewing these studies is intended to provide a context and a theoretical basis for the work on object detection and localization that will be presented later;

- Chapter 3 begins by presenting the definition of the problem being addressed. Then it presents the methods and mathematical formulations used to parameterize the relationship between the real world and digital images. Information about the components and technical specifications involved by the hardware, including details about the camera used, the processor, memory and the components necessary for it to work, is presented. Afterward, the embedded operating system and the built-in tools for running the software are presented. Finally, the chapter briefly explain the algorithms developed, as techniques that are used to detect objects and how they work within the system.

- Chapter 4 begins by showing the union of hardware components, the installation of the operating system, and tools used by the software, forming the detection module. The chapter provides a detailed account of the algorithms developed for the system's operation, along with the object detection techniques employed and their functioning within the system. Finally, the way in which the object detection method was trained is presented, including details about the dataset and the training techniques employed;

- Chapter 5 presents the tests and results related to object training and detection. First, the tests that characterize the training quality of the object detection model and their results are presented, to evaluate the effectiveness of the trained model. Next, the tests of the detection module and the results obtained are described, evaluating its performance in different situations and scenarios; and

- Chapter 6 presents the conclusions based on the results obtained, with the tests carried out during the development of the object detection system. The performance of the system in different scenarios is discussed, as well as its limitations and possibilities for improvement. The main contributions of the work and the achieved objectives are presented. In addition, the chapter proposes future work and lines of investigation that can be pursued to improve the system and extend its reach in terms of applicability.

Chapter 2

State of Art

2.1 Object Detection and Classification

From the classification method, one can evolve to detection and expand its application. The main objectives of object detection involve finding different objects present in the image, through rectangular boxes, and classifying them according to their category.

Generic detection methods are divided into two groups as dhowed in 2.1. The first follows the traditional principle of separating the image into smaller parts and later joining the common parts, following the principle of dividing a big problem into smaller ones. The second, on the other hand, seeks a faster response to the problem, working with regression and adopting a unified structure to reach the final results directly. Next, the state of the art of [24] object detection will be presented.

2.1.1 Detection Based on Region Proposal

This structure is based on two fundamental aspects. Firstly, it scans all the information, and then it analyzes the Regions of Interest Region ff Interest (RoI) specifically. It draws inspiration from works such as [25], [26], and primarily [27], which utilize Convolutional Neural Networks Convolutional Neural Network (CNN) to project the Bounding Boxes Bounding Box (BB)s directly onto the regions of highest confidence in the resource map.

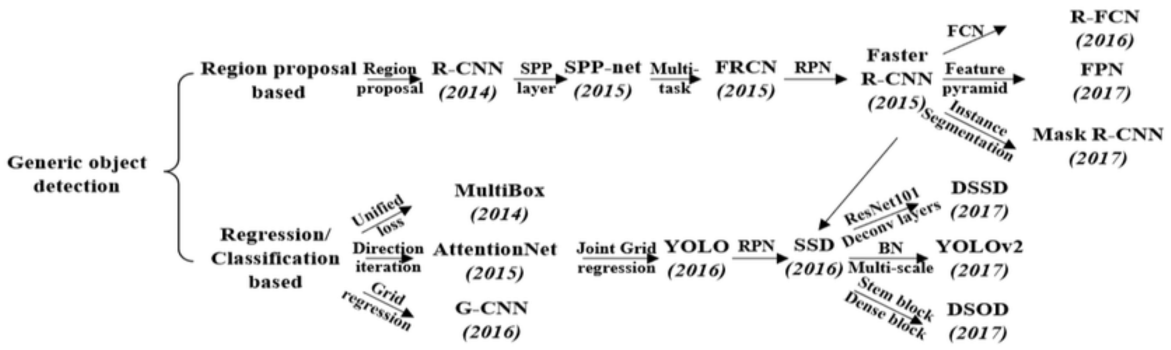


Figure 2.1: Two types of frameworks: region proposal based and regression/classification based [24].

R-CNN

In 2014, Girshick proposed the R-CNN algorithm [28] considered the first convective neural network-based object detection model obtaining an accuracy of 66% mAP. As illustrated in Figure 2.2, the model uses selective search to extract about 2000 proposed regions from each image to be detected [29]. Subsequently, each region is manipulated and sent to the CNN module [30], generating a feature vector with 4096 dimensions for each region. This vector is fed into the Support Vector Machine (SVM) classifier [30], where a linear bounding box regression occurs, delimiting objects by scoring regions. Although its accuracy is significantly higher than the traditional detection method, the required computation is very large, resulting in low efficiency. In addition, directly scaling the proposed region to a fixed-length feature vector may cause object distortion.

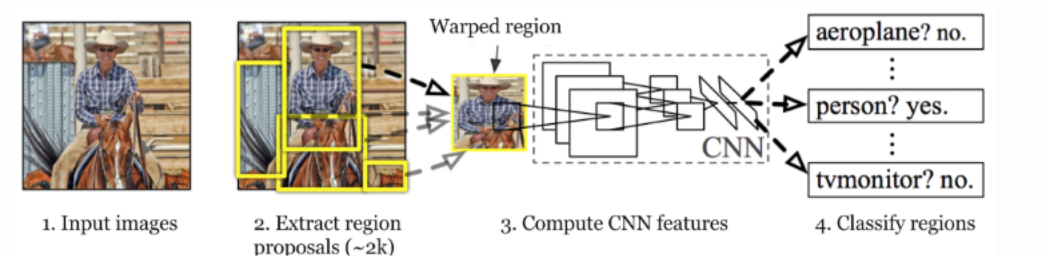


Figure 2.2: Flowchart of R-CNN [31]

SPP-Net

The Fully-Connected (FC) layers take a fixed size input. The deformation or cut of each proposed region is necessary, but with this comes the problem of possible exclusion of areas of interest.The variable scale of objects results in a reduction in recognition accuracy.

To work with this problem, He et al.[32] studied the theory of Spatial Pyramid Matching (SPM)[33], proposing a new architecture called SPP-Net. This method adopts more refined parameters to divide the image and its architecture can be seen in Figure 2.3. This method reuses data from the fifth conv layer (conv5), and according to the number of layers of the pyramid and the number of feature maps in the conv5 layer, we have, at the end of the SPP layer, the final feature vector. We obtain better results for regions with different scales and better performance by sharing of computational cost before the SPP layer.

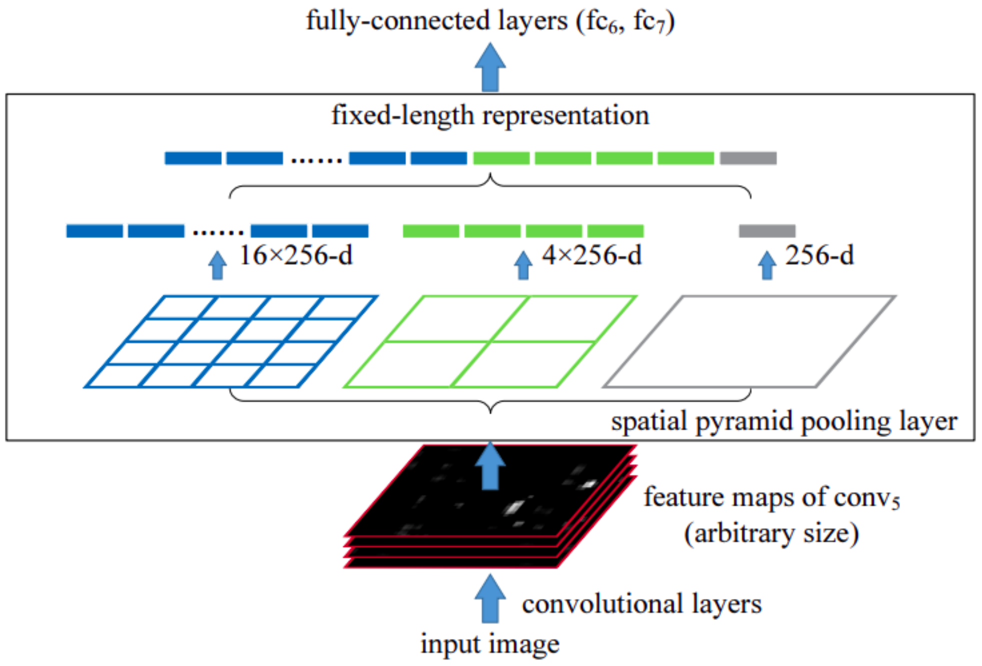

fully-connected layers (fc_6, fc_7)
fixed-length representation
16×256-d
4×256-d
256-d
spatial pyramid pooling layer
feature maps of $conv_5$
(arbitrary size)
convolutional layers
input image

Figure 2.3: Archtecture of SPP-Net for Object Dection [32]

Fast R-CNN

SPP-Net has shown impressive improvements over R-CNN, but it still has disadvantages. It uses many stages similar to R-CNN, causing an additional expense of unnecessary storage space. Layers before the SPP layer cannot be updated, with fine-tuning [32] decreasing precision. To work around this problem, Girshik et al.[28] made some modifications to their initial project, creating Fast R-CNN.

The architecture is similar to SPP-Net, and the entire image is analyzed at the conv layer, instead of splitting it into 2000 initial proposals, producing a convolutional feature map. Then a feature vector is extracted from each proposed region using a pooling layer RoI reshapes itself to a fixed size for sending to FC. From the resource vector RoI two outputs are obtained, one through a softmax layer predicting the class of the proposed region for all categories, including the background. The other uses regression to encode the refined bounding box positions, producing four real values. Thus, training all layers can be done at once, lowering storage space and accuracy expenses.

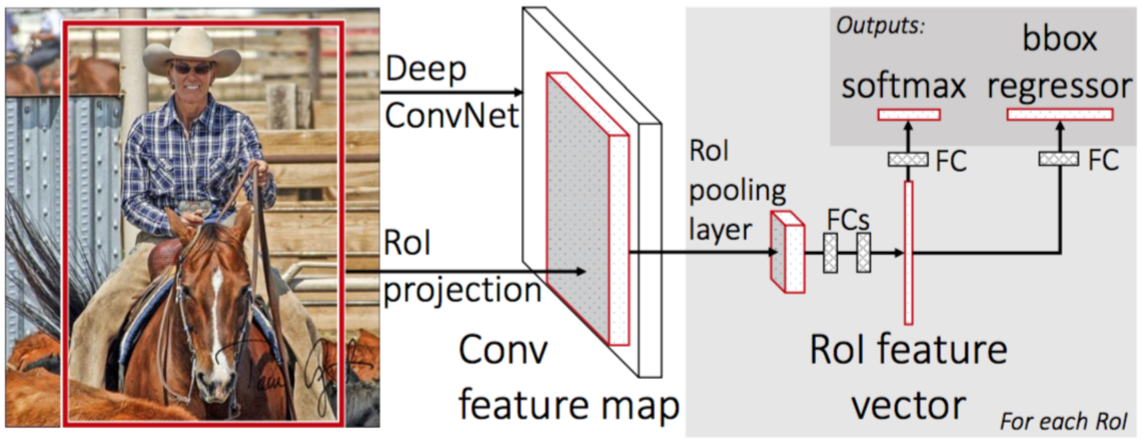


Figure 2.4: Archtecture of Fast R-CNN [31]

Faster R-CNN

Predecessor methods to Faster R-CNN used selective search to find proposed regions, making the process slow and affecting network performance. To solve this, Ren et al.[34]

[35] created the Faster R-CNN method that does not use selective search, streamlining the process and allowing the network to learn the proposed regions. In this method the image is provided as input, similar to its predecessor method, providing the map of convolutional features. Instead of traversing the entire map through selective search, using a separate network speeds up the process, working simultaneously. This small network is slid over the convolutional resource map of the last layer. Next, Region Proposal Networks (RPN) generate object proposals, using a classifier and a regressor. In this process, the concept of Anchor (k) was included, which is the central point of the sliding window. The classifier informs the probability of the proposal containing an object and the regressor presents the coordinates. These are remodeled in a grouping layer RoI, classifying the image within the analyzed region and modeling the displacement and allocation of the bounding boxes.

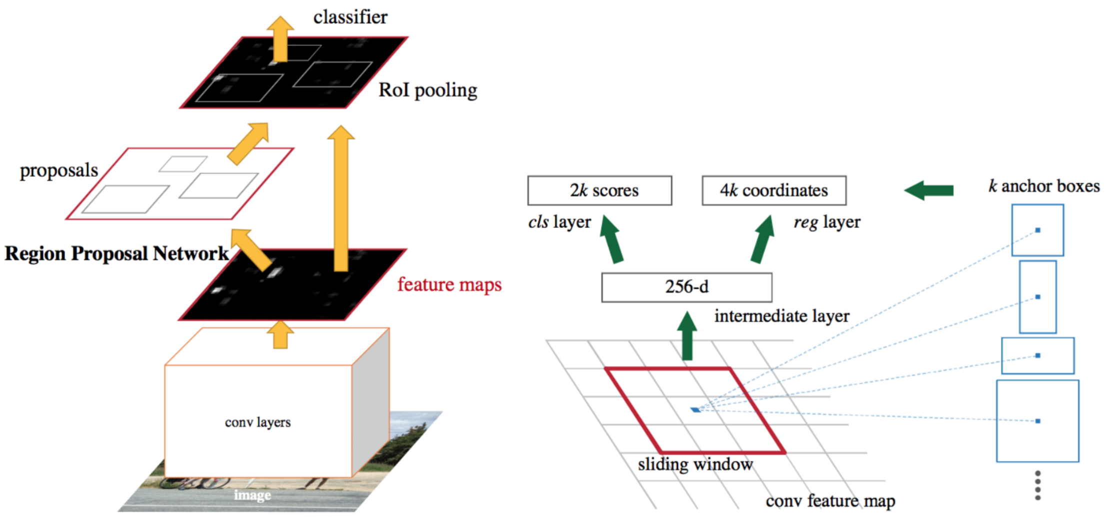


Figure 2.5: Archtecture of Faster R-CNN [34]

2.1.2 Regression/Classification-Based Framework

Regression/classification-based frameworks are more agile, incorporating fewer processes into their environment. Instead of generating region proposals or extracting features with CNNs, they proceed directly to image pixel analysis, classifying the bounding box position and class, thus reducing the analysis time and increasing the speed of obtaining answers.

The context of these methods, based on [24], is analyzed below, with You Only Look Once (YOLO) [36] and Single Shot Multibox Detector (SSD) [37].

YOLO

The original proposal of YOLO [36] utilized a large feature map to accelerate the process, eliminating the need for region proposal extraction, to detect bounding boxes BBs and confidence levels related to the class. The operational process of this method is described as follows is based on dividing the input image into a grid, with cells of S $\times$ S dimensions. Each cell in the YOLO model predicts B bounding boxes BBs and their corresponding confidence levels. The BBs are then used to locate the object within the linear reference of the cell.

The confidence scores [24] are defined as the probability of objects existing and the confidence of that prediction. Additionally, the probability of there being C classes in each cell is calculated simultaneously. If both criteria are met, then they are taken into consideration. On a single TitanX, its detection speed can reach 45fps working in real time, but has poor recognition performance when dealing with objects in group form.

According to the authors, the YOLO approach offers significant major advantages over traditional object detection methods. First, the processing speed is much higher than models in the R-CNN family, even reaching and surpassing real-time performance levels. In addition, a lighter and speed-optimized version, known as Fast YOLO, has also been developed.

The second advantage the reduced false detections in the image background because they use features of the entire image to predict each bounding box, taking into account the context of the objects in the image and detecting fewer objects incorrectly.

The third advantage is that YOLO models learn generic representations of objects. Since they are highly generalizable, these models are more robust when applied to new domains and unknown images for analysis. However, it should be noted that these models also have disadvantages and limitations. They are generally less accurate than other objects detection methods, especially when it comes to detecting small objects.

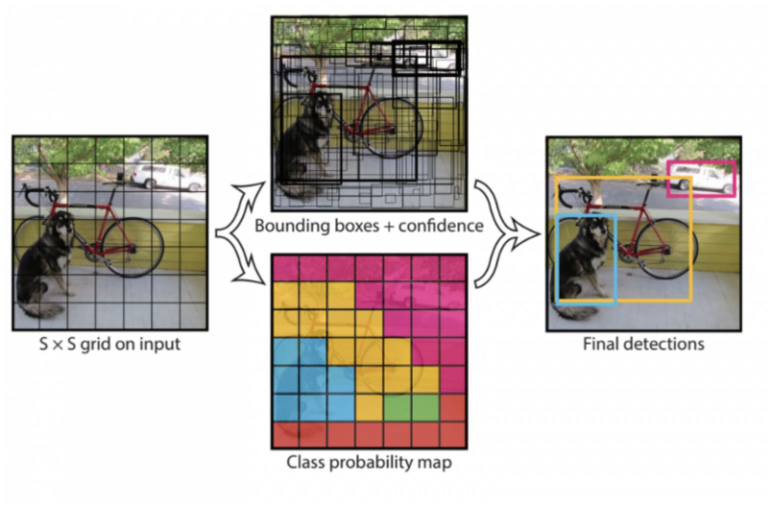


Figure 2.6: Main idea of YOLO [36].

