A Yolo-NAS Based Approach for Door and Door Handle Detection and Integration of a Euclidean Geometric Model for Grip Point Localization: An Application In Robot Navigation.

Laura Milena Muñoz Amaya

A Yolo-NAS Based Approach for Door and Door Handle Detection and Integration of a Euclidean Geometric Model For Grip Point Localization : An Application In Robot Navigation.

Dissertation submitted in partial fulfillment of the requirements for the Degree of Master of Science in Geospatial Technologies

February 20, 2024

Laura Milena Muñoz Amaya
lm.munozamaya@uni-muenster.de
https://github.com/laura-munoz

Supervised by:
Prof. Dr. Raúl Marín Prades
Dep. Computer science and Engineering
Universitat of Jaume I

Supervised by:
Prof. Dr. Marco Painho
Nova Information Management School
Universidade Nova de Lisboa

Co-supervised by:
Prof. Dr. Jorge Mateu
Dep. of Mathematics
Universitat of Jaume I

UNIVERSITAT JAUME I

NOVA IMS Information Management School

ifgi Institute for Geoinformatics University of Münster

Declaration of Academic Integrity

I hereby confirm that this thesis on *A Yolo-NAS Based Approach for Door and Door Handle Detection and Integration of a Euclidean Geometric Model For Grip Point Localization: An Application In Robot Navigation.* is solely my own work and that I have used no sources or aids other than the ones stated. All passages in my thesis for which other sources, including electronic media, have been used, be it direct quotes or content references, have been acknowledged as such and the sources cited.

February 20, 2024

I agree to have my thesis checked in order to rule out potential similarities with other works and to have my thesis stored in a database for this purpose.

February 20, 2024

Acknowledgements

I am deeply grateful to Professor Raúl Marín for generously opening the doors to the world of robotics with his vast experience. His belief in my curiosity and willingness to learn, access to his laboratory, and the support of his trusted students have been fundamental. I appreciate the creative freedom he allowed me, thoroughly enjoying working alongside him. Under his mentorship, I have explored the fascinating world of robotics, thanking him for inspiring me to start this path with passion and determination in this field.

In sincere appreciation, I extend my thanks to Professor Jorge Mateu, my co-supervisor, for his consistent motivation and interest in my project. His expert guidance and valuable contributions have been crucial to the integrity and success of my research, enriching my academic experience.

I express special gratitude to my friends from CIRTESU, Amparo, Salvador, Josep, for their sincere interest and valuable assistance in my project. Their willingness and prompt responses not only simplified the process but also made it more rewarding. I also thank my great friend Akshay, whose company made my master's journey highly enjoyable; his support and friendship have been fundamental pillars in this academic venture.

I want to express immense thanks to the Erasmus Mundus program for providing this extraordinary life experience. Through this program, I grew professionally, learned, and broadened my perspective through countless lived experiences. It has been one of the best experiences so far, fostering personal and professional growth. I consider this program instrumental in my holistic development, valuing it as a crucial stage in my trajectory.

My sincere thanks to all the professors who brought passion and dedication to their classes, contributing to my comprehensive professional development. I greatly value their influence on my educational journey.

I express profound gratitude to my friends in Colombia and those I met during the master's program worldwide. Special thanks to my mother, father, grandma, sister, Uriel, and David. Their unwavering support and love have been my greatest motivation, guiding me to the person I am today and providing the tools to fully enjoy life. Their influence has been a constant beacon on my journey, and I am eternally grateful for their presence in my life.

Finalmente, sobre todo gracias a todas mis ancestras a lo largo de la historia, quienes abrieron y construyeron el camino para que yo hoy pueda disfrutar de tan grandes beneficios! Gratitud infinita a todo!

Contents

1 Introduction **1**

1.1 Related work . . . 1

1.2 Motivation . . . 2

1.3 Aim and Objectives . . . 3

1.4 Outline . . . 3

2 Background **4**

2.1 Computer Vision . . . 4

2.1.1 Convolutional Neuronal Network (CNN) . . . 5

2.1.2 Object Detection and Localization . . . 6

2.1.3 Yolo Architecture . . . 6

2.1.4 Yolo NAS . . . 7

2.2 Euclidean Geometry . . . 8

2.3 System of Reference . . . 8

2.4 Lever Principle . . . 9

2.5 Graphics Processing Unit (GPU) . . . 9

2.6 Robot Operating System (ROS) . . . 9

3 Methodology **10**

3.1 Model Design . . . 10

3.2 Data Input . . . 11

3.3 Proposed Methodology . . . 13

3.3.1 Yolo NAS model . . . 13

3.3.2 Euclidean geometric model . . . 14

3.3.3 Implementation in Robot simulation . . . 18

3.4 Evaluation Metrics . . . 19

3.4.1 Precision . . . 19

3.4.2 Recall . . . 19

3.4.3 F1 Score . . . 20

3.4.4 Mean Average Precision . . . 20

3.4.5 Confusion Matrix . . . 21

3.5 System Setup . . . 21

3.5.1 Python . . . 22

3.5.2 Ubuntu (v. 20.04) . . . 24

3.5.3 ROS (v.1.15 Noetic) . . . 24

3.5.4 AndroidROS1 Exercise1 (v.1) . . . 24

4 Data and experimental design **25**

4.1 Data acquisition and Data collection . . . 25

4.2 Data Labeling . . . 28
4.3 Data preprocessing and Data augmentation . . . 29
4.4 Experimental design . . . 30
4.4.1 Dataset Splitting . . . 30

5 Results and Discussion **32**
5.1 Yolo NAS Model Training . . . 32
5.2 Model Comparision . . . 35
5.3 Euclidean Geometry Model. . . . 38
5.4 Comparison Euclidean model output: Predicted and manual labeling 39
5.5 Model implementation on Robot Brain simulation. . . . 40
5.6 Discussion . . . 42