The evolution of YOLO continues after its first release [36]. The second version [38] was proposed to be faster and more accurate than the first version, using anchor boxes, dimension clustering, and multilevel training. The release of the third version [39] inproved prediction, due to the introduction of 3 different scales it solved the shortcoming in the previous version, but led to a loss in speed, due to the process becoming more expensive of processing. The fourth official version [40] improved inference and accuracy, due to better efficiency in running on Graphics Processing Unit (GPU), occupying less memory. This last version proved to be the most efficient in real-time object detection using the metrics of the [41] metrics. This method had its Average Precision (AP) and Frames per Second (FPS) improved by approximately 25% and 22%, respectively, as shown in the table 2.1.

SSD

To address the challenges posed by small object detection and the various sample reduction operations in YOLO, SSD [37] was developed. Instead of using fixed grids as in the previous method, this approach employs anchor boxes with varying scales to generate the output BBs, combining multiple predictions and resolutions.

This method is based on the VGG16 backbone architecture with the addition of two convulational layers at the end of the network, predicting displacement of boxes at different scales, as presented in Figure 2.7. Its training is based on information reduction and the final result obtained by Nonmaximum Suppression (NMS). It achieves 77,2% mAP on VOC2007 at 46 FPS on an Nvidia Titan X. However, the result for small targets is poor, and the feature maps of different scales are independent, leading to simultaneous detection of the same object by different sized boxes.

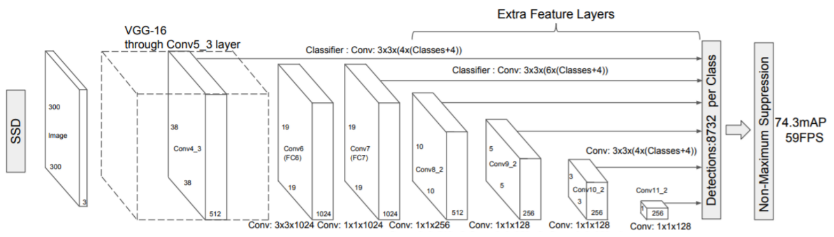

Extra Feature Layers
VGG-16
through Conv5_3 layer
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
SSD
300
Image
300
3
38
Conv4_3
38
512
19
Conv6
(FC6)
19
1024
19
Conv7
(FC7)
19
1024
10
Conv8_2
10
512
5
Conv9_2
5
256
Conv: 3x3x(4x(Classes+4))
3
Conv10_2
3
256
Conv11_2
1
256
Conv: 3x3x1024 Conv: 1x1x1024 Conv: 1x1x256 Conv: 1x1x128 Conv: 1x1x128 Conv: 1x1x128
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP
59FPS

Figure 2.7: Architecture of SSD 300[37].

2.1.3 Analysis of the methods

The results announced by the authors of the various models are usually associated with the datasets used [42]. It is because, over the years, several competitive challenges for object detection were organized, and typically the datasets used to evaluate the competing methods were predefined. Since then, these datasets have continued to be used to evaluate newly developed models. The metrics considered in these challenges are not always the same. Thus, it is common for each dataset to be associated with specific metrics. Despite all this variability, some metrics are always relevant, such as accuracy, revocation, and IoU.

It is possible to locate objects in images through bounding boxes with different qualities and processing speeds, depending on the available resources and the model used. No single model that is best in all cases, and it is necessary to evaluate the needs of each application, such as time, quality, accuracy, and precision. R-CNN models offer more accurate results, while YOLO models prioritize speed over quality. The SSD model is an intermediate

Method	**Backbone**	**Size/Pixel**	**Test**	**mAP/%**	**fps**
YOLOv1	VGG16	448×448	VOC 2007	66.4	45
SSD	VGG16	300×300	VOC 2007	77.2	46
YOLOv2	Darknet-19	544×544	VOC 2007	78.6	40
YOLOv3	Darknet-53	608×608	MS COCO	33	51
YOLOv4	CSP Darknet-53	608×608	MS COCO	43.5	65.7
R-CNN	VGG16	1000×600	VOC2007	66	0.5
SPP-Net	ZF-5	1000×600	VOC2007	54.2	-
Fast R-CNN	VGG16	1000×600	VOC2007	70.0	7
Faster R-CNN	ResNet-101	1000×600	VOC2007	76.4	5

Table 2.1: Comparison of Object Detection Algorithms [42].

option, offering higher speeds than the R-CNN models and equal or better quality than the YOLO models.

The YOLOv4 was chosen to implement the application because it is the latest and most improved version of the YOLO family of models. Several models have been studied to locate objects through bounding boxes, but the performance in terms of speed and quality of detections has not been analyzed in an absolute way due to the characteristic variables that involve the process.

2.2 Artificial Vision

Machine vision is one of the most active and promising areas of artificial intelligence. It involves the development of algorithms and models that enable machines to understand and interpret images and videos like humans. Machine vision has applications in many fields, such as medicine, robotics, security, and entertainment [43].

In recent years, there have been significant advances, driven by progress in the development of machine learning algorithms [44] and the availability of large labeled datasets for training [45]. Some of the major areas of advancement in machine vision include:

- Object detection and classification: Object detection and classification is one of the fundamental tasks of machine vision [14]. In recent years, convolutional neural networks (CNNs) have been widely used for object detection and classification [24].

Algorithms such as YOLO, R-CNN, and their variants have achieved impressive results in this area [42].

- Image segmentation: Image segmentation is another important task of machine vision, which involves dividing an image into several regions or segments [46]. Recent advances in image segmentation include semantic segmentation, which involves assigning semantic labels to each pixel in an image.
- Face recognition: Face recognition is an important area of machine vision, which has applications in the security and identification of individuals [47]. Face recognition has been improved using deep neural networks such as FaceNet and DeepFace.
- Self-driving: Self-driving is another important area of machine vision, which involves detecting and classifying objects in real time to allow vehicles to move autonomously [48]. Machine vision technologies are a key part of developing autonomous driving systems [6].

Machine vision constantly evolves, and recent advances in machine learning algorithms and data sets have driven its development. The applications of machine vision are vast and range from object detection and classification to self-driving cars. Machine vision will continue to be an active area of research and development in artificial intelligence.

2.2.1 Localization

Object localization, through distance estimation using images, is a widely used technique in many machine vision applications [49], such as autonomous vehicles, robotics and security systems, among others. In recent years, there have been significant advances in the state of the art of using images to measure the distance of objects, thanks to the evolution of [50] technologies such as cameras, sensors, and [51] image processing algorithms.

Some of the main methods for measuring the distance of objects using images include:

- Stereo-vision: Stereo-vision is a technique that involves using two cameras to capture images of an object from different points of view. Distance is measured by comparing the differences between the two images [52]. This method is widely used in robotics and autonomous vehicle vision.

- Machine Learning Algorithms: Machine learning algorithms such as Convolutional Neural Networks (CNNs) have been used to measure the distance of objects based on images. These algorithms are trained on a large set of image data and can detect patterns that indicate distances [53].

The use of images to measure the distance of objects has advanced considerably in recent years, thanks to the development of technologies. Often, it is possible to combine several techniques to measure the distance of objects more accurately. For example, stereo-vision can be combined with machine learning algorithms to improve the accuracy of the measurement. This possibility of coupling continues to allow systems to reach new levels of accuracy and efficiency.

Work was analyzed that calculates the distance from specific points to a robotic platform [17], which uses stereoscopy to avoid platform collision. The chapter also includes an analysis of object detection in robot soccer competitions [54], as well as the use of monocular vision to measure the distance and height of horizontal edges in mobile robots [55]. All these works used cameras to obtain images of the real world for subsequent analysis. For this, they used trigonometric analysis of the data, mainly related to the Pinhole model. Thus, this work will be based using a camera to acquire the images, obtaining the data through machine learning for object detection and analyzing the data through mathematical modeling of the Pinhole model.

Chapter 3

Methodology

The upcoming chapter will present the problem formulation and a proposed solution. Then, it will introduce the concepts used in the development process. Finally, the architecture of the detection and localization module will be presented, including all the constituent parts.

3.1 Definition of the Problem

There are several ways to approach the localization problem for a line following robot. Some ordinary methods include: inertial markers, visual markers, GPS, and SLAM. The vast majority of the robots competing in Robot@Factory Lite have used mainly encoders and infrared detection sensors to locate themselves and reach the target object. Regardless of the method chosen, it is important to regularly check the accuracy of the methods used and make the necessary adjustments to ensure that the robot stays on track. However, in certain specific cases, the information from these sensors alone is unnecessary for error correction. Examples are when the robot leaves the predefined route and/or is displaced by external factors and when the route is obstructed by the insertion of unknown items. Regarding the evolution of the race, the target objects (boxes) can be positioned in previously unknown locations on the competition track. Thus, the need arises for a detection and localization system that circumvents these problems, detecting objects and estimating parameters such as distance and angular deviation, providing information for further implementations in conjunction with robots developed for competition.

3.2 Concepts

3.2.1 Digital Image

The image can be presented as a representation or storage of visual perception. The computer processes discrete data represented in binary form. Therefore, the images must be digitalized for their processing. Continuous images are spatially discretized and quantified in intensity, represented by $I(x, y)$. The matrix indices, row and column, identify a point in the image, and the corresponding value (I) of the matrix element identifies the gray level at that point, assuming value and a finite interval. The elements of this digital matrix are called [18] pixels. The more picture element (pixels) the image has, the better its resolution and quality, however, this entails an increase in its processing time. Thus, images only capture information about the amount of light, and the spatial information about the direction of the light rays is lost.

3.2.2 Pinhole Model

o utilize images and aid in robot localization, it's crucial to comprehend the geometric principles underlying the formation of the image captured by a camera. By analyzing the data present in the image, it becomes feasible to establish the relative positioning of objects concerning the camera's position. The main focus is to be able to transform and project the real 3D world into a 2D image, but this entails a loss of depth information. But there are some ways to get around this problem, such as using multiple cameras, multiple images from a camera or the relationship between the points present in the image and the objects that make up the scene.

The model used with greater recurrence linked to computer vision is pinhole [18], which uses perspective projection, this model is presented in figure 3.1. The projected image, plane I, is composed of all the rays that pass through a small hole at the entrance to the camera, called the focal plane F. The distribution of pixels is defined by the focal length, generally, patterns in images are intrinsic to the camera or obtained through calibration.

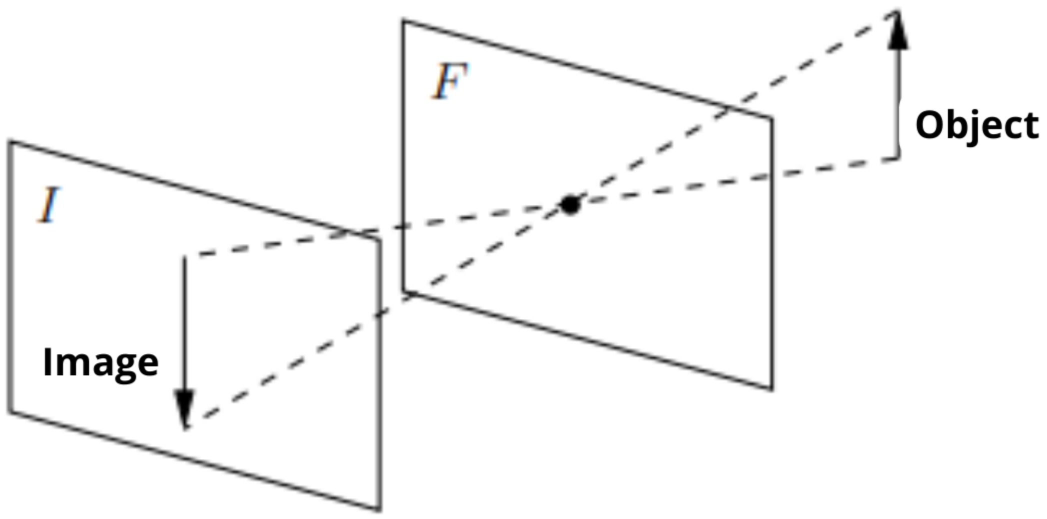


Figure 3.1: Image formation in a perspective camera adapted [18].

The analysis of a point in the real world P_c on an image point P_i, is presented in figure 3.2. The point P_c its coordinates (X_c, Y_c, Z_c), defined by the camera reference C . The point P_i has coordinates (x_i, y_i) defined in the reference frame of image I. The point C represents the optical center, whose distance to the reference frame I is λ of the focal length. The line passing through the optical center and perpendicular to the I plane is the optical axis.

Using trigonometry and the relationship of triangles, the relationship between the coordinates of the points in the two reference frames are given by

$$x_i = \frac{\lambda}{z_c} x_c, \tag{3.1}$$

where

$$y_i = \frac{\lambda}{z_c} y_c. \tag{3.2}$$

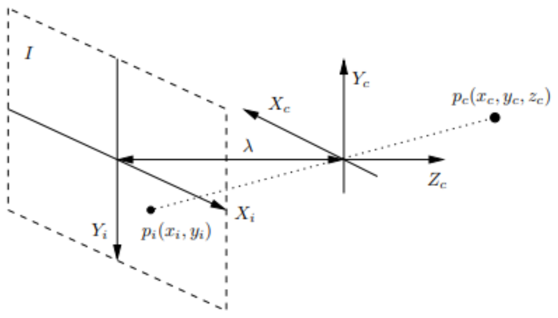


Figure 3.2: Camera perspective projection model[18].

The formulation to calculate the relationship between objects size using pinhole is similar to the one used to calculate the relationship of points. The RaspiCam is equivalent to a camera with a rectilinear lens, not deforming the objects in the image and preserving their characteristics. From the camera specifications, one can consult the Horizontal Field of View (hFoV), the Vertical Field of View (vFoV) and the focal length, which are related, as shown in Figure 3.3.

Given that the field of view angles of the actual image and the projected image are identical,

$$\theta_o = \theta_i \tag{3.3}$$

and

$$\frac{\theta_o}{2} = \frac{\theta_i}{2}, \tag{3.4}$$

$$tan(\frac{\theta_o}{2}) = \frac{\theta_i}{2}. \tag{3.5}$$

But the tangent relates to the sides of the triangle as

$$tan(\theta) = \frac{C\_opposite}{C\_Adjacent}. \tag{3.6}$$

Then

$$tan(\frac{\theta_o}{2}) = \frac{o}{2} \tag{3.7}$$

and

$$tan(\frac{\theta_i}{2}) = \frac{i}{2}. \tag{3.8}$$

Like this

$$\frac{o}{2 * d} = \frac{i}{2 * f} \tag{3.9}$$

and

$$\frac{d}{o} = \frac{f}{i}. \tag{3.10}$$

By using these formulas and parameterizations, it is possible to establish relations between the sizes of objects in the real world, measured in centimeters, and their corresponding sizes in the image, measured in pixels.

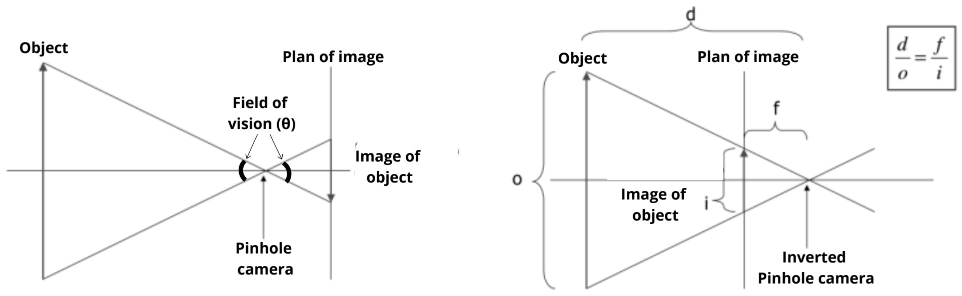


Figure 3.3: Pinhole geometric relations adaptation of [56].

Several works are based on the use of camera vision, through focal length, to locate objects and know the distance between the camera and the object. [55] presents a method to spatially locate objects detected in the central column of images obtained by a moving camera, restricted to the same direction as the optical axis, and at a constant speed, as frequently occurs in certain mobile robots. The analysis of visual odometry is presented in [57], among other methods based on sensory fusion for location in mining fields.

The localization system will be based on the pinhole camera model. It differs from analog camera models, in addition to not having a lens, as it does not have a shutter, which is the mechanism that controls the exposure time of material sensitive to light rays. Despite being based on simple construction, they work perfectly by respecting the physical laws of image formation, obtaining an inverted image because the illuminated objects emit light in several directions, and the small hole in the camera only allows a few rays to pass through. As the rays fall in a straight line, those that fall at an angle have their inverse projection. This model describes the relationships between the objects dimension in the real world and its projection on the image plane.

3.3 Solution Components

3.3.1 Hardware

The proposed hardware to create the localization module, presented in the Figure 3.4, was based on related works presented in [58], [59],[60], [61], [62], [63],[64],[65] and [66]. The main components and specifications taken into account were:

- **Computer Platform** Raspberry Pi 4 Model B - 4 GB RAM:
  - Fast CPU;
  - Image/video processing and neural networks;
  - Adaptive;
  - High hardware and software compatibility and
  - Low cost (92 euros).
- **Camera** Raspberry Pi Cam Rev 1.3 plus flexible apartment cable:
  - Compatibility with own connector on Raspberry;
  - Light and compact module and
  - Low cost (7,57 euros).

Raspberry Pi 4 Model B

This device offers flexible memory options, with configurations of up to 4 GB of DDR4 Random Access Memory (RAM). It is powered via a USB-C port and features a micro

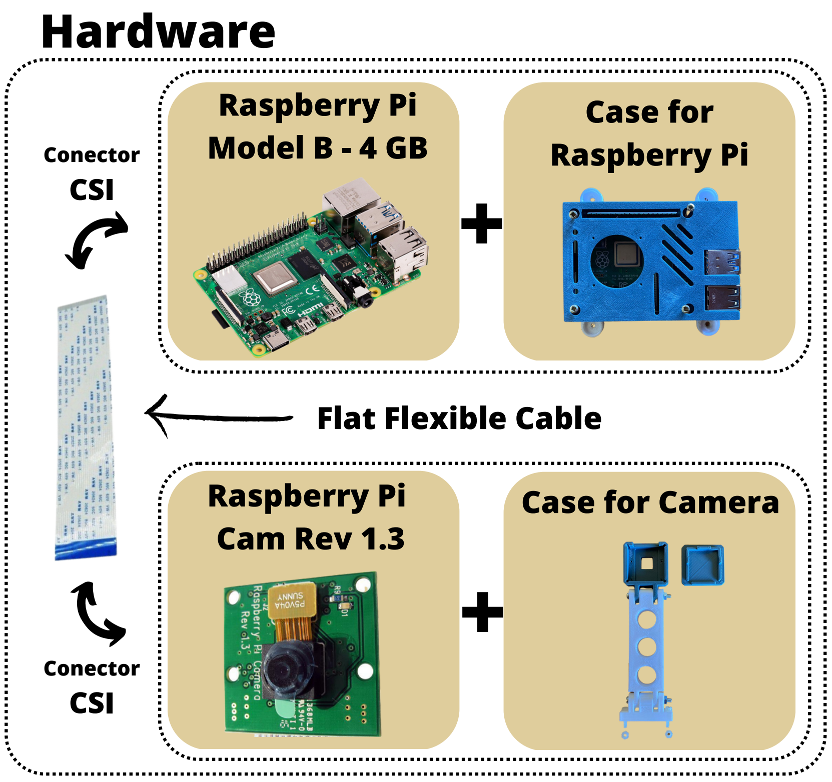


Figure 3.4: Hardware Module

HDMI output that supports up to 4k dual-display monitors. Additionally, it offers enhanced connectivity and extended GPIO, as presented in Table 3.1 and Figure 3.5, all while remaining affordable.

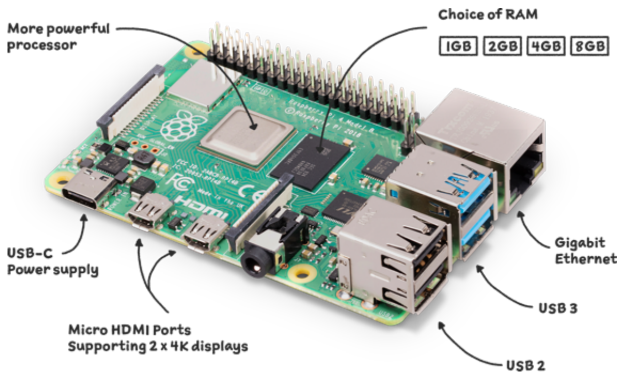


Figure 3.5: Raspberry Pi 4 model B [67]

Specifications	**Quantity**
Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz	1
1GB, 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)	1
2.4 GHz and 5.0 GHz IEEE 802.11ac wireless	
Bluetooth 5.0	
BLE Gigabit Ethernet	1
USB 3.0 ports	2
USB 2.0 ports	2
Raspberry Pi standard 40 pin GPIO header	
micro-HDMI ports (up to 4kp60 supported)	2
lane MIPI DSI display port	2
lane MIPI CSI camera port	2
4-pole stereo audio and composite video port	
Micro-SD card slot for loading operating system and data storage	1
5V DC via USB-C connector (minimum 3A*)	1
5V DC via GPIO header (minimum 3A*)	1

Table 3.1: Raspberry Pi 4 Tech Specs [68].

This small, powerful and extremely affordable computer is ideal for embedded projects, using cameras and much more. Just connect it to a computer monitor or television, keyboard and mouse, add a Secure Digital (SD) card with the system installed, and it is ready for use, as shown in Figure 3.6.

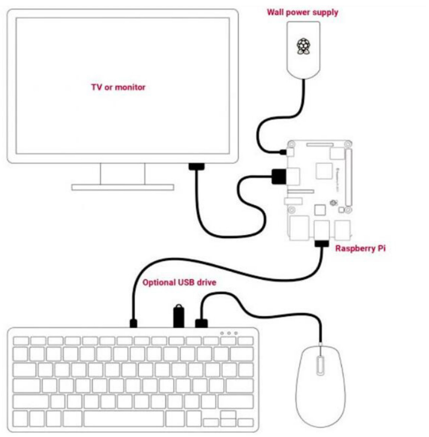


Figure 3.6: Raspberry Pi utilization with accessories [69]

Raspberry Pi Cam Rev 1.3

Raspberry Pi Cam Rev 1.3 module is mainly used in computer vision works like [63],[64],[65] and [66] as it is easy to access and customized for use on various versions of the Raspberry Pi. It is compatible with the latest version of Raspbian, the main operating system used on the Raspberry Pi.

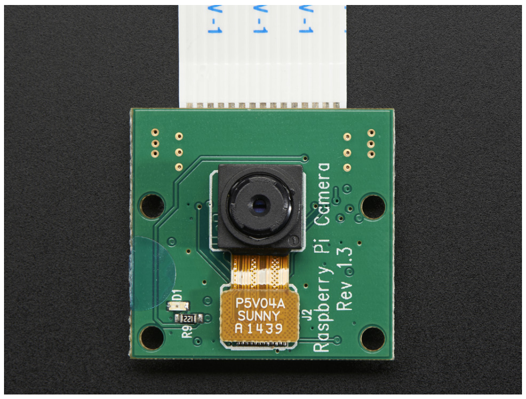


Figure 3.7: Raspberry Pi Camera Board [70]

The dimensions of the board are shown in Figure 3.8 and its weight is approximately 3 grams. These specs make this module perfect for placement on mobile devices where size and weight are important.

Its uses vary from recording videos in high quality, to taking photos and creating tyme

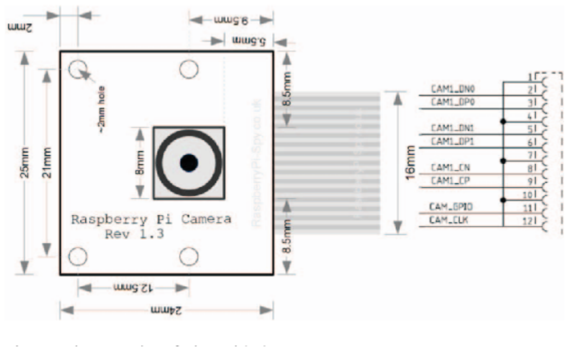


Figure 3.8: Raspberry Pi Camera Board [70]

lapse files.With its built-in fixed focus lens, this device is capable of capturing crystal-clear pictures at 5 megapixels megapixels (MP), with still images of 2592 x 1944 pixels. It also supports video recording in 1080 pixels at 30 frames per second frames per second (fps), 720 pixels at 60 fps, and 640 x 480 pixels at 60/90 fps [71].

Case for Raspberry and Cam

The Raspberry Pi 4 Case and the support for the Raspberry Pi Camera were manufactured through 3D printing, using polylactic acid (PLA), as shown in Figure 3.9. They bring a safer design to the project, offering more security and stability to the circuit boards. The case guarantees a perfect fit and compatible design for the project.

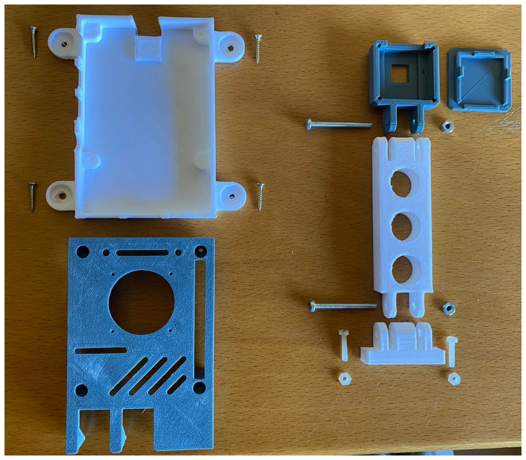


Figure 3.9: Cases for Raspberry and Camera

The Raspberry board is attached to the case with screws and has a cover with a perfect fit to keep the board wholly sealed, with only the side connectors available for access and use. The camera module is securely attached to the device's case and connected to the Raspberry Pi via a flat cable that passes through the designated slot in the case. These two cases are joined together by a connecting rod and screws. In this way, it becomes malleable.

3.3.2 Software

The development of the localization system, occurred through the Operational System (OS) Raspbian interface. The neural network training, required to obtain the configuration weights was performed solely on Google Colab. This platform provides a Jupyter notebook running on a GPU and is free to use. To train the network, GPU and cuDNN were used with specific drivers installed, and OpenCV. The coding of the localization program, was developed entirely in Python programming language through the Thonny interpreter, native to the Raspbian interface. For execution, all dependencies were installed in our system: PiCamera and OpenCV. The algorithm for detecting objects and calculating their coordinates will be executed directly on the Raspberry, using the OpenCV tool. The schematic of the software and algorithms is shown in Figure 3.10.

Operating System - Raspbian

An operating system glsOS is a collection of programs that provide an interface for users to interact with the computer. There are numerous glsOS options available on the market, including Windows, GNU/Linux, and MacOS, which are primarily used in desktops/notebooks. Raspbian is a popular OS based on Debian that is specifically designed for the Raspberry Pi. Mobile devices also have their own operating systems, such as Apple iOS, Windows Phone, and Android.

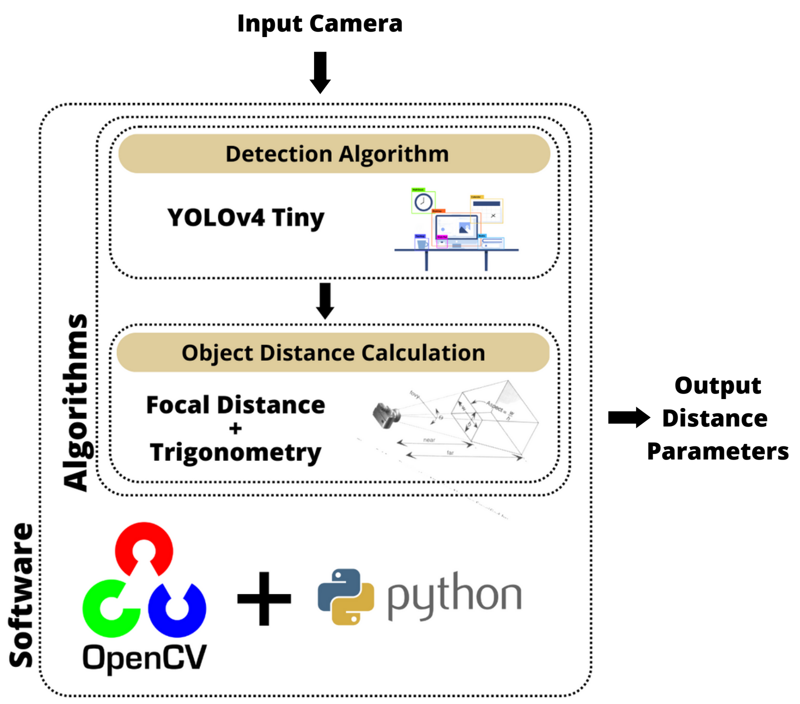


Figure 3.10: Software diagram and algorithm

Integrated Development Environment - Thonny

Thonny is an Integrated Development Environment (IDE) with simple language and interpretation for beginners in aprogramming language (Python) [72]. Its use ranges from being used to create simple programs to creating applications. It makes programming acessible, like its similar Microsoft Visual Studio [73] or NetBeans IDE [74], as it has several tools , libraries, and dependencies for the most part already incorporated.

When executing the program, the programmer can simultaneously see the execution and the code together, facilitating the analysis. Also, it is possible to see the code, the values of the variables, and their change in real time. It has an integrated automatic debugger, and the completion code, which displays the errors on the screen. Its main advantages are that it is free, open source, intuitive, and focused on the beginner, with a high visual experience. One of the main drawbacks is its limited scope and support for Python. Since June 2017, it has been included by default in the official Raspberry Pi operating system [75].

Programming Language - Python

Programming language is writing in a structured way that helps to specify a set of instructions and rules used to generate programs (Software). The software can be developed to run on computers, mobile devices, embedded systems and any other devices that allow code execution. There are many programming languages, such as Java Script, Java, Python, C, C++, among others [76].

Python will be used to develop the project because it is cross-platform, simple, efficient, free, has mature code, consistent updates, a large bank of ready-made functions, and is compatible with several devices, mainly because it is the language used in programming in Thonny (Section 3.3.2).

Open Source Computer Vision Library (OpenCV)

An open source computer vision and machine learning library that provides a common infrastructure for machine perception applications. With over 2,500 optimized algorithms, the library includes a comprehensive set of classic and state-of-the-art algorithms. It can be used in various of applications, from street view image stitching to fast face and object detection. It has C++, Python, Java, and MATLAB interfaces and supports various operating systems, such as Windows, Linux, Android, and Mac OS. The library is handy for real-time vision applications and consists of more than 500 algorithms and about 10 times more support functions for focusing on this area. For this work, OpenCV version 4.6 was used in the Linux environment.

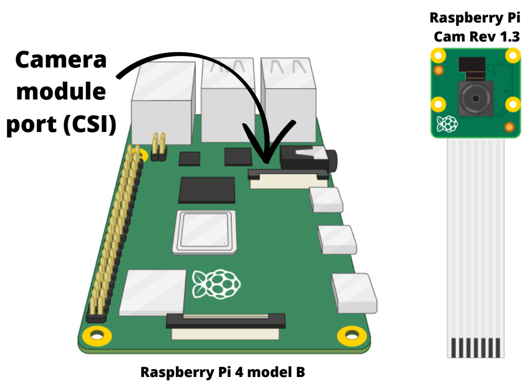
Chapter 4

Development

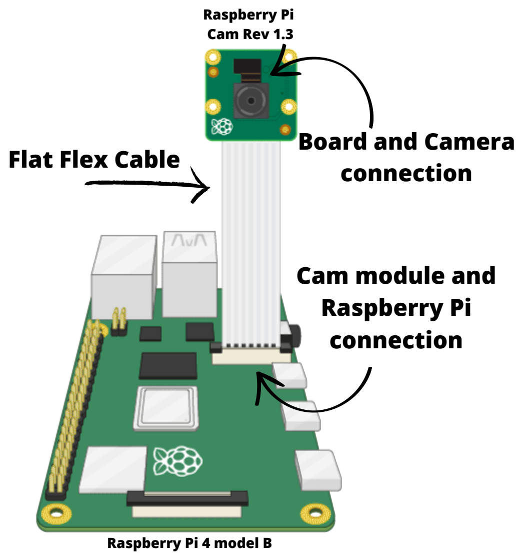
4.1 Module Assembly

The connection of the components is simple and easy to implement, as shown in figure 4.1. The Raspberry Pi connects to the camera via an cable between the CSI ports. The union of these components forms the localization module. This module can be embedded in any robot to obtain its location, with the main focus on solving competition problems. The module can still be adapted and expanded for implementation in environments beyond the competition, such as industries, hospitals, among others.

For purposes of protection, handling, and coupling of the module, before connecting the cable between the components, they must be fitted in the case. Then the flat cable can be connected to the Raspberry and the camera. The complete module can be seen in Figure 4.2.