6 Conclusions **45**
6.1 Limitations . . . 47
6.2 Future Work . . . 47

Appendices **47**

A Data Repository created **48**
A.1 Door and Handle door detection . . . 48
A.2 Door and Handle Detection for testing . . . 49
A.3 Just Handle door detection . . . 50

B Code **51**
B.1 Yolo Nas L model . . . 51
B.2 Yolo v8 . . . 54
B.3 Euclidean Geometric Model . . . 55

List of Tables

2.1 Yolo NAS sizes and params. 8

3.1 Confusion Matrix. 21

4.1 Camera settings. 26
4.2 Data splitting. 30

5.1 Key Parameters Yolo NAS model 32
5.2 Results comparison between Yolo NAS m and Yolo NAS l. 33
5.3 Results model comparison between Yolo NAS l, Yolov8s and Yolo NAS only 1 class: Handle. 36

List of Figures

2.1 CNN sequence to classify images(Saha, 2018). 5
2.2 Simplified Yolo workflow (Rosebrock, 2018). 6
2.3 Yolo NAS Architecture. The system utilizes AutoNAC, a Neural Architecture Search (NAS) system, to automatically discover the architecture. This process aims to achieve a balance between latency and throughput in the system (Terven & Cordova-Esparza, 2023). 7

3.1 Thesis implementation workflow. 11
3.2 (a) Elongated door handle. refers to the type of preference for the input model. (b) contrasted luminosity. changes on light around of the door. (c) Blurry door. refers to difficulties of see door frame. (d) dark image. images with a low luminosity. 12
3.3 Bounding box coordinates: Where (w) is the width, (h) is height, Xmin is x axis coordinate of the bottom left corner, and Y min is the y axis coordinate of the bottom left corner of the bounding box. 14
3.4 (a) Original bounding box coordinates. (b) Bonding box coordinates after applying the system of reference. 15
3.5 Center of bounding box calculation. 16
3.6 Grip point calculation from center of bounding box when the door handle opens to the right side. 17
3.7 Grip point calculation from center of bounding box, when the door handle opens to the left side. 17

4.1 (a) UJI hospital simulation. (b) indoor UJI location. 26
4.2 (a) Glass door with high contrast. (b) Door and wall share the same color, making it challenging to distinguish the door frame. . 27
4.3 Door handle type considered in this study. 27
4.4 Labeling process in Roboflow. 29
4.5 Preprocessing and Data augmentation. 30

5.1 Intersection over union (IoU). Calculation of IoU is equal to dividing the intersection of two boxes (real object localization and detected box) by area of union between of these boxes. 33
5.2 Confuxion Matrix - Yolo NAS l. 34
5.3 Output predictions Yolo NAS l model. (a) Wall and door with same color and texture. (b) Glass door. (c) Hospital door handle type. 35
5.4 Yolo NAS l output. 37
5.5 Yolo v8m Output. 37
5.6 Yolo NASm Handle class. 38

5.7 Reference frame of predicted coordinates. 38
5.8 Grip point for door handle that opens to left size generated for the euclidean geometric model, with coordinates in same framework. . 39
5.9 Coordinates bounding box done manually. 40
5.10 Output Euclidean model. (a) Image from predicted model. (b) Image from labeling manual. 40
5.11 Input information from Mobil read in robot view simulation. . . . 41
5.12 Importing and Invoking Autobackend library. 41
5.13 Model keys. 42
5.14 Model yolov8 uploaded in robot simulation for verify the script. . 42

A.1 Door and handle door dataset. 48
A.2 Test door and handle dataset. 49
A.3 Test door and handle dataset. 50

List of Acronyms

AutoNAC	Automated Neural Architecture Construction
CNN	Convolutional Neural Network
CPU	Central Processing Unit
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
HRS	Human Support Robot
ISO	International Organization for Standardization
IoU	Intersection over Union
NAS	Neural Architecture Search
RAM	Random Access Memory
ROI	Regions of Interest
ROS	Robot Operating System
UAVs	Unmanned aerial vehicles
UJI	Universitat Jaume I
YOLO	You Only Look Once
VM	Virtual Machine

Abstract

Object detection and localization play a significant role in artificial intelligence, as they facilitate understanding of the surrounding environment. While architectures designed for this purpose have proven promising and continue to advance, certain objects, such as doors and door handles, have not been extensively explored. Recognizing these specific objects is crucial for autonomous decision-making, especially in robotics, enabling safe and efficient interaction in dynamic environments like hospitals. Decision-making regarding particular objects, such as doors and their handles, involves the robot executing specific actions, such as opening a door. However, achieving this objective goes beyond merely identifying the object; the robot needs information on how to interact with it. In the case of handles, this involves indicating to the robot the specific grip point to open the door. In this study, the novel YOLO NAS architecture, unexplored in these objects, was trained. The results demonstrated remarkable effectiveness in detecting true positives, with a recall of 0.99. However, a lower precision was observed compared to the reference YOLO v8 version. It is noteworthy that despite the lower precision, the visual performance of the model was notable, successfully detecting doors and handles under challenging conditions of light, contrast, and other relevant considerations considered during the study.

A distinctive aspect of this work is the integration of a model based on Euclidean geometry for locating the grip point. Unlike previous studies that typically place this point at the centroid of the handle, the proposed model positions it at the ends of the handle, thus leveraging effective force to open the door with potentially less effort. For the generation of this Euclidean model, the predicted bounding boxes by the YOLO NAS model serve as input. Additionally, the detection model was integrated with the Euclidean model in a robotic simulation using ROS. This implementation allowed for the identification and analysis of challenges that may arise when applying such recent architectures in robotic simulation environments.

Keywords: Computer vision, Object detection and localization, Yolo, YOLO NAS, Euclidean Geometry, Door and Handle Detection, Grip Point Localization, lever principle, Robotics, Navigation Robots, Hospital Environments, ROS

Chapter 1

Introduction

1.1 Related work

Continuous technological advancements aim to enhance the autonomy of robots, enabling them to carry out various tasks without human (Valtchev et al., 2014). Numerous studies across various domains have been conducted to achieve this goal. Autonomy, exemplified by the delivery of meals in hospitals through navigating robots, involves a sequence of steps. This includes tasks such as door opening, which entails more specific subprocesses, such as door and handle recognition, gripping point detection (Fernández-Caramés et al., 2014). The interpretation of these tasks by the robot's system, as addressed in this project. Despite various approaches in studies to execute these tasks, many solely focus on object detection without interaction, leading to uncertainties regarding the implementation and interaction of these models.

In contrast, this project conducts a thorough search to identify initiatives that address not only object detection but also their interaction and implementation. Two notable studies were found. The first study focuses on indoor environments to detect doors and cabinets. It utilizes two Convolutional Neural Network (CNN) methods: the first, based on the YOLO architecture without specifying the version, and the second, based on k-means of point clouds obtained with Kinect to generate handles. Point extraction was performed within 80 or 20 cm from the Region of Interest (ROI) to enhance processing times. These two methods were then merged. The gripping point detection was carried out by importing the Movelt library, which operates by detecting the center of the object. Despite achieving a 90% successful detection rate, it presents limitations related to the robot's distance to the door and the generation of false positives from a distance greater than 1.5 meters (of Electrical & Engineers, n.d.).

The second study is applied in the cleaning and maintenance of door handles for sanitary purposes, aiming to overcome challenges posed by disinfection during COVID-19. It utilizes the Human Support Robot (HSR) equipped with an integrated camera and LIDAR 2D to capture environmental information. The detection model in this study is based on the YOLOv3 architecture. The cleaning point is determined by calculating the centroid of the Bounding Box generated by the model. This study achieves a 95% detection rate for true positives and does not delve into the implementation of the model on the robot (Ramalingam et al., 2020).

With the aim of addressing the lack of specific studies employing the YOLO NAS architecture for door and handle detection, studies focusing on the implementation of this architecture in various contexts were found. The investigated scenarios include smoke and fire detection as well as the detection of rust presence in beans.

The initial study conducts a comprehensive evaluation of YOLO architectures, spanning from YOLO v5 to YOLO NAS, specifically targeting smoke and fire detection. The evaluation emphasizes the recall metric, deemed relevant in this context. The results unveil YOLO NAS as outstanding in effectively minimizing missed detections. However, it exposes lower precision compared to other architectures, ranking as the lowest among all evaluated models, with a precision of 0.082 and a high recall of 0.97 (Casas et al., 2023).

The second study addresses fire detection and geolocation using aerial images from UAVs, employing the YOLO v8 and YOLO NAS architectures. Three classes are identified: fire, non-fire, and smoke. The results demonstrate that YOLO NAS emerges as the superior model, achieving a mAP of 0.71 and a recall of 0.66. The model is implemented in hardware, integrated into a drone equipped with a Pixhawk microcontroller (Choutri et al., 2023). In the last study, the focus is on rust detection in beans using the YOLO v8 architectures. The results reveal a precision of 95% a mAP of 93.7% and a recall of 90.3%This study holds relevance when analyzing the YOLO v8 version in the context of this project, allowing for a comparison of its effectiveness with that of YOLO NAS.

1.2 Motivation

The underlying motivation of this project stems from the limited presence of research utilizing recent object detection and localization architectures for identifying handles and doors, crucial elements in the performance of autonomous robots. This study stands out significantly by employing the advanced YOLO NAS version and incorporating principles of Euclidean geometry for grip point localization. Special attention is given to optimizing the force required by the robot to open the handle by shifting the grip point from the centroid to the end of the handle. This approach is grounded in Archimedes' lever principle, which asserts that applying force at the end of a rotation axis generates a greater moment of force (Bunn, 2017). Consequently, the robot can exert potentially less effort by identifying the grip point at the end of the handle.

Additionally, this study explicitly addresses the inherent limitations of implementing such a recent architecture in a robotic simulation through ROS. By confronting these specific challenges, the aim is to enhance the robots' ability to perform door-opening tasks in hospital environments. This specific setting is configured as the targeted study location for this project, considering its relevance and applicability in real-life scenarios. The uniqueness of this research lies in its comprehensive approach, from the selection of the architecture to the consideration of mechanical aspects and the complexities of robotic simulation, all with the purpose of advancing autonomy and efficiency in robots performing these critical tasks of recognition and door opening.

1.3 Aim and Objectives

This study aims to detect doors and doors- handles through convolutional neuronal network, based on Yolo Nas architecture, and develop a model based in Euclidean geometry for localize a grip point on door handle using the bounding box coordinates. Additional works in the integration of these models in robotic simulation system, that it will have potential application in enhancing robotics interactions.

This study will attempt to answer the following questions:

- How can the effective implementation of the Convolutional Neural Network model – YOLO NAS address challenges related to the detection and localization of doors and door handles across diverse conditions within indoor environments?
- How can the implementation of the Euclidean geometry contribute to the effective localization of a grasp point on the door handle?
- Where are the failure points and limitations when deploying NAS-YOLO handle detection with grip point modeling on simulation robot systems?

1.4 Outline

The thesis is structured as follows:: Chapter 2 offers a theoretical foundation for the methodological concepts that underpin this study. In Chapter 3 the methodology for the primary component of this research door and handle detection, grip point localization, and implementation in robot simulation is elaborated in detail. In Chapter 4 outlines the data and experimental design. In Chapter 5, the results are interpreted and discussed. Finally, Chapter 6 concludes the findings of this thesis.