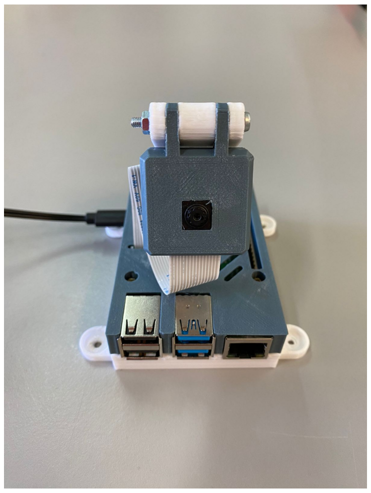


(a)

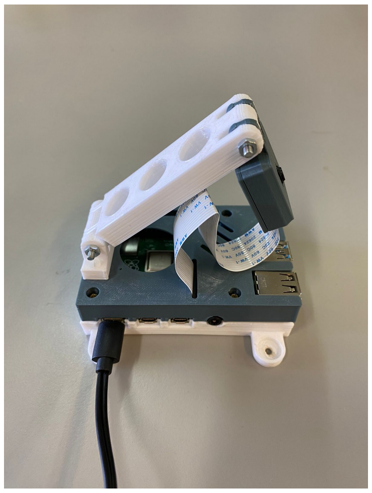

Raspberry Pi Cam Rev 1.3
Flat Flex Cable
Board and Camera connection
Cam module and Raspberry Pi connection
Raspberry Pi 4 model B

(b)

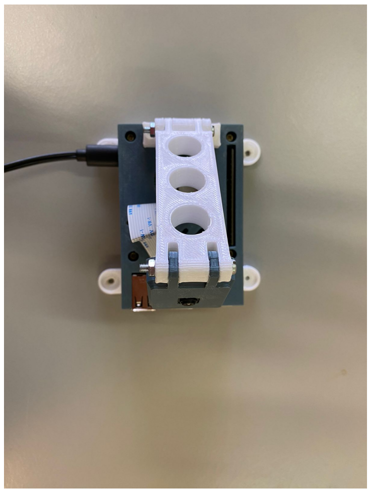
Figure 4.1: Raspberry Pi e Module Raspi Cam, adaptation of [77]. (a) Separate components e (b) Connection of components



(a)



(b)



(c)

Figure 4.2: Location module prototype: (a) Front View, (b) Side View, and (c) Top View

4.2 Location Module Algorithm

4.2.1 Detection

Configurations Net and Model Detection

The algorithm starts by importing the libraries needed to execute all developed code use in the module. They are presented, along with their respective functionalities, in Figure 4.3. Next, as shown in Figure 4.4, the network configurations should be performed, which will later be used in the detection model. Finally, this data and model will be used in the object detection function.

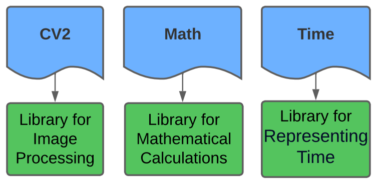

CV2
Math
Time
Library for Image Processing
Library for Mathematical Calculations
Library for Representing Time

Figure 4.3: Specifications used libraries.

The explanation of our detection model setup, represented by the flow chart in figure 4.4, will be presented next. The first step is to insert the paths of the necessary file folders, which were previously used in training performed on Google Colab, to obtain the weights. The initial path should point to the folder containing the names of the classes to be detected, followed by the paths of configuration and weights. These parameters are necessary for the network's configuration.

With the paths prepared, the configuration operations begin. The folder that contains the names has its files read and stored in a vector (Class Names) through the code. When storing data, blank spaces at the beginning and end of each word are removed, and the names are separated into individual positions within the vector. In the second part, the network configuration takes place. For this, the functions of the CV2 library are used, which presents an API, to build, modify and use models with different types of structures. The network is created using the command for reading binary and text files containing the

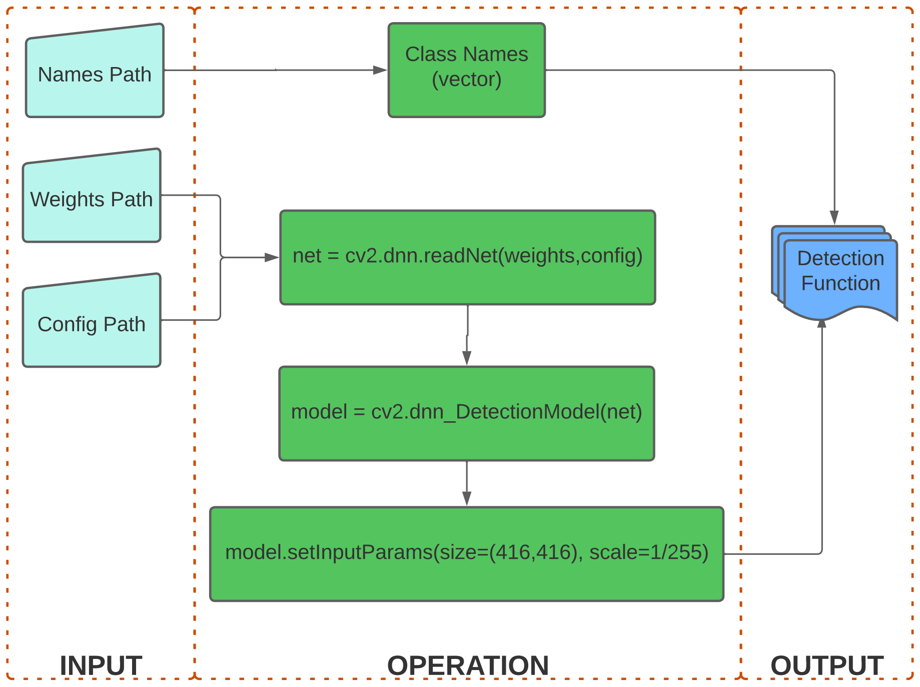

Names Path
Weights Path
Config Path
Class Names (vector)
net = cv2.dnn.readNet(weights,config)
model = cv2.dnn_DetectionModel(net)
model.setInputParams(size=(416,416), scale=1/255)
Detection Function
INPUT
OPERATION
OUTPUT

Figure 4.4: Network configuration flowchart and detection model.

weights and settings, respectively. The extensions (.weights and .cfg) and Deconvolutional Neural Network (DNN) required by the model are provides by YOLO, which uses the Darknet framework. After passing the network as an argument for creating the model in the algorithm, the input frame's size and scale settings are specified. The resulting name vector and detection model are subsequently utilized by the object detection function.

Detection Function

The object detection function has its operation flowchart presented in the Figure 4.5. The object detection function is invoked in the main function with three parameters: the frames captured by the camera, the object accuracy threshold, and the non-maximum suppression value, which is used to eliminate multiple bounding box predictions for the same object. In addition, a list is created in the arguments passed to the function, which will store the name of the objects to be detected.

The object detection function first invokes the detection model with the parameters specified in the function call. If no object is detected, the respective variables will store the class identification, accuracy, and the coordinates of the bounding box. If no object is found, the variables will be empty. At the same time, if the list of objects has no names stored yet, the names of the classes to be identified will be passed. Also, a list is created that will store information regarding the detection of possible objects found.

After calling the model and creating the necessary variables, the next step is to verify if any object was found, testing if the class identification variable has a non-zero stored value. If any value is found, three lists will be created storing, respectively, the identification, accuracy, and detection box values of all the objects. Then, with the identification of the objects, it is verified to which class the object belongs, in this case having only one class (box), then all objects found will belong to the same class. Finally, if the object belongs to the target class, all information will be stored in a variable created for this, the function will return the frame and the objects information. Otherwise, the function returns the frame and the value zero for the information.

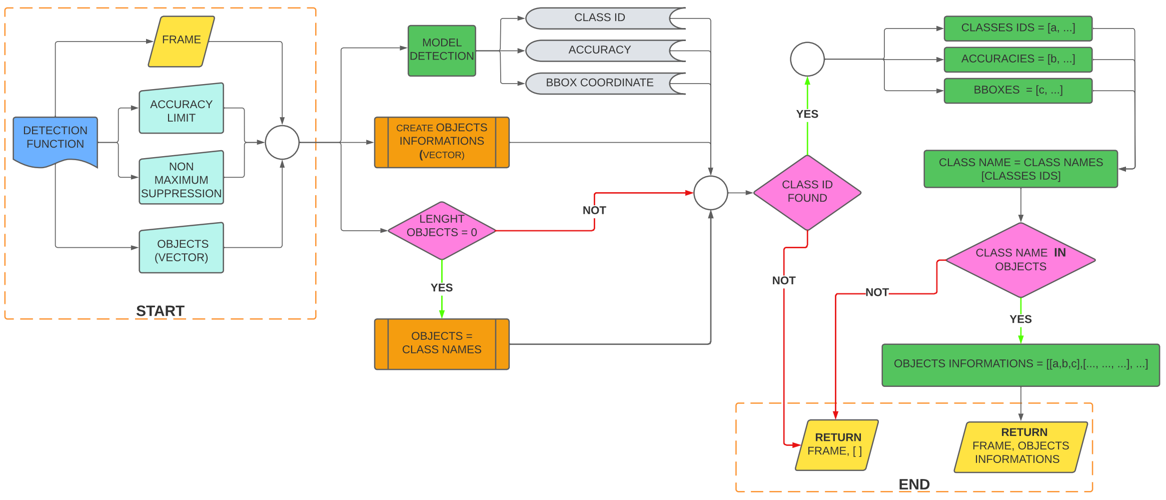


Figure 4.5: Flowchart of the detection function.

4.2.2 Coordinate Location

The estimated calculation of the location coordinates, distance, and orientation angle, was based on the Pinhole camera, as in Figure 4.6. It can be seen that most of the parameters are kept constant, such as the maximum size the image can be, according to the camera resolution setting, producing a specific output image according to the user's selection. The camera's angle of view and focal length also remain constant. The only pattern that varies is the distance between the object and the camera, leading to changes in the size of the object in the image. Based on this information, the distance and the angle of orientation will be calculated.

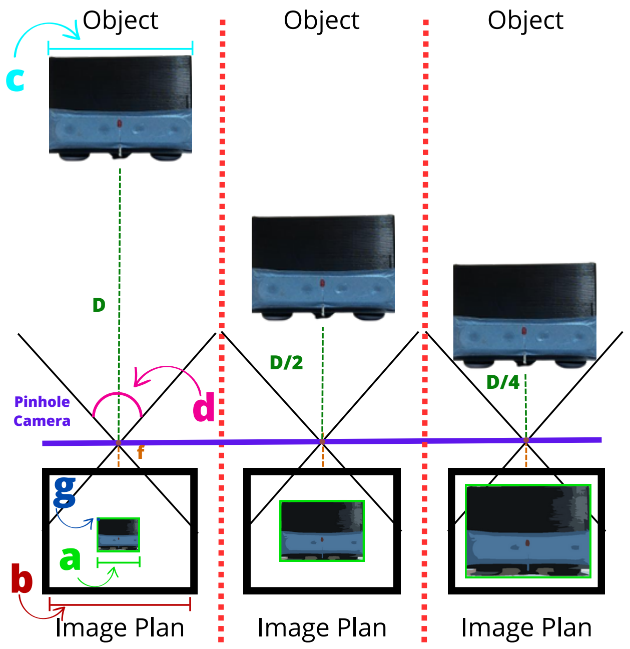


Figure 4.6: Distance and angle variation based on the Pinhole model.

Distance

The distance between the object and module is obtained using the camera parameters, which are focal length, hFoV, and image resolution, all related to each other. As these values are fixed, based on what was presented in section 3.2.2 and in the equation 3.10, the only parameter changed is the distance between the camera and the object. However, this change leads to a change of the size that the image in the object occupies in the final image. These values can be related to each other by a rule of proportionality. Thus, the calculation of the distance is based on the variation in the width of the object in the image.

The algorithm uses the information of the resolution (width), the actual size of the object (width) and the hFoV, entered by the user and represented respectively by the letters "b", "c" and "d", in the figure 4.7. The value provided by the object detection function is also used, with this parameter being the width of the detection box formed, represented by the letter 'a", in figure 4.7. Using the trigonometric relations for a right triangle, considering that the center of the object and camera of the module are aligned, through the equation 3.10 and substituting the value of the focal length for the angle hFoV, the value of the distance is calculated.

The representation of the object is made in 2D, losing the information of depth. Therefore, it is assumed that the object is always aligned parallel to the center of the module. But this value can be corrected after the calculation of the alignment angle between the centers. So, whenever the distance value is found, its value is used to find the angle of alignment between the centers. With this value, the distance is converted to an angular offset by dividing its original value obtained by the cosine of the value returned by the angle calculation function. In this way, a value closer to the real one is obtained.

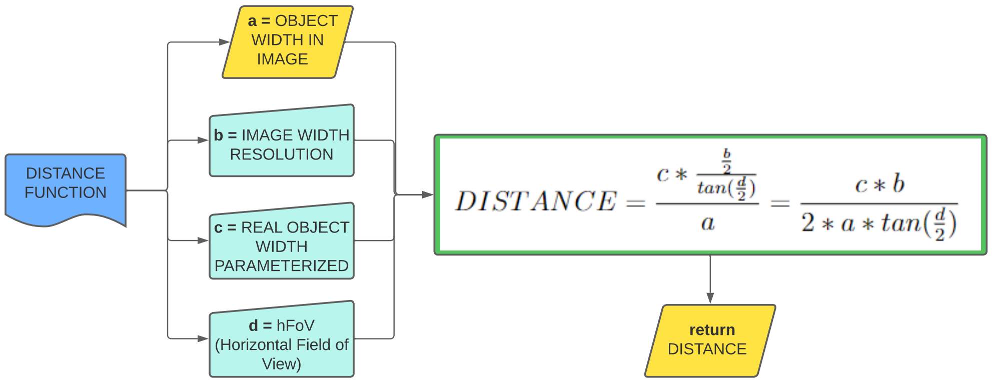

DISTANCE FUNCTION
a = OBJECT WIDTH IN IMAGE
b = IMAGE WIDTH RESOLUTION
c = REAL OBJECT WIDTH PARAMETERIZED
d = hFoV (Horizontal Field of View)
$DISTANCE = \frac{c * \frac{\frac{b}{2}}{tan(\frac{d}{2})}}{a} = \frac{c * b}{2 * a * tan(\frac{d}{2})}$
return DISTANCE

Figure 4.7: Flowchart of the distance calculation function

Orientation - Angle

The center offset angle, between the object and the module, is obtained based on the positioning of the object's center in the image. There can be three possibilities: where the object are aligned, offset to the right, or offset to the left. This identification is made for possible future implementations. Finally, assuming the object is always aligned with the module, relative to the y-axis, the angle is calculated using trigonometric relations.

The algorithm uses the information of the resolution (width) and the actual size of the object (width), entered by the user and represented respectively by the letters 'b' and 'c', in Figure 4.8. Also used are the values provided by the object detection function, the initial position, on the x-axis, of the detection box and its width, and the return of the distance calculation function, represented respectively by the letters 'g', 'a' and 'D', also in Figure 4.8.

As already mentioned, the function is based on using trigonometric relations for a right triangle. The distance information is calculated, representing the adjacent catet. But as its value is obtained in the centimeter scale, a conversion is made to the pixel scale, using the value of the real size of the object and the resolution of the image. The opposite skew, on the other hand, is found by calculating the size between the center of the image and the value of the center of the object, with respect to the x-axis. To do this, the return value of the detection function is used with the parameters of the box, adding the value of its initial position, with its width divided by 2. The value of the center of the image is obtained by dividing the resolution by 2. Finally, the modulus of this equation is obtained, trying to find only the value of the skew, considering that the direction of rotation has already been identified previously. Thus, the angle is obtained by calculating the arc tangent of the division between the opposite and adjacent sides.

4.2.3 Main Function

The main function is starts by capturing the frame. To the detection function then it is sent, along with the parameters already mentioned in Section 4.2.1. If no object is found,

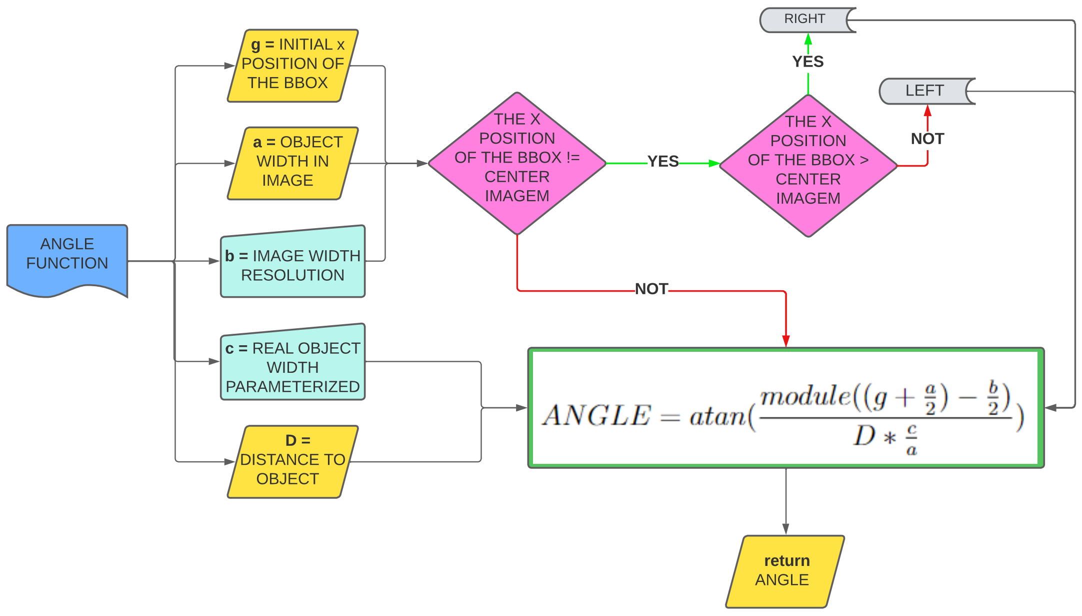

ANGLE FUNCTION
g = INITIAL x POSITION OF THE BBOX
a = OBJECT WIDTH IN IMAGE
b = IMAGE WIDTH RESOLUTION
c = REAL OBJECT WIDTH PARAMETERIZED
D = DISTANCE TO OBJECT
THE X POSITION OF THE BBOX != CENTER IMAGEM
YES
THE X POSITION OF THE BBOX > CENTER IMAGEM
YES
RIGHT
NOT
LEFT
NOT
$ANGLE = atan(\frac{module((g + \frac{a}{2}) - \frac{b}{2})}{D * \frac{c}{a}})$
return ANGLE

Figure 4.8: Flowchart of the angle calculation function

no value is returned and another frame is sent for analysis. However, if one or more objects are found, the function returns the coordinates of the detection box, the name of the identified class, and the accuracy.