Chapter 2

Background

In Chapter 2 of this thesis, an in-depth exploration of fundamental concepts sets the stage for the methodology employed. The first Section 2.1 Computer Vision, delves into key terms integral to understanding the visual processing aspects crucial to the study. Following this Section 2.2 Euclidean Geometry, provides a geometric principles used in the grip point localization. Section 2.3 navigates through the System of Reference The subsequent Section 2.4 Lever Principle, elucidates the mechanical principle shaping the approach to grip point optimization. Section 2.5 explores GPU, shedding light on the computational powerhouse essential for efficient processing in computer vision tasks. Lastly, Section 2.6 introduces ROS (Robot Operating System), highlighting its role in orchestrating the integration of components within the robotic simulation framework.

2.1 Computer Vision

Computer vision, an integral discipline within the field of artificial intelligence, focuses on endowing machines with the capability to interpret and comprehend visual information from their surroundings. Through the utilization of specialized algorithms and models, computer vision addresses a myriad of tasks, ranging from object detection and classification to image segmentation and the recognition of intricate patterns. Its impact extends across diverse domains, including medicine, industrial automation, and augmented reality (S. Xu et al., 2021).

In the realm of academic discourse, computer vision assumes a pivotal role due to its capacity to process visual data efficiently. The algorithms employed in computer vision algorithms play a fundamental role in deciphering complex visual information, contributing to advancements in fields such as image analysis and pattern recognition. The interdisciplinary nature of computer vision underscores its significance in fostering innovative solutions for real-world challenges, further reinforcing its role as a cornerstone in the broader landscape of artificial intelligence research and application (Singh et al., 2022).

2.1.1 Convolutional Neuronal Network (CNN)

Convolutional Neural Networks (CNNs), a class of deep learning models, have emerged as a cornerstone in the field of computer vision, revolutionizing the landscape of image processing and pattern recognition. Developed to mimic the human visual system, CNNs excel in tasks like image classification, object detection, and facial recognition (Dhillon & Verma, 2020).The architecture of a CNN is characterized by layers designed to capture hierarchical features from input images. Convolutional layers, with learnable filters, perform feature extraction by convolving input images with these filters. Subsequent pooling layers reduce spatial dimensions, enhancing computational efficiency and translational invariance. Fully connected layers at the end of the network consolidate high-level features for classification or regression tasks (Li et al., 2022). CNNs exhibit notable adaptability to varying complexities of visual data. Transfer learning, a prevalent technique, allows pre-trained CNNs to be fine-tuned for specific tasks with limited data, expediting model convergence and enhancing performance. This adaptability extends to diverse domains, from medical imaging for disease diagnosis to autonomous vehicles for object detection and tracking. (Krichen, 2023).

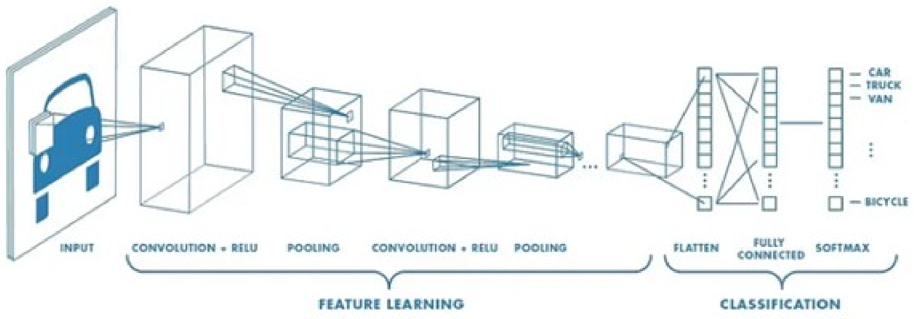


Figure 2.1: CNN sequence to classify images(Saha, 2018).

One of the pivotal achievements of CNNs lies in their ability to automatically learn hierarchical representations, obviating the need for manual feature engineering. This end-to-end learning paradigm empowers CNNs to discern intricate patterns, providing a robust framework for tasks like image segmentation, where delineating objects within images is imperative (Krichen, 2023).

However, the efficacy of CNNs is not without challenges. As models grow in complexity, the risk of overfitting increases, necessitating regularization techniques. Moreover, interpretability remains a concern, as the black-box nature of deep learning models raises questions about understanding their decision-making processes, especially in critical applications such as healthcare. In the realm of academic research, CNNs have spurred groundbreaking advancements, fostering interdisciplinary collaborations and contributing to the evolution of artificial intelligence (Tian, 2020). The continuous refinement of CNN architectures, coupled with innovations like attention mechanisms and neural architecture search, underscores their dynamic nature and ongoing relevance in pushing the boundaries of what is achievable in computer vision (Lin et al., 2021; Tian, 2020).

2.1.2 Object Detection and Localization

Object detection and localization represent pivotal tasks in computer vision, playing a foundational role in the development of intelligent systems capable of comprehending and interacting with the visual world. These tasks are essential components in applications ranging from autonomous vehicles and robotics to medical imaging and surveillance (Lecrosnier et al., 2021).

Object detection involves identifying and classifying multiple objects within an image or video, marking their locations with bounding boxes. This complex challenge has been addressed through various methodologies, with deep learning, particularly Convolutional Neural Networks (CNNs), emerging as a dominant paradigm. CNNs learn hierarchical features that enable them to discern patterns and objects within images, allowing for accurate detection across diverse scenarios. Localization, a subset of object detection, specifically refers to determining the precise location of objects within an image. Techniques like regression analysis are often employed to predict the coordinates of bounding boxes encapsulating detected objects. This spatial awareness is crucial for understanding the spatial context and relationships between objects, enabling more nuanced and contextually aware applications (Zou et al., 2023).

2.1.3 Yolo Architecture

The You Only Look Once (YOLO) architecture stands as a seminal contribution in the domain of computer vision and object detection, offering a paradigm shift in efficiency and accuracy. Developed by Joseph Redmon and Santosh Divvala, YOLO represents a pioneering approach to real-time object detection by addressing the trade-off between speed and precision inherent in traditional methods (B. Xu et al., 2018). YOLO employs a single neural network to process an entire image, negating the need for complex pipelines involving region proposals and subsequent classification. This unique approach enables YOLO to achieve remarkable speed, making it particularly suited for applications requiring real-time object detection, such as video analysis and autonomous vehicles (Jiang et al., 2021).

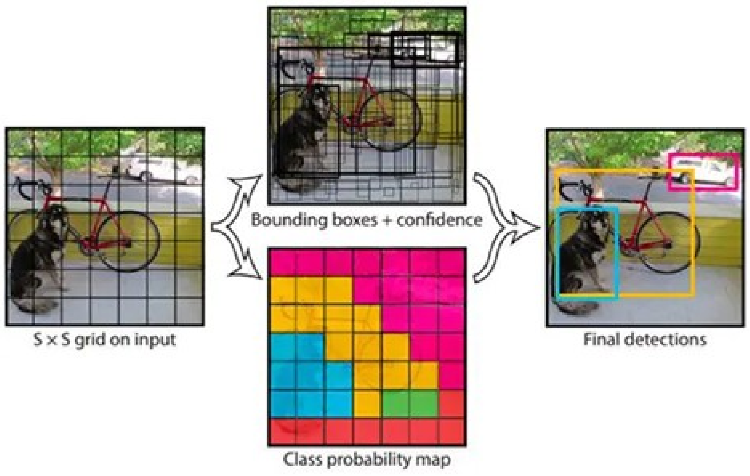


Figure 2.2: Simplified Yolo workflow (Rosebrock, 2018).

A distinctive feature of YOLO is its division of the input image into a grid, each cell of which predicts bounding boxes and class probabilities simultaneously. This grid-based approach enhances spatial precision and facilitates the detection of multiple objects within a single image. The architecture employs anchor boxes to optimize bounding box predictions, offering adaptability to diverse object shapes and sizes. YOLO's architecture is hierarchical, comprising multiple convolutional layers for feature extraction and subsequent fully connected layers for high-level reasoning. The combination of these layers enables YOLO to discern intricate patterns and representations within images, contributing to its robust object detection capabilities (Diwan et al., 2023).

2.1.4 Yolo NAS

You Only Look Once Neural Architecture Search (Yolo NAS) architecture was made for DECi's AutoNAC. This architecture is based on original YOLO but this incorporates and advances that make it different from the last versions of YOLO, this address some limitations like enhance ability to detect small objects, improve localization accuracy and making the model more accessible for real time applications (Deci.ai, 2023).

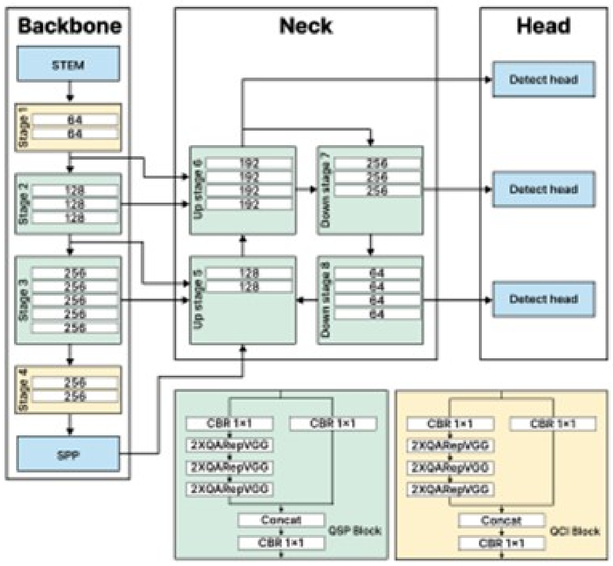


Figure 2.3: Yolo NAS Architecture. The system utilizes AutoNAC, a Neural Architecture Search (NAS) system, to automatically discover the architecture. This process aims to achieve a balance between latency and throughput in the system (Terven & Cordova-Esparza, 2023).

NAS automates the creation of neural network structures using optimization algorithms. The main aim is to find the best balance between model accuracy, computational complexity, and model size. YOLONAS, notably, offers three model architectures that work with different precisions: FP32 (single precision floating point), FP16 (half precision floating point), and INT8 (8-bit integer) (Terven & Cordova-Esparza, 2023).Additionally Given the diverse range of possible applications, YOLO-NAS comes in multiple versions: small (s), medium (m), and large (l). We will delve into each variant, providing detailed insights that are visually presented in Table 2.1 (Casas et al., 2023).

Table 2.1: Yolo NAS sizes and params.

Model	Size (Pixels)	Params (Million)
Yolo NAS- s	640	19
Yolo NAS- m	640	51.1
Yolo NAS- l	640	66.9

These models have different sizes and parameter counts. YOLO-NASS is the smallest model, with 19 million parameters. YOLO-NASI is the largest model, with 66.9 million parameters. The "size" column indicates the resolution of the images, which is 640 pixels in width and height. In general, YOLO NAS can come in various sizes and parameter counts. Smaller models are more efficient but may not be as accurate as larger models. Larger models are more accurate but can be slower and require more computational resources. Additionally, utilizing GPU acceleration, real-time applications have extended these YOLO methods, which typically depend on region proposals. Furthermore, these approaches often incorporate advanced optimization techniques for improved efficiency and speed (Nugraha & Supangkat, 2023)

2.2 Euclidean Geometry

Basic Euclidean geometry, originating from the foundational work of the ancient Greek mathematician Euclid, serves as a fundamental branch of mathematics that explores the properties of space and shapes through axioms and theorems. In an academic context, Euclidean geometry provides a rigorous framework for reasoning and deducing geometric relationships, laying the groundwork for various mathematical disciplines and practical applications. Euclidean geometry operates in a two-dimensional plane and a three-dimensional space, employing points, lines, angles, and polygons as its basic building blocks (Magro & García-Pérez, 2019).