Moving on to the localization phase, the necessary parameters, as mentioned in Section 4.2.2, are passed to the distance calculation function. With the distance, the algorithm calculates the angular displacement, as described in Section 4.2.2. At first it is checked in which position the object is with respect to the central axis. If the object is shifted from the system center towards the right or left, the angle is computed, which is subsequently utilized to apply a distance correction. Otherwise, there is no need for correction and, as described above, the angle is calculated. Finally, all information of interest is presented, being the class and accuracy, along with the distance and displacement angle, of all objects, if more than one is detected.

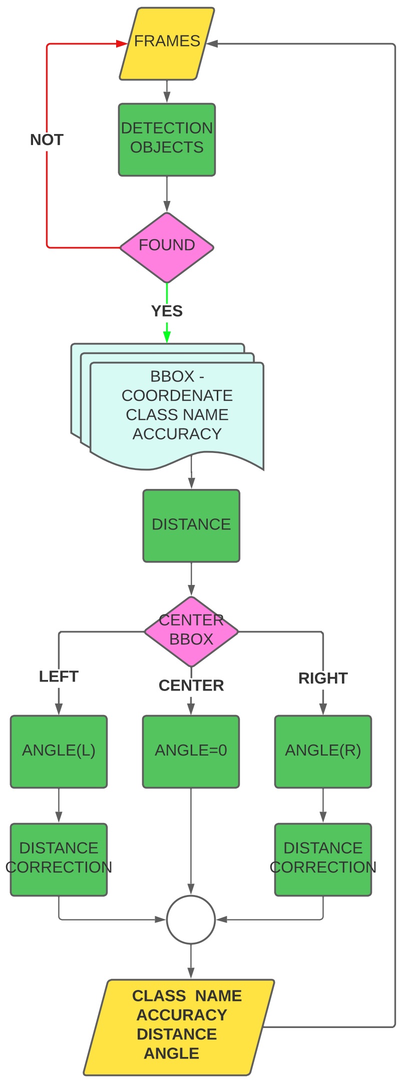


Figure 4.9: Flowchart of the main function

4.3 Detection of Objects

4.3.1 Creating the Dataset

The first step in training a network is to have a good dataset. For this experiment, one was initially created with 220 images of boxes used in the robot@factory Lite competition. These images were obtained using an iPhone 11 Pro, at the Electronic Systems Laboratory of the Polytechnic Institute of Bragança, using various backgrounds, orientations, and other objects together. The images obtained were in JPG format, with dimensions of 3024x4032 pixels.

The Roboflow online tool was employed to label the objects in the image and generate annotations in the dataset. This tool facilitates the task of computer vision in the field of deep learning. It was selected for its user-friendly features as it is free, requires no installation, and can be accessed through any web browser online.

The first step to using Roboflow is to create a free account. To begin, a workspace is created wherein an individual or a team of collaborators can produce, administer, annotate datasets, as well as train and utilize models. Thus, a new project was created with the following steps:

Creation Project

- Project Type: Object Detection (Bounding Box);
- What Are You Detecting: Boxes;
- Project Name: Boxes; and
- License: CC BY 4.0.

Upload and Annotate

- **File upload:** At this stage, the 220 images were uploaded and in JPG format. If we had the annotations file, these could have been uploaded together.

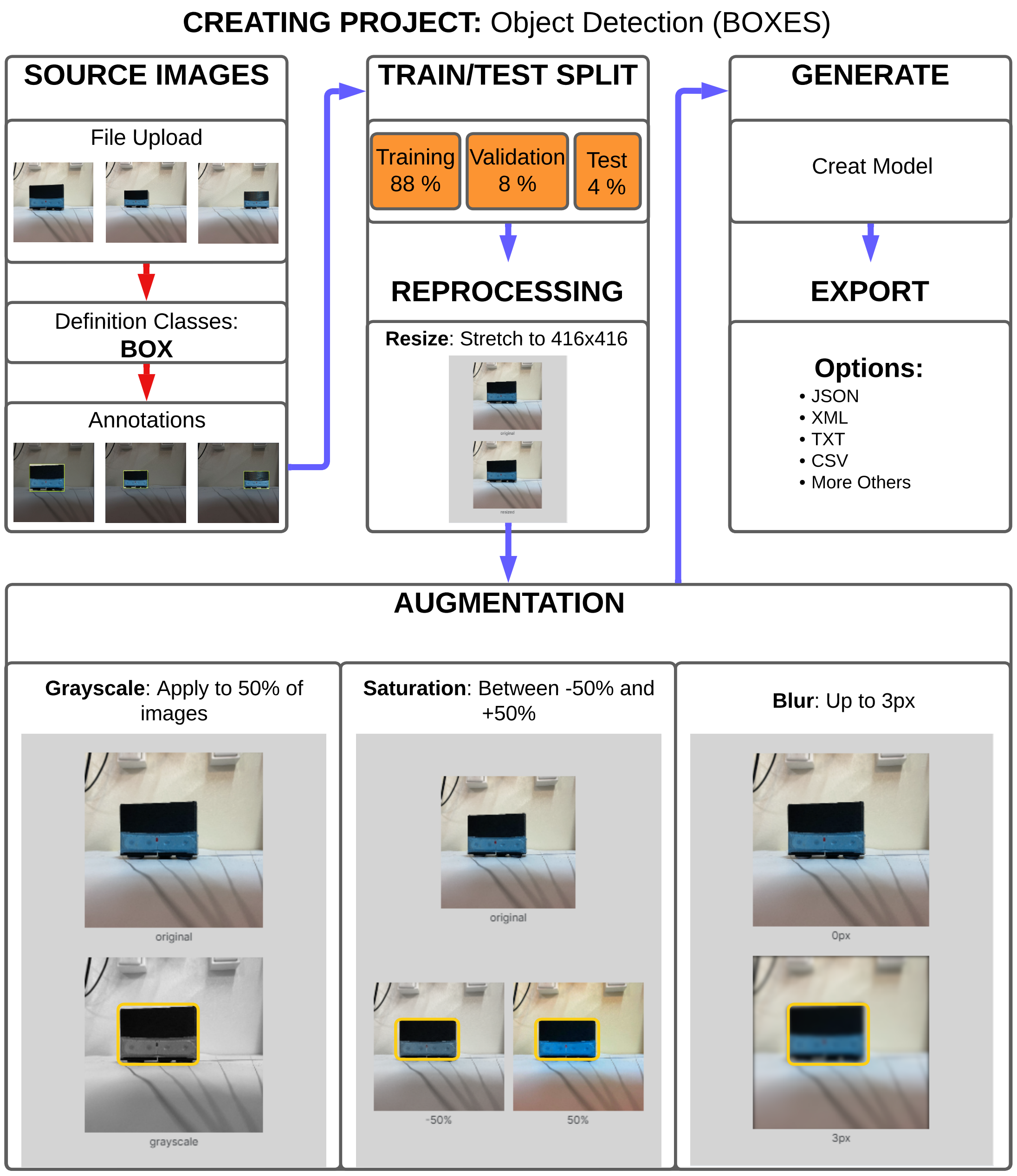


Figure 4.10: Creating customizing dataset.

- **Annotation of image:** As we did not have the annotations file for these objects, the objects were annotated, one by one, in each image. This part consists of creating rectangular boxes to demarcate where the object of interest is. On the Roboflow platform this is easy and can be done using the annotate tool. In this work, since only one object was annotated, the words for each class of annotated objects were generated, and for this purpose, the 'box' class was created. Subsequently, this markup is converted to a format, most commonly of the text type, with the coordinates of these contours.

- **File segmentation:** The annotated data is later separated into training, validation, and test folders. The number of files are chosen, in this case being separated into 88% for training, 8% for validation and 4% for testing.

Reprocessing

Reprocessing the images consists of making changes to our dataset to meet the input requirements of the training model. For this case, the modifications made were the application of self orientation and resizing the images, which initially had a size of 3024x4032 pixels. Finally, the application summary is presented:

- **Auto-Oriente:**Applied; and

- **Resize:** Stretch 416x416 pixels.

Expansion

The main problem, most often related to the dataset, is the small number of images used. Thus, image augmentation was performed, creating new images. The operations performed on the dataset are described:

- **Outputs per training example:** 3;

- **Flip:** Horizontal;

- **Crop:** 0% Minimum Zoom, 17% Maximum Zoom;
- **Rotation:** Between -15° and +15°;
- **Grayscale:** Apply to 25% of images; and
- **Saturation:** Between -41% and +41%.

At the end of the dataset expansionthe number increased from 220 to 564 images. In this way, the subsequent training of the detection method can be improved, through the greater amount of information.

Export Dataset

The end of dataset creation consists of exporting the data to be used. Roboflow offers two ways: downloading a zipped file or using a code Jupyter, via terminal or URL. It is also possible to choose the download file format, which can be JSON, XML, TXT, CSV, among others. In this work, a zipped file in YOLO Darknet format, with TXT files, was downloaded.

4.3.2 Custom Model Training

The integration of all the tools used to train of this custom model is shown in Figure 4.11. The explanation of the steps after creating the dataset are presented below. The script developed to obtain the weights for configuring the network was based on the model made available in the Roboflow platform tutorials for training detection methods with custom data [78]. The training of the model for box detection, based on YOLOv4 Tiny, was performed using the Google Colaboratory platform. This tool is free, runs online in the Google cloud, supports Python programming language, and provides GPU and TPU resources to be used.

Creating a Darknet environment was required, requiring OpenCV, Cuda Toolkit, cuDNN and GPU. However, Google Colab has all of this, just need to configure and enable these options. After these steps, an execution command is activated that updates

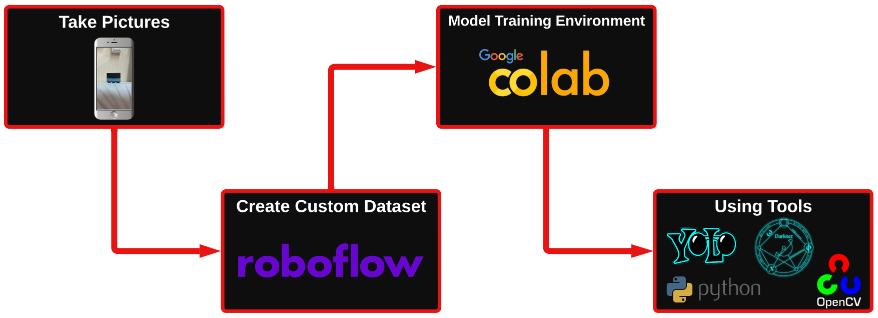


Figure 4.11: Tools union used for training.

the compilation. Then, the training weights of the 29 layers of the YOLOv4 Tiny model, pre-trained in the coco dataset, are downloaded to be used as the training starting point.

During this phase, specific configurations for box detection are initiated. The YOLO Darknet compressed file, which consists of JPG and TXT files, is loaded and placed in the drive folder, separated into training and test directories. This folder is subsequently accessed by the script, where it is unzipped. Additionally, three more files are uploaded to the same folder, which will be accessed by the script to establish the training settings. They are the object names, object data files and training model settings, with their settings being displayed:

1. **obj.names:**
    - BOX.
2. **obj.data:**
    - classes = 1;
    - train = path to folders of train.txt files;
    - valid = path to folders of test.txt files;
    - names = path to folders of obj.names files; and
    - bachup = path to trained weights storage folders.

3. **yolov4-custom.cfg:**

    - batch = 64 (images for iteration);
    - subdivisions = 16 (batch iteration images subdivision);
    - max_batches = 6000 (num_classes*2000 but never less than 6000);
    - width = 416 (multiple of 32. Larger size improves accuracy but increases training time);
    - height = 416 (has to be multiple of 32);
    - steps = 8000, 9000 (80% of max_batches), (90% of max_batches);
    - classes = 1; and
    - filters = 18 ((num_classes + 5) * 3).

Finally, with all the settings made, the network is trained. For this, the following files were passed: obj.data, yolov4-custom.cfg and yolov4-tiny.conv.29. Thus, the network begins its training until it completes 6000 iterations, and at every 1000 iterations, the training personnel files achieved are stored in the backup folder. At the end of the training its final value is saved as yolov4-tiny-custom_final.weights, its performance is checked and the weights are tested in the object detection algorithm.

.

Chapter 5

Tests and Results

5.1 Detection of Boxes

The training dataset was built using the Roboflow platform.The accuracy of the data used to train object detection models could be observed through the platform, like chapter 4. However, the tool is limited, and the detector can be used only or optimized for the NVIDIA Jetson line, both of which are not within the scope of this work. For verification purposes, show in Figure 5.2 using the training data, an mAP of 99% and a precision of 100% were obtained.

The training of the custom box detectorin the localization module was based on the YOLOv4-tiny and Darknet method. His training took place on Google colab, using the dataset produced in Roboflow. His training took approximately one hour, 14 minutes and 34 seconds. The training mAP is shown in figure 5.3. The performance of this network is presented:

- 38 layers of weight files;
- ap = 100 % ;
- mAP@0.50 = 1.00;
- confidence threshold = 0,25:

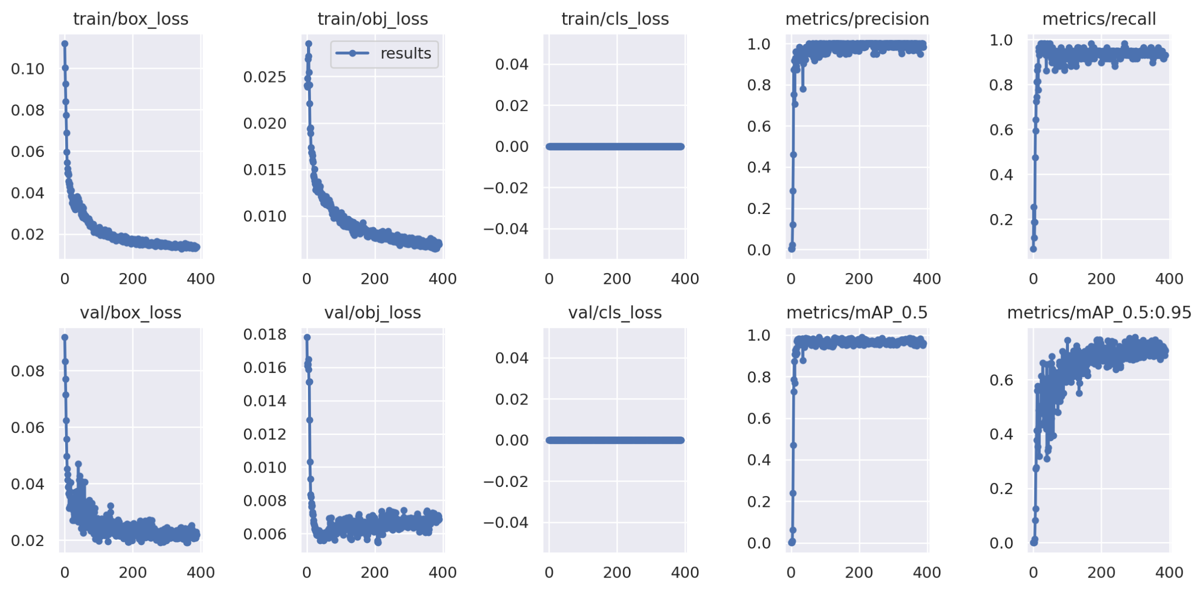


Figure 5.1

Figure 5.2: Results of training in Roboflow Platform.

- Precision = 1 (correct positive identifications);
- Recall = 1 (predict positives);
- F1 - score = 1 (Balance between accuracy and recall); and
- TP = 23 / FP = 0 / FN = 0.

- Points datasets:
  - MS COCO = 101;
  - PascalVOC 2007 = 11; and
  - ImageNet and PascalVOC 2010-2012 = 0.