2.3 System of Reference

A basic system of reference, a foundational concept in mathematics and physics, provides a framework for locating and describing objects in space. This system is indispensable in scientific endeavors, offering a standardized approach to specifying positions, directions, and distances. In an academic context, understanding and utilizing a system of reference is fundamental for comprehending spatial relationships, conducting experiments, and formulating mathematical models. A Cartesian coordinate system, conceived by René Descartes, is a prevalent example of a basic system of reference. It employs perpendicular axes labeled x, y, and sometimes z in three-dimensional space. Points are located by specifying distances along each axis, creating an ordered pair (x, y) (Moritz, n.d.).This system facilitates precise geometric representation and mathematical analysis of spatial configurations.

2.4 Lever Principle

The lever principle posits that the product of the force applied to a lever and its perpendicular distance from the fulcrum (moment arm) remains constant in the absence of external torques. This principle is encapsulated in the equation:

$$\tau = rF \tag{2.1}$$

where (τ) represents torque, (r) denotes the moment arm, and (F) is the applied force. The basic lever principle attributed to Archimedes, a foundational concept in classical mechanics, elucidates the equilibrium conditions and mechanical advantage inherent in lever systems. Archimedes, the ancient Greek mathematician and physicist, articulated this principle, emphasizing the pivotal role of torque and force distribution in lever mechanisms (Goe, 1972).

2.5 Graphics Processing Unit (GPU)

A Graphics Processing Unit (GPU) is a specialized electronic circuit designed to accelerate the processing of graphics and parallel computations. Originally developed to enhance the rendering capabilities of computer graphics, GPUs have evolved into highly efficient processors capable of handling parallel workloads. In academic and technical discourse, GPUs are recognized for their crucial role in scientific simulations, artificial intelligence, and high-performance computing (Owens et al., 2008). One distinguishing feature of GPUs is their architecture, optimized for parallel processing. Unlike Central Processing Units (CPUs), which excel in sequential tasks, GPUs boast numerous cores that can simultaneously execute multiple parallel tasks. This parallelism is particularly advantageous in applications involving large-scale data processing and complex mathematical computations (Nickolls & Dally, 2010). The advent of General-Purpose GPU (GPGPU) computing has expanded the scope of GPU applications beyond graphics. Researchers and scientists leverage GPUs for tasks like machine learning, molecular dynamics simulations, and weather modeling. The parallel nature of GPU architecture accelerates these computations, significantly reducing processing times compared to traditional CPU-centric approaches (teinkraus' et al., 2005).

2.6 Robot Operating System (ROS)

The Robot Operating System (ROS) stands as a robust middleware framework designed to facilitate the development and operation of robotic systems. Created by Willow Garage and later maintained by the Open Robotics organization, ROS provides a comprehensive suite of tools, libraries, and conventions that streamline the complexities associated with building and controlling robots (Suárez et al., n.d.). Key features of ROS include a standardized communication protocol, a centralized package management system, and a plethora of pre-built libraries for common robotic functionalities. The platform supports a wide array of robotic hardware and sensors, facilitating interoperability among different robotic platforms and simplifying the development of complex robotic applications (Quigley et al., n.d.).

Chapter 3

Methodology

This chapter outlines the methodology implemented of Yolo NAS architecture, development of the Euclidean geometric model and implementation of models integration in robot simulation. The structure of this chapter is as follows: The first Section 3.1. provides a general model design. Moving forward, Section 3.2.Data Input, elucidates the intricacies of the dataset employed, offering information fed into the model for training and evaluation. Section 3.3. Proposed Methodology, where a detailed account of the steps taken in door and handle detection, grip point localization, and their subsequent integration into the robotic simulation. The subsequent Section 3.4, delves into the Evaluation Metrics, Finally, Section 3.5, System Setup, provides an intricate overview of the configuration of the system.

3.1 Model Design

The collaborative utilization of the YOLO NAS model and the Euclidean geometric model serves a dual purpose: the identification of doors and door handles and the precise determination of the grip point on the door handle. This intricate process strategically employs the predicted coordinates as a pivotal system of reference, significantly influencing the decision-making aspects of the robot's navigation capabilities. The YOLO NAS model exhibits a classification mechanism, categorizing input data into two distinct classes: Doors and Door Handles. The output manifests as an object recognition output, complete with a bounding box featuring corresponding coordinates and an accuracy percentage prediction. These predicted coordinates seamlessly integrate into a system grounded in Euclidean geometry, functioning as key reference points, that effectively enhances the localization accuracy of the grip point within the given image. Figure 3.1 is delineated into three fundamental steps: Input data, model development, and evaluation. In the initial phase of Input data, the focus is on acquiring source information for the models, involving the use of a free database and the capture of visual data through a built-in camera. These data go through a labeling and processing procedure.

The second phase involves the model development, starting with the workspace setup for running libraries and code dedicated to model training. Following the setup, data is loaded and partitioned, followed by the definition of relevant classes: doors and door handles. Hyperparameters are then established, and model training ensues, producing an output that reveals predictions in captured images. Model evaluation primarily employs metrics such as recall and mAP.

Subsequently, attention shifts the development of the Euclidean geometric model to determine the grip point, utilizing coordinates from bounding boxes of the door handle class. Integration of this model with the detection model is assessed through metrics such as recall, mAP, and visualization of grip point coherence in images.