5.2 Initial Applicability Test

From some initial tests performed for generic training of the detection model, a good detection of the objects was noticed. However, the selection box formed by the detection

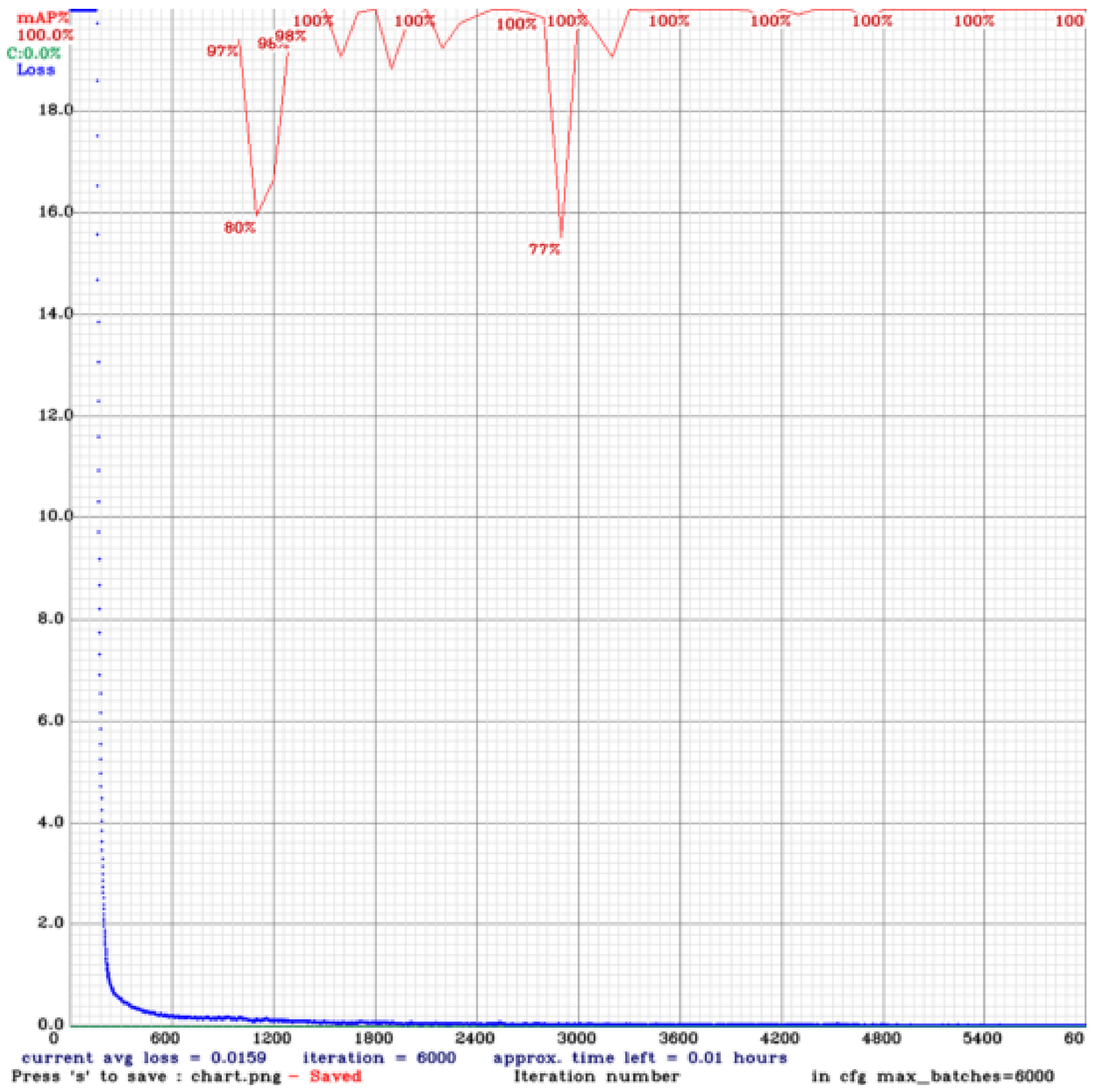


Figure 5.3: Results mAP of training method YOLOv4-Tiny on Google Colab.

code to frame the object was very distant from the object's contours. It causes an error in calculating the localization parameters since the value of the width of the bounding box is used. Thus, through the results in Figure 5.4, the error was parameterized. The difference between the actual contour and the one produced by the detection algorithm is calculated by

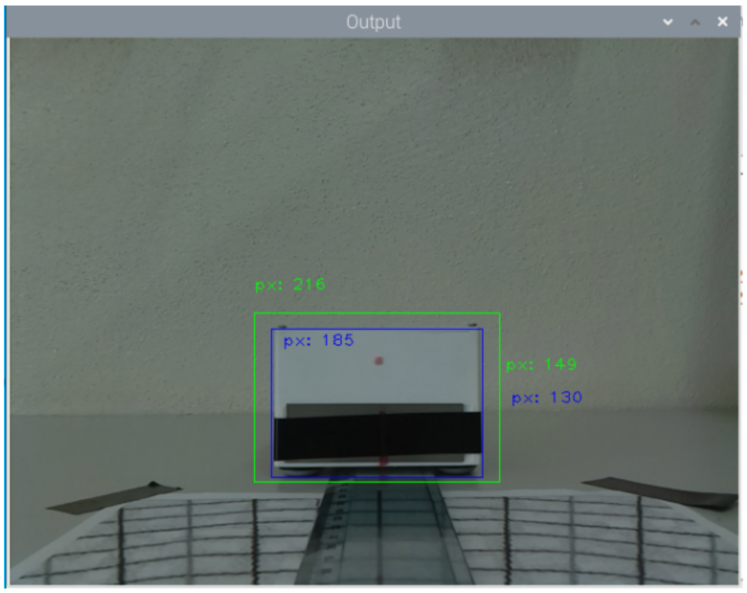


Figure 5.4: Comparison of actual bounding box and detection model

$$Multiplication\_factor = \frac{Object}{BBox} = \frac{185}{216} = 0,8564 \cong 0,85, \tag{5.1}$$

which is the factor applied to the bounding box. The algorithm keeps the same when detecting the object but reduces by approximately 15% of its original size when plotting. Thus, even if the detection model does not fit the object in the detection box in the best way, but it has a good detection capability, correction factors can be applied. In this way, it was verified that errors could be corrected if found, and the experiment could proceed.

This type of error occurs because of the way objects are annotated in the training dataset. If they are offset or turned differently, they are still identified, but not properly. For this reason, the pre-training phase of object annotation is crucial, and it must be presented as accurately as possible how the objects should be located, with respect to their shapes and proportions. In order to proceed with the tests, the detection model had its dataset well structured as presented in Section 4.3.1 and 5.1. The objective was to detect only the largest side face of the box, used as a parameter for algorithm development and calculations.

5.3 Parameterized Lengths

The main tests started parameterizing the measures, the real world and the digital image. For this purpose, the test box was placed at a known distance of 20 cm, as shown in figure 5.6a. Its actual dimensions are presented in the figure 5.5. The Teflon feet placed at the bottom of the box are approximately 0.5 cm, totaling an effective height of 6.5 cm.

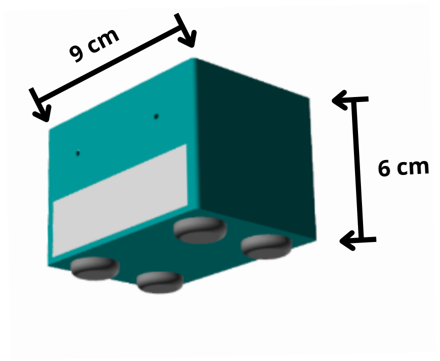
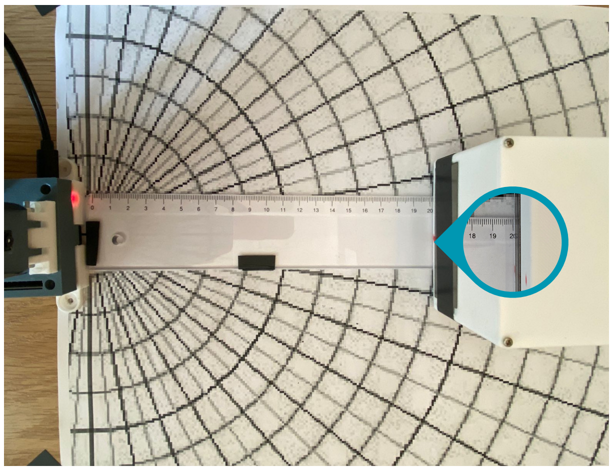


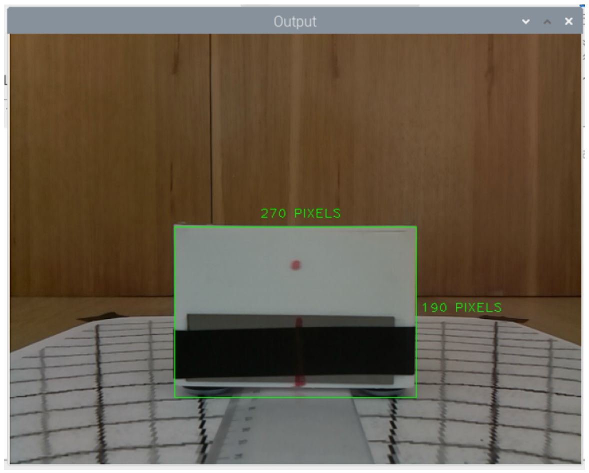
Figure 5.5: Box dimensions of the robot factory lite test, adapted from [11]

By utilizing the RaspiCam camera, Python, and OpenCV, a series of selection boxes with predetermined dimensions were drawn until the optimal fit for the tracked object was achieved, obtaining the results shown in Figure 5.6. The best values being 270 × 190 pixels. The optimal size for the box is 270 × 190 pixels, and the image it is contained within measures 640 × 480 pixels in total.

With the parameterized data, it is possible to pass on to tests with applications of the algorithms to obtain the detections of the objects and calculate the distances.



(a)



(b)

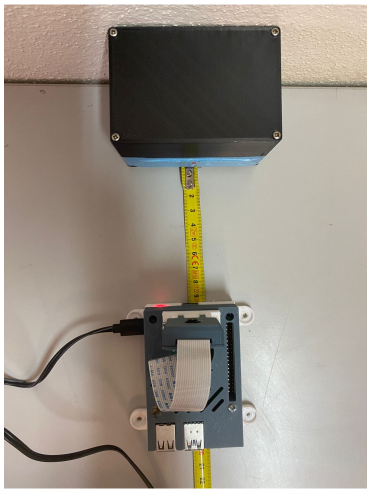
Figure 5.6: Parameterized Lengths. (a) Top image of parameterized e (b) Digital image parameterized

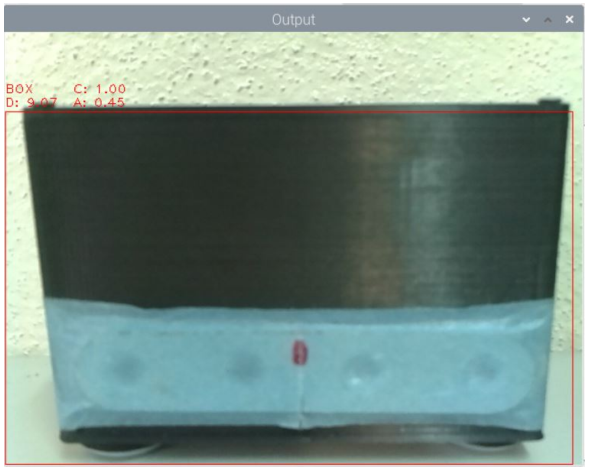
5.4 Position variation tests

5.4.1 Operating Distance Measurement

The operating range, referring to the detection distances, of the module was obtained through practical tests. Initially, the box and the detection module were positioned facing each other, in such a way that the camera was practically touching the box, with their centers aligned. Then the detection module was moved away centimeter by centimeter from the box, respecting the central alignment, until the box was detected by the module. This detection refers to the minimum distance necessary for the object to be detected. The next step was to move the module further away until the object was no longer detected, this being the maximum distance that the module can detect the objects. For each case, 10 reference measurements were noted.

The test, for measuring the maximum detection distance, can be seen in the image 5.8. The figure was taken in order to visually identify where the module and the box were, as well as the space between them. The figure 5.8b shows the image obtained by the algorithm embedded in the detection module. The visualization of the minimum operating distance can be seen in the figure 5.7, and the real images and the images obtained by the module, can be seen respectively in the figures 5.7a and 5.7b. The results that appear in the image, besides the rectangular selection box, are the object's class, confidence, distance and angle. The results obtained for the system's operating range distances can be seen in the table 5.1, and the analysis of the values found in the table 5.2.


(a)



(b)

Figure 5.7: Minimum operating distance measurement. (a) Image actual of minimum distance (b) Image of module with calculated minimum distance.

Experiment	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Maximum	248,45	259,74	248,45	248,45	248,45	248,45	248,45	259,74	248,45	248,45
Minimum	8,93	9,10	9,08	8,93	8,96	9,07	9,07	8,93	8,93	8,98

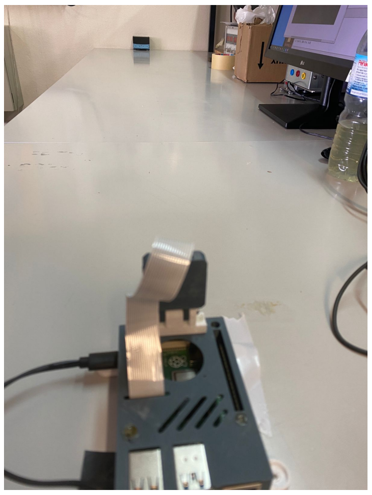
Table 5.1: Obtained operating distance measurements in centimeters.

Distance	**Actual**	**Average**	**Absolute Error**
Maximum	218,5	250,7	32,2
Minimum	8,99	9,4	0,4

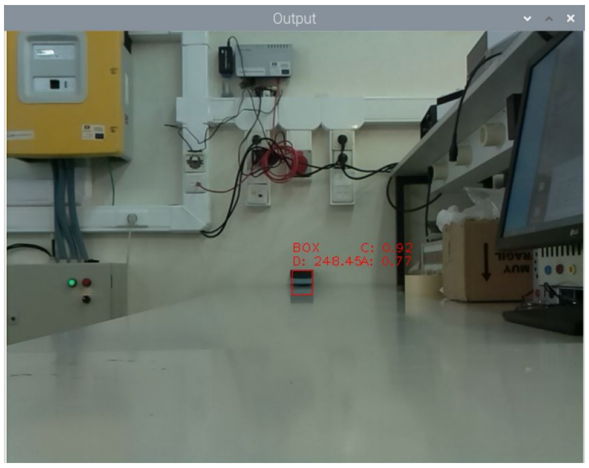
Table 5.2: Distance Analysis Calculation.

5.4.2 Identification in different angle and distances

A second experiment was done to verify the operation of the detection module for locating the box at different distances and angles. For this it was fixed on a polar graph sheet,



(a)



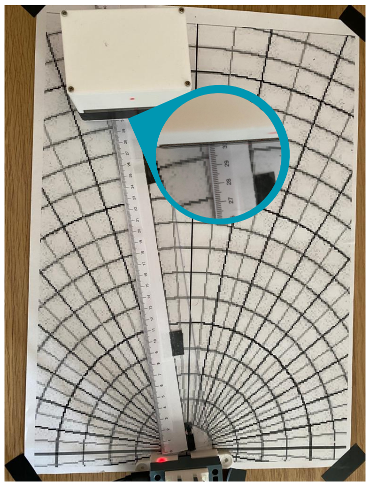
(b)

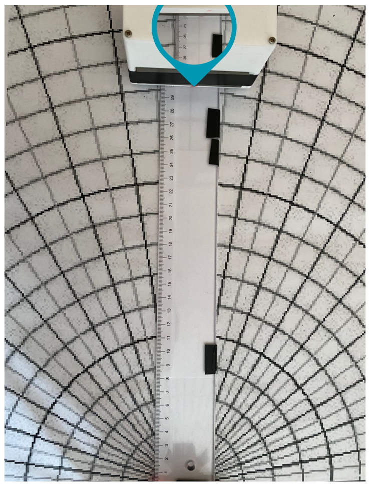
Figure 5.8: Maximum operating distance measurement. (a) Image actual of maximum distance (b) Image of module with calculated maximum distance.

with the center of the camera aligned with the center of the graph. The experiment was done respecting the hFoV of the camera [71], of 53.5 degrees. The distances chosen for the measurements were from 20 to 60 centimeters, with a variation of 10 centimeters between each distance. The angles were 0 to 20 degrees, varying in 10 degrees for both sides, right and left, with respect to the central reference axis. For convenience we adopted the visualization only of the angle value in modulo, since the direction of rotation is previously verified, as presented in Section 4.2.2.

The measurements were made with the camera always stationary, varying only the position of the box. Ten measurements were made for each position, always in such a way that the camera could see only the target face of the experiment. Starting always from the center, without varying the angle with respect to our references, as shown in figure 5.9b, obtaining the results of the module, as shown in figure 5.10b. Then, for box shifted with respect to the center of the system, on the left and right side, as presented respectively in figures 5.9a and 5.9c. Thus, obtaining results as presented in figures 5.10a 5.10c.

The calculated results, as a reference of the various measurements are presented in the figures 5.11 and 5.12, respectively referring to the distance and angle values. For each


(a)


(b)

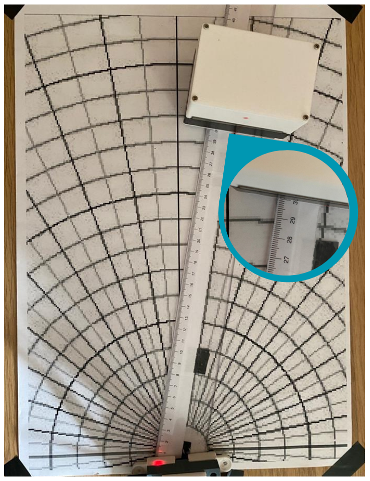
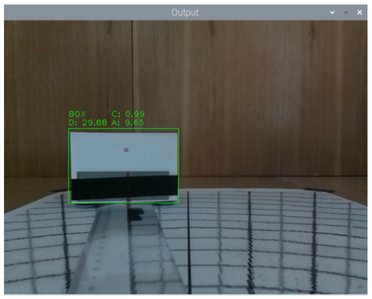
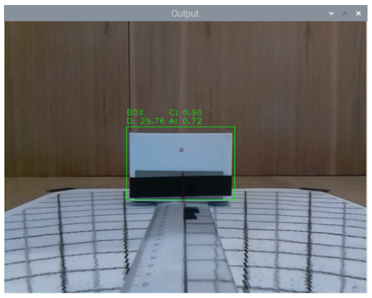

(c)

Figure 5.9: Measurement of actual parameters with the box to the camera. (a) 30 cm at 10° at left, (b) 30 cm at 0°, (c) 30 cm at 10° at right.



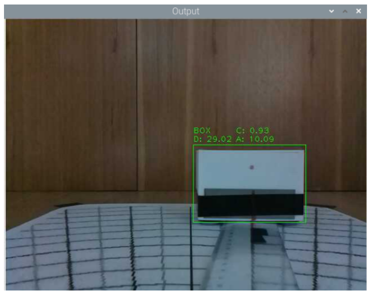
```
BOX      C: 0.99
D: 29.88 A: 9.65
```

(a)



```
BOX      C: 0.90
D: 29.76 A: 0.72
```

(b)



```
BOX      C: 0.93
D: 29.02 A: 10.09
```

(c)

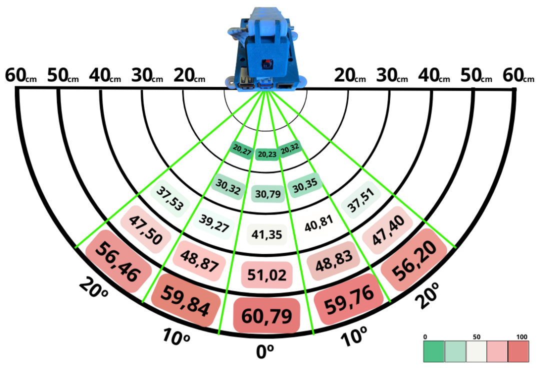
Figure 5.10: Result obtained by the detection and localization module. (a) 30 cm at 10° at left, (b) 30 cm at 0°, (c) 30 cm at 10° at right.

position, as mentioned earlier, ten measurements were performed, where the box object was positioned and the first ten readings of the algorithm were noted. In order to remove the possibility of introducing human error, the box was not moved while the readings were being taken, changing position only when the readings were finished.

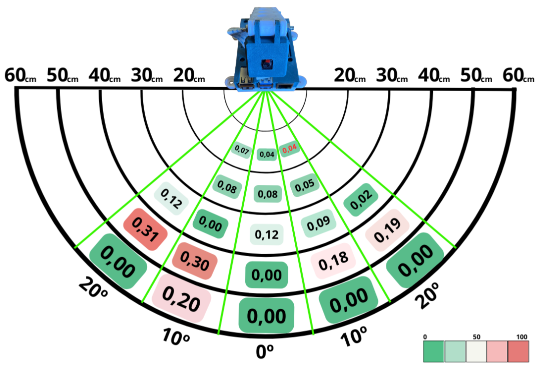
The values obtained and displayed by the module are the class name of the detected object, accuracy, the distance and the displacement angle. As we sought to find a class, when an object was detected, reference to this class was obligatorily made. With regard to accuracy, we obtained a unanimous measurement of 100% in all cases, using rounding and two decimal places. For the distance and angle metrics, four reference values were obtained for analysis, being the mean, standard deviation, absolute error, and relative error, referring to the measurements. These results are presented in the figures 5.11a, 5.11b, 5.11c and 5.11d, referring to distance, and in the figures 5.12a, 5.12b, 5.12c and 5.12d, referring to the angle.

Distance - analysis of the results obtained

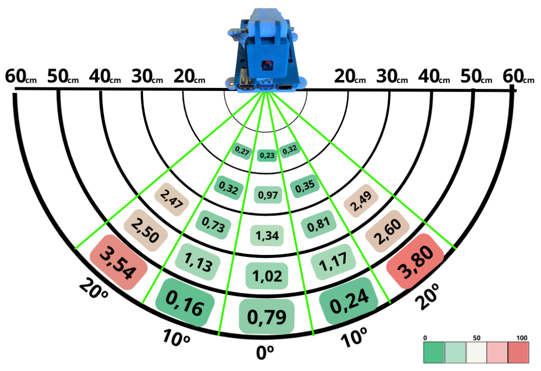
The analysis of the results presented the variations referring to the change position of the object. Starting with the analysis of the standard deviation, we tried our best to reduce the introduction of errors that could lead to obtaining different values in the samples, and seeking a common value. But, these differences still occurred, as can be seen in Figure 5.11b, where there is growth concerning the increase of distance, mostly up to 50 centimeters. From 60 centimeters on they began to decrease, and except the position at 20° to the right, all the values came to zero. The most significant differences occurred at 50 centimeters, at 10º and 20º on the right side. With respect to the value of the average of the samples, presented in Figure 5.11a, the absolute error is, presented in Figure 5.11c. Following a pattern almost similar to the standard deviation, the error increases with respect to distance up to 50 centimeters. After 50 centimeters, the error starts to decrease, at the values closest to the central axis. But at the extremities, the value increases and it can also be seen that the error is greater on the left side. The figure 5.11d, presents the percentage referring to the absolute error and the value of the



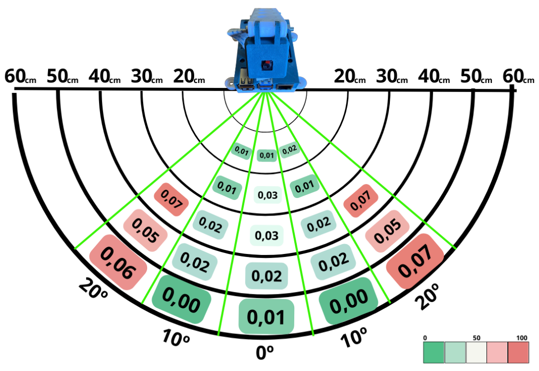
(a)



(b)

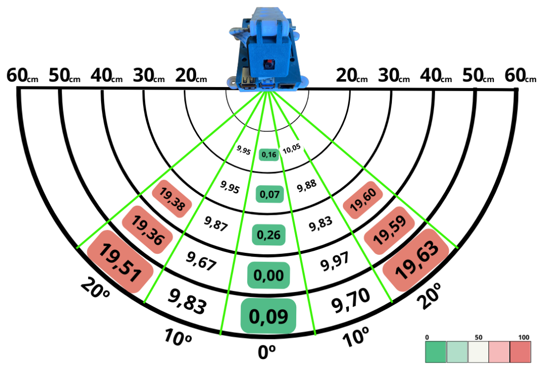


(c)

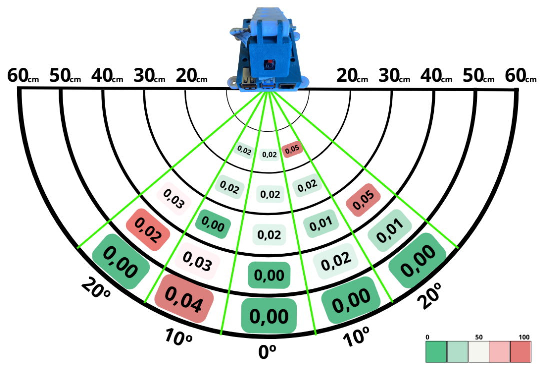


(d)

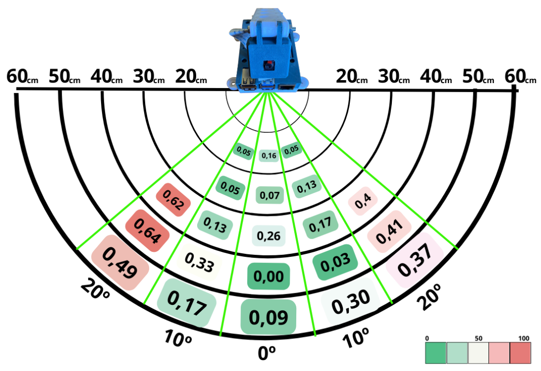
Figure 5.11: Relative distance measurements, based on 10 samples for each position of interest. Relative angle measurements, based on 10 samples for each position of interest. (a) Mean (cm), (b) Standard Deviation (cm), (c) Absolute Error (cm) and (d) Relative Error.



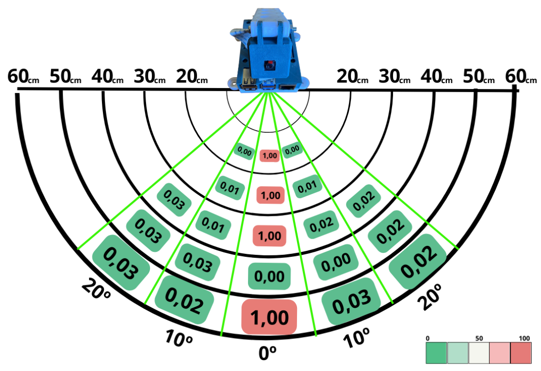
(a)



(b)



(c)



(d)

Figure 5.12: Relative angle measurements, based on 10 samples for each position of interest. (a) Mean (graus), (b) Standard Deviation (graus), (c) Absolute Error (graus) and (d) Relative Error.

mean, which is similar on both sides, with a slight difference only for the distance of 60 centimeters at 20^{o}, on the right side.

The differences in measurements that occurred, between items located at the same distance and displacement, but on opposite sides, did not present equal values due to the irregularities and physical errors introduced. Some of these errors are the difference in lighting, imperfections in the module support and measurement environment, possible imperfections in the camera, among others. However, the values were as expected, because the farther away it is, the more difficult it is to visualize it, consequently, not being able to assertively locate its contour, used as a parameter. Thus, leading to errors with respect to the estimated and actual value.

Angle - analysis of the results obtained

The parameters, referring to the angles, presented a higher standard deviation on the right side, figure 5.12b but without a pattern of distribution of the values. With the average values, presented in the figure, the absolute error can be calculated, figure Absolute Angle. The absolute error presents an increase concerning the angular variation, being small in the center and larger at the extremities in the vast majority of values, what cannot be said about the distance variation where the error value of the angle does not present a pattern. For the relative error, figure 5.12d, shows that its largest values are in the center, being smaller and varied around it.

As in the case of distance, different values are found in all cases. Some of these errors are the difference in illumination, imperfections in the module support and measurement environment, possible imperfections in the camera, among others. But it can be seen what was expected, that as the object gets further away, the object gets smaller and thus displacement movements have less impact compared to distances close to the camera.

Chapter 6

Conclusion and Future Works

This work had as its main objective the development of a system of detection and localization of objects in real-time, aiming to incorporate it in a physical module to test its application. Artificial Intelligence concepts were used with computer vision to acquire information from the real world to create the system. Object detection models based on neural networks were utilized for machine learning. The image data was then analyzed using geometric modeling techniques to identify specific image features.

The choice of tools and methods, was based on the state of the art of machine vision and object detection. Several methods were analyzed for detection, taking into account accuracy and speed. For localization, the analysis was based on the components, such as cameras, and the geometric modeling of distance estimation. A union of methods and tools was sought to achieve the main goal. Thus, the YOLO-based detection method, Specifically YOLOv4 tiny version, was chosen for its accuracy and speed, and its optimization for working in restricted processing environments. Also, relying on geometric modeling based on the Pinhole model, use the information returned by the detection to parameterize a distance relationship between the image and the real world.

The development was divided into three fundamental parts: the choice of components to assemble the module, the development of the object detection/localization algorithm, and the creation and training of the custom dataset.

The choice of module components was based on cost, finding inexpensive components, and the ability to perform the project requirements. The Raspberry Pi Camera Rev 1.3 module was chosen for image acquisition, with a maximum resolution of 1080 pixels, mainly because of its great compatibility with the microcomputer, Raspberry Pi 4 Model B, used for processing the application. This microcomputer can run object detection applications and is compatible with the OpenCV library, providing tools for working with artificial vision.

The algorithm was programmed in Raspbian's Thonny interpreter and divided into two fundamental parts: object detection and distance estimation. The first part uses functions from the OpenCV library and network configuration files, created outside the program, to create a function to detect specific objects. After detecting the objects, the localization functions (distance and angle) were created, based on the detection and trigonometry feedback data. To estimate the location, object detection and distance estimation techniques were combined through the use of object framing and trigonometry parameters, along with the Pinhole method.

The files used in the detection function, were previously created for training the custom network. The Roboflow platform was used to create a custom dataset, which included annotated files, method configuration values, and class names stored in separate files. With this data, the training was done in Google Colab, in order to obtain the weights for programming the network in the algorithm.

After completing all the specified development, the system proceeded to the testing phase to measure its effectiveness. The training data and the framing of the detection box were analyzed for detection. For this, two tests were performed, one in relation to the detection model and the other in relation to the localization. The training data and the framing of the detection box were analyzed for detection. To analyze the accuracy of the distance estimate, first the module was positioned in front of the object, The objective is to detect a box and measure the distance range of the module by obtaining the minimum and maximum values. Then, the object was fixed in a position and the object had its location varied, with respect to distance and angle, to obtain its accuracy in most of its area of operation. Thus, varying the angle within hFoV and the distance up to the value of 60 cm.

To conclude, the results obtained with respect to object detection were very satisfactory, obtaining a mean average precision (mAP) of 100% and the parameters of the detection box. Which frame the object, obtained results with accuracy above 95%, varying with respect to the distance from the object and camera, parameterized according to the expected and obtained detection box. For distance, the operating range is from 8.99 cm to 218.5 cm, measured in the central alignment. With angle variation, larger errors were obtained, to some extent, with increasing distance and angle. The largest absolute error obtained was 3.8 cm, located at 60 cm with 20° to the left. The smallest error was obtained at 10º to the right at 60 cm, with a value of 0.17 cm. Regarding the angle, the absolute error also increases to some extent with respect to distance, but in this case, the largest errors are at the largest angles and shortest distances. The largest error was obtained at 50 cm at 20° to the right, with an error value of 0.64°. The smallest was 0.05º, at 20 cm on both sides at 10º.

It is verified that the errors are not standardized with the position variation analyzing both values related to the absolute error due to the non-ideality of the equipment used, lighting variation, module temperature, among others. No matter how hard one tries to keep the same parameters for all measurements, this is not possible due to the problems cited. However, the error variation increasing up to a certain distance and decreasing from a certain point on was expected. Because the lens imperfection is introduced, no matter how much the camera module software tries to correct this deformation, this does not happen in its entirety. And the quality and blur of the image obtained by the camera, gets worse as an object moves away. Thus, the image loses information and will return to a higher error at huge distances, close to the maximum detected value. Bounding box will have errors, as presented in the parameterization of the longest distance value at which the object is still detected.

6.1 Future Works

This work addresses several applications for the detection and localization of objects. With this, some implementations for future work are presented:

- Train and attempt to deploy other object detection models, to improve performance and accuracy in object detection. Once the model has been trained for the new objects of interest, it is possible to test the model on the localization module, provided that the module can receive the outputs of the detection model for the new classes;

- Test the module for application in environments with uneven terrain, using the vertical field of view of the camera to estimate both the distance between the module and the object and the vertical displacement angle of the object's center relative to the camera's central axis. This approach allows for more precise information on the object's location in environments with irregular terrain;

- Installing heatsinks and fans on the Raspberry Pi to keep the temperature stable, aiming for long periods of operation, given that increased temperature can affect the module's performance;

- Using a Deep Learning accelerator is an effective strategy to maximize the performance of the detection algorithm, allowing for achieving the maximum possible FPS and, thus, increasing the detection speed. With this approach, it is possible to process a large volume of data on time, ensuring more accurate and reliable results; and

- Embarking the module on a robot allows it to use the collected information to autonomously locate objects and perform tasks, making the most of the system's capabilities in applications that require mobility and interaction with the environment. The integration of the system with the robot would open up new possibilities for application, allowing the use in various areas such as robotics, industrial automation, and logistics, among others.