Finally, the implementation of the integrated model in the ROS environment is applied, specifically tailored for robotic applications. Emphasis is placed on the selection of this platform due to its suitability and specialized functionalities in the field of applied robotics.

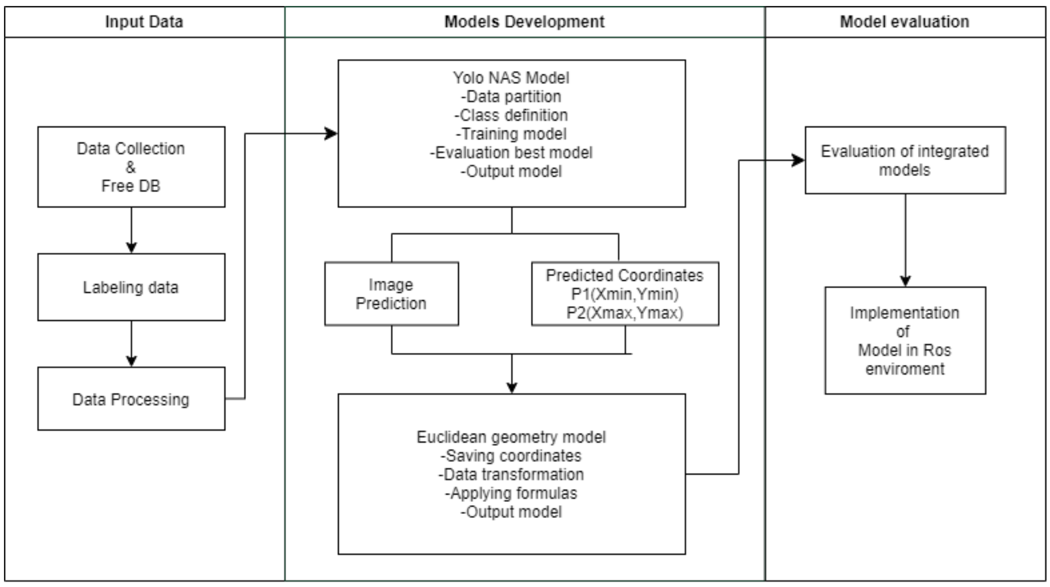

Input Data
Models Development
Model evaluation
Data Collection & Free DB
Labeling data
Data Processing
Yolo NAS Model -Data partition -Class definition -Training model -Evaluation best model -Output model
Image Prediction
Predicted Coordinates P1(Xmin,Ymin) P2(Xmax,Ymax)
Euclidean geometry model -Saving coordinates -Data transformation -Applying formulas -Output model
Evaluation of integrated models
Implementation of Model in Ros enviroment

Figure 3.1: Thesis implementation workflow.

3.2 Data Input

The input dataset designed for the YOLO NAS detection model was meticulously planned, comprising images captured through an integrated camera and an open database. This dataset addresses various conditions, including variations in lighting, shape, texture, distance, door status, and image quality, aiming to ensure the model's robustness views in Figure 3.2.

Concerning lighting conditions, variations were considered to encompass different shades, ranging from high to low luminosity.Regarding shape, emphasis was placed those predominantly showing the door frame. Elongated door handles were preferred, as they are common in hospital environments, where the navigation robot for which these models are developed will be deployed. Additionally, doors with diverse textures and colors were included. Regarding distance, images were captured from the robot's perspective, at an apparent distance of 1 meter from the wall in front of the door, typical for a navigation robot. Images were also taken at farther distances, considering open spaces, and at closer distances, especially at door handles, considering the robot's approach to the door.

It is crucial to note that, for this model, the door's status was considered, limited solely to closed doors, excluding open or partially open doors. From the open dataset, predominantly closed doors were selected, aligning with the navigation robot's objective of opening doors. Regarding image quality, both sharp and blurry, dark images were captured. Images of fictitious or drawn doors were excluded from the open dataset, focusing exclusively on real doors. This approach ensures the authenticity and applicability of the model to real-world situations.