Bibliography

[1] J. L. Crowley, H. I. Christensen, and A. Chehikian, *Vision as process: basic research on computer vision systems.* Springer, 1995.

[2] V. Wiley and T. Lucas, “Computer vision and image processing: A paper review,” *International Journal of Artificial Intelligence Research*, vol. 2, no. 1, pp. 29–36, 2018.

[3] F. F. Feliciano, I. L. de Souza, and F. R. Leta, “Visão computacional aplicacada à metrologia dimensional automatizada: Considerações sobre sua exatidão,” *Engevista*, 2005.

[4] “Uma perspectiva sobre técnicas de localização de alcance para visão computacional,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, DOI: 10.1109/TPAMI.1983.4767365.

[5] K. P. Valavanis and G. N. Saridis, *Intelligent robotic systems: theory, design and applications.* Springer Science & Business Media, 2012, vol. 182.

[6] M. Bertozzi and A. Broggi, “Gold: A parallel real-time stereo vision system for generic obstacle and lane detection,” *IEEE transactions on image processing*, vol. 7, no. 1, pp. 62–81, 1998.

[7] J. França, “Desenvolvimento de algoritmos de visão estereoscópica para aplicações em robótica móvel,” *UFSC, Florianópolis-SC, Brasil*, 2003.

[8] R. Vassallo, A. Franca, and H. Schneebeli, "Detecção de obstáculos através de um fluxo óptico padrão obtido a partir de imagens omnidirecionais," in *Proc. of Latin America IEEE Robotics Symp.(SBAI/IEEE-LARS)*, 2005.

[9] L. Brancalião, J. Gonçalves, M. Á. Conde, and P. Costa, "Systematic mapping literature review of mobile robotics competitions," *Sensors*, vol. 22, no. 6, p. 2160, 2022.

[10] L. B. Almeida, J. Azevedo, C. Cardeira, *et al.*, "Mobile robot competitions: Fostering advances in research, development and education in robotics," 2000.

[11] J. Lima, P. Costa, And V. Pinto, *Robot@factory lite*, Disponível em: `https://web.fe.up.pt/~robotica2019/index.php/en/robot-factory-lite-2`. Acesso em: 06 de janeiro 2023, 2019.

[12] P. Costa, J. Lima, And V. Pinto, *Robot@factory 4.0*, Disponível em: `https://www.festivalnacionalrobotica.pt/2022/competicoes/robotfactory-4.0-11`. Acesso em: 06 de janeiro 2023, 2022.

[13] L. E. Luiz, L. Pilarski, K. Baidi, *et al.*, "Robot at factory lite - a step-by-step educational approach to the robot assembly," in *ROBOT2022: Fifth Iberian Robotics Conference*, D. Tardioli, V. Matellán, G. Heredia, M. F. Silva, and L. Marques, Eds., Cham: Springer International Publishing, 2023, pp. 550–561, ISBN: 978-3-031-21065-5.

[14] Y. Amit, P. Felzenszwalb, and R. Girshick, "Object detection," *Computer Vision: A Reference Guide*, pp. 1–9, 2020.

[15] X. Zhang, Y.-H. Yang, Z. Han, H. Wang, and C. Gao, "Object class detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, pp. 1–53, 2013.

[16] P. P. Sampedro and H. A. G. de Souza, "Análise de sistemas de produção de imagens estereoscópicas baseados em câmeras com uma única objetiva," *Linguagens-Revista de Letras, Artes e Comunicação*, vol. 8, no. 2, pp. 185–205, 2014.

[17] G. R. Esteves, M. FEITOSA, and B. J. T. Fernandes, "Estereoscopia no cálculo de distância e controle de plataforma robótica," in *SIBGRAP-Conference on Graphics, Patterns and Images*, 2011, pp. 65–70.

[18] R. C. Gonzalez and R. E. Woods, *Processamento de imagens digitais*. Editora Blucher, 2000.

[19] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," *Advances in neural information processing systems*, vol. 1, 1988.

[20] J. A. Da Silva, J. A. Aznar-Casanova, N. Pinto-Ribeiro Filho, and J. E. Santillán, "On the metric of visual space," *Arquivos brasileiros de oftalmologia*, vol. 69, no. 1, pp. 127–135, 2006.

[21] F. Ribeiro, G. Lopes, J. Costa, N. Pereira, J. Cruz And E. Bicho, *Robot@factory*, Disponível em: `http://robotica2012.dei.uminho.pt/12/index.php-option=com_content&view=category&layout=blog&id=75&Itemid=99.html`. Acesso em: 06 de janeiro 2023, 2012.

[22] P. J. Costa, N. Moreira, D. Campos, J. Gonçalves, J. Lima, and P. L. Costa, "Localization and navigation of an omnidirectional mobile robot: The robot@ factory case study," *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, vol. 11, no. 1, pp. 1–9, 2016.

[23] J. Lima, V. Oliveira, T. Brito, *et al.*, "An industry 4.0 approach for the robot@ factory lite competition," in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, IEEE, 2020, pp. 239–244.

[24] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[25] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *International conference on artificial neural networks*, Springer, 2011, pp. 44–51.

[26] G. W. Taylor, I. Spiro, C. Bregler, and R. Fergus, "Learning invariance through imitation," in *CVPR 2011*, IEEE, 2011, pp. 2729–2736.

[27] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," English (US), Publisher Copyright: © 2014 International Conference on Learning Representations, ICLR. All rights reserved.; 2nd International Conference on Learning Representations, ICLR 2014 ; Conference date: 14-04-2014 Through 16-04-2014, 2014.

[28] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587. DOI: `10.1109/CVPR.2014.81`.

[29] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, pp. 154–171, 2013.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[31] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448. DOI: `10.1109/ICCV.2015.169`.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015. DOI: `10.1109/TPAMI.2015.2389824`.

[33] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, 2006, pp. 2169–2178. DOI: `10.1109/CVPR.2006.68`.

[34] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. [Online]. Available: `https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf`.

[35] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017. DOI: `10.1109/TPAMI.2016.2577031`.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. DOI: `10.1109/CVPR.2016.91`.

[37] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[38] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. arXiv: `1612.08242`. [Online]. Available: `http://arxiv.org/abs/1612.08242`.

[39] A. Farhadi and J. Redmon, "Yolov3: An incremental improvement," in *Computer vision and pattern recognition*, Springer Berlin/Heidelberg, Germany, vol. 1804, 2018, pp. 1–6.

[40] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. arXiv: `2004.10934`. [Online]. Available: `https://arxiv.org/abs/2004.10934`.

[41] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[42] J. Deng, X. Xuan, W. Wang, Z. Li, H. Yao, and Z. Wang, "A review of research on object detection based on deep learning," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1684, 2020, p. 012 028.

[43] S. K. Baduge, S. Thilakarathna, J. S. Perera, *et al.*, "Artificial intelligence and smart vision for building and construction 4.0: Machine and deep learning methods and applications," *Automation in Construction*, vol. 141, p. 104 440, 2022.

[44] R. Boutaba, M. A. Salahuddin, N. Limam, *et al.*, "Uma pesquisa abrangente sobre aprendizado de máquina para redes: Evolução, aplicações e oportunidades de pesquisa," *Jornal de Serviços e Aplicações da Internet*, vol. 9, ed. by Springer,

[45] A. Paullada, I. D. Raji, E. M. Bender, E. Denton, and A. Hanna, "Data and its (dis) contents: A survey of dataset development and use in machine learning research," *Patterns*, vol. 2, no. 11, p. 100 336, 2021.

[46] F. D'Antoni, F. Russo, L. Ambrosio, *et al.*, "Artificial intelligence and computer vision in low back pain: A systematic review," *International Journal of Environmental Research and Public Health*, vol. 18, no. 20, p. 10 909, 2021.

[47] A. Alblushi, "Face recognition based on artificial neural network: A review," *Artificial Intelligence & Robotics Development Journal*, pp. 116–131, 2021.

[48] "Aprendizado profundo para detecção de objetos e percepção de cena em carros autônomos: Pesquisa, desafios e problemas em aberto," vol. 10, ed. by Elsevier,

[49] G. da Silva Vieira, R. T. Parreira, F. A. A. Soares, G. T. Laureano, and R. M. Costa, "Depth map production: Approaches, challenges and applications,"

[50] Y. Lu, "Artificial intelligence: A survey on evolution, models, applications and future trends," *Journal of Management Analytics*, vol. 6, no. 1, pp. 1–29, 2019.

[51] L. Steffen, D. Reichard, J. Weinland, J. Kaiser, A. Roennau, and R. Dillmann, "Neuromorphic stereo vision: A survey of bio-inspired sensors and algorithms," *Frontiers in neurorobotics*, vol. 13, p. 28, 2019.

[52] K. Y. Kok and P. Rajendran, "A review on stereo vision algorithm: Challenges and solutions," *ECTI Transactions on Computer and Information Technology (ECTI-CIT)*, vol. 13, no. 2, pp. 112–128, 2019.

[53] L. E. de Carvalho, A. C. Sobieranski, and A. von Wangenheim, "Revisão da literatura para reconhecimento/classificação de objetos 3d," 2017.

[54] G. Magalhães, L. Colombini, R. Técnico-IC-PFG, and P. F. de Graduação, "Detecção de objetos no futebol de robôs," *Campinas-SP, Brazil*, 2017.

[55] N. Werneck, F. Truzzi, and A. Costa, "Medição de distância e altura de bordas horizontais com visão monocular linear para robôs móveis," Jan. 2009.

[56] G. Magalhães, L. Colombini, R. Técnico-IC-PFG, and P. F. de Graduação, "Detecção de objetos no futebol de robôs," *Campinas-SP, Brazil*, 2017.

[57] J. D. Domingues, "Localização de robôs móveis por meio de fluxo ótico e fusão sensorial em ambientes de mineração.," 2022.

[58] A. Dennis, *Raspberry Pi Computer Architecture Essentials*, 1st. UK: Packt Publishing Ltd, 2016, ISBN: 1784397970.

[59] D. Pena, A. Forembski, X. Xu, and D. Moloney, "Benchmarking of cnns for low-cost, low-power robotics applications," in *RSS 2017 Workshop: New Frontier for Deep Learning in Robotics*, 2017, pp. 1–5.

[60] Z. Zhao, Z. Jiang, N. Ling, X. Shuai, and G. Xing, "Ecrt: An edge computing system for real-time image-based object tracking," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018, pp. 394–395.

[61] K. Manjari, M. Verma, and G. Singal, "Creation: Computational constrained travel aid for object detection in outdoor environment," in *2019 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, IEEE, 2019, pp. 247–254.

[62] V. Gonzalez-Huitron, J. A. León-Borges, A. Rodriguez-Mata, L. E. Amabilis-Sosa, B. Ramírez-Pereda, and H. Rodriguez, "Disease detection in tomato leaves via cnn with lightweight architectures implemented in raspberry pi 4," *Computers and Electronics in Agriculture*, vol. 181, p. 105 951, 2021.

[63] G. Mu, Z. Xinyu, L. Deyi, Z. Tian-lei, and A. Lifeng, "Traffic light detection and recognition for autonomous vehicles," *The Journal of China Universities of Posts and Telecommunications*, vol. 22, pp. 50–56, 2015.

[64] P. Pydipogu, M. Fahim, and M. Shafique, *Robust lane detection and object tracking in relation to the intelligence transport system*, 2013.

[65] K. Horak and L. Zalud, "Image processing on raspberry pi for mobile robotics," 2016.

[66] M. Malathi and R. Geetha, "A real time image processing based fire safety intensive automatic assistance system using raspberry pi," *Int. J. Mod. Trends Sci. Technol*, vol. 2, pp. 34–38, 2016.

[67] Raspberry Pi, *Raspberry pi 4*, Disponível em: `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/`. Acesso em: 12 de janeiro 2023, 2023.

[68] Raspberry Pi, *Raspberry pi 4 tech specs*, Disponível em: `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/`. Acesso em: 12 de janeiro 2023, 2023.

[69] FILIPEFLOP, *Keyboard and mouse raspberry pi: New at filipeflop*, Disponível em: `https://www.filipeflop.com/blog/teclado-e-mouse-raspberry-pi/`. Acesso em: 12 de janeiro 2023, 2023.

[70] Adafruit, *Raspberry pi camera board*, Disponível em: `https://www.adafruit.com/product/1367`. Acesso em: 12 de janeiro 2023.

[71] Raspberry Pi Foundation, *Raspberry pi documentation - camera modules*, Disponível em: `https://www.raspberrypi.com/documentation/accessories/camera.html`. Acesso em: 12 de janeiro 2023, 2023.

[72] A. Annamaa, "Introducing thonny, a python ide for learning programming," in *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, ser. Koli Calling '15, Koli, Finland: Association for Computing Machinery, 2015, pp. 117–121, ISBN: 9781450340205. DOI: `10.1145/2828959.2828969`. [Online]. Available: `https://doi.org/10.1145/2828959.2828969`.

[73] Microsoft, *Code editing.redefined.* Disponível em: `https://code.visualstudio.com/`. Acesso em: 04 de fevereiro 2023, 2023.

[74] The Apache Software Foundation, *Apache netbeans*, Disponível em: `https://netbeans.apache.org/download/index.html`. Acesso em: 04 de fevereiro 2023, 2022.

[75] Simon Long (Raspberry Pi blog), *A raspbian desktop update with some new programming tools*, Disponível em: `https://www.raspberrypi.com/news/a-raspbian-desktop-update-with-some-new-programming-tools/`. Acesso em: 04 de fevereiro 2023, 2017.

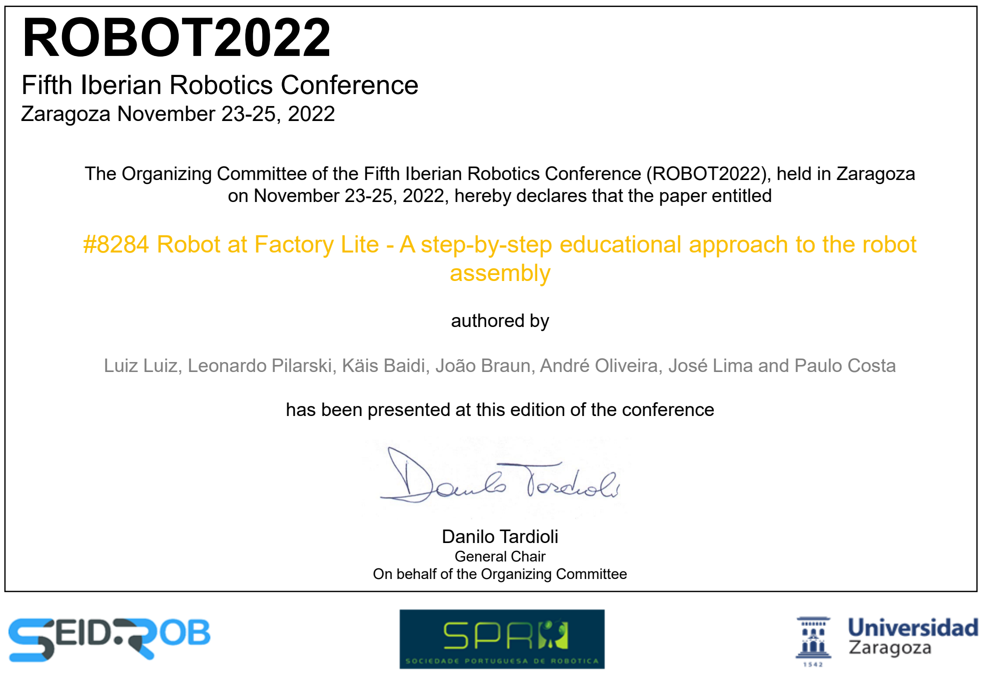
[76] K. R. Chowdhary, "On the evolution of programming languages," *CoRR*, vol. abs/2007.02699, 2020. arXiv: `2007.02699`. [Online]. Available: `https://arxiv.org/abs/2007.02699`.

[77] Raspberry Pi Foundation, *Getting started with the camera module*, Disponível em: `https://projects.raspberrypi.org/en/projects/getting-started-with-picamera/2`. Acesso em: 12 de janeiro 2023, 2023.

[78] Jacob Solawetz, Joseph Nelson, Samrat Sahoo, *How to train yolov4 on a custom dataset*, Disponível em: `https://blog.roboflow.com/training-yolov4-on-a-custom-dataset/`. Acesso em: 13 de fevereiro 2023, 2020.

Appendix A

Article Produced



ROBOT2022

Fifth Iberian Robotics Conference

Zaragoza November 23-25, 2022

The Organizing Committee of the Fifth Iberian Robotics Conference (ROBOT2022), held in Zaragoza on November 23-25, 2022, hereby declares that the paper entitled

#8284 Robot at Factory Lite - A step-by-step educational approach to the robot assembly

authored by

Luiz Luiz, Leonardo Pilarski, Käis Baidi, João Braun, André Oliveira, José Lima and Paulo Costa

has been presented at this edition of the conference

[illegible]

Danilo Tardioli
General Chair
On behalf of the Organizing Committee

Figure A.1: Certificate of participation in the production of an academic article during the development of this work.