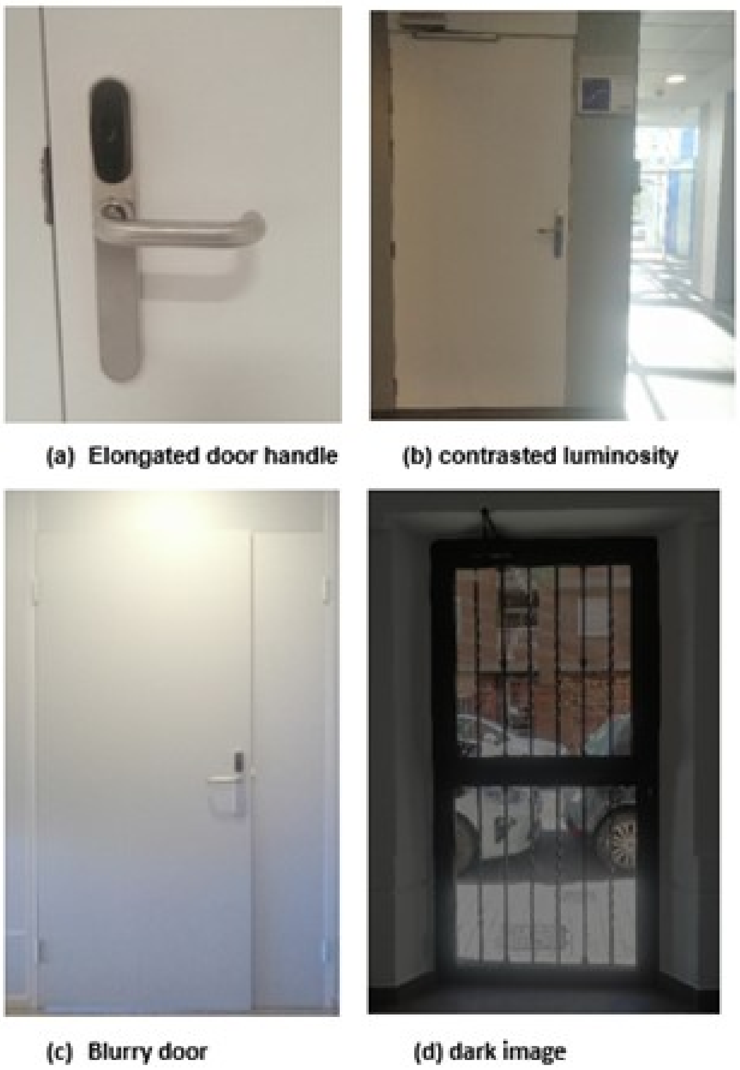


Figure 3.2: (a) Elongated door handle. refers to the type of preference for the input model. (b) contrasted luminosity. changes on light around of the door. (c) Blurry door. refers to difficulties of see door frame. (d) dark image. images with a low luminosity.

During the data preparation phase, a data splitting process is undertaken, creating two distinct sets for the training and validation stages, and a separate set for testing. The test dataset comprises images with more stringent and complex conditions, including variations in luminosity, sharpness, and the factors described above, with the aim of rigorously evaluating the robustness of the model against challenging scenarios.

The distribution of images is structured so that 80% of the total is allocated to the training and validation set, while the remaining 20% is assigned to the test set. In total, the collection encompasses 3,334 images, ensuring a diverse and representative sample for each phase of the process. This partitioning strategy ensures the effectiveness and reliability of the model.

Within the framework of the Euclidean geometric model, the input data consists of the coordinates of the bounding box (4.1), which operates as a reference system grounded in Euclidean geometry. These coordinates play a vital role in supplying crucial information regarding the spatial location and dimensions of detected objects. It is essential to note that, for the geometric model, only the class door handle was considered. This positional information serves as the cornerstone for pinpointing the grip point on the door handle. This, in turn, greatly facilitates the precision and efficiency of robotic interactions within the physical environment.

3.3 Proposed Methodology

In this section, the methodology employed for the three main components will be elucidated: the implementation of YOLO-NAS, the Euclidean geometric model, and its implementation in the robot's brain simulation.

3.3.1 Yolo NAS model

The selected architecture for object detection in this study was YOLO, an architecture developed by Ultralytics known for its high precision in real-time object detection and localization, as well as its good reputation in indoor environments (Jiang et al., 2021).

As a second step, the YOLO-NAS version was chosen to address a gap identified in the literature review, specifically the lack of previous studies on the detection of doors and door handles using this version. The selection of this version is supported by its enhanced ability to detect small-scale objects, which is convenient for detecting door handles. Additionally, one of the main goals of YOLO-NAS is to find a balance between model accuracy, computational complexity, and model size (Casas et al., 2023). This is crucial, considering the planned implementation in a navigation robot, where processing efficiency is essential.

The architecture configuration for this project included two classes: "door" and "door handle", and the model size that yielded the best results was "large". Key hyperparameters were configured with the Adam optimizer, and maximum of 30 epochs.

3.3.2 Euclidean geometric model

The methodology employed in the development of the Euclidean model starts with the selection of coordinates for the "door handle" class generated from predictions made by the YOLO-NAS model. These coordinates include the two-dimensional bounding box's bottom-left corner (x, y), as well as its width and height Figure 3.3. By utilizing a reference system, we can derive the coordinates of the top-right corner.

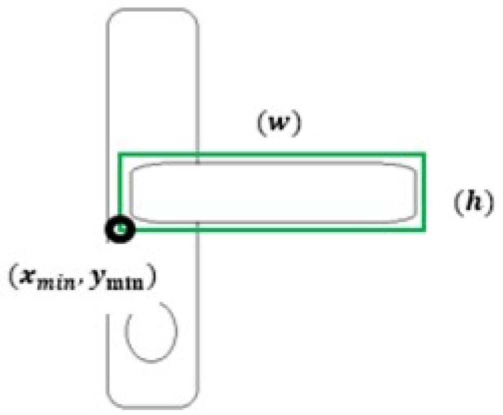


Figure 3.3: Bounding box coordinates: Where (w) is the width, (h) is height, Xmin is x axis coordinate of the bottom left corner, and Y min is the y axis coordinate of the bottom left corner of the bounding box.

$$\begin{aligned}
&\textit{Bounding Box Coordinates} = (X_{min}, Y_{min}, W, h) \\
&Where: \\
&\qquad (X_{min}, Y_{min}) : \textit{Coordinates of the bottom left corner} \\
&\qquad\qquad \textit{W: Width of Bounding Box} \\
&\qquad\qquad \textit{h: Height of Bounding Box} \\
&\textit{Bounding Box Coordinates} = [\{(x,y) \mid x_{\min} \le x \le x_{\max},\, y_{\min} \le y \le y_{\max}\}] \\
&\quad \textit{So, any point (x,y) within this range will be inside of bounding box.}
\end{aligned} \tag{3.1}$$

The reference system of the YOLO-NAS model is based on taking the bottom-left corner of the image as the origin coordinates [0,0]. From this reference point, the positions of the bounding boxes that delineate the regions of interest in object detection are generated. This choice allows establishing a coordinate framework that facilitates the precise representation of the location and dimensions of the detected objects in the image, thereby contributing to the effectiveness of the model in the detection task. Formula 3.2 explains how the reference system is utilized to obtain the coordinates of the top-right corner figure 3.4

$$\begin{gathered}\textit{Applying system of reference:}\\ \textit{The X and Y coordinates of the upper right corner are equal to:}\\ x_{\max} = x_{\min} + W\\ y_{\max} = x_{\min} + h\end{gathered} \tag{3.2}$$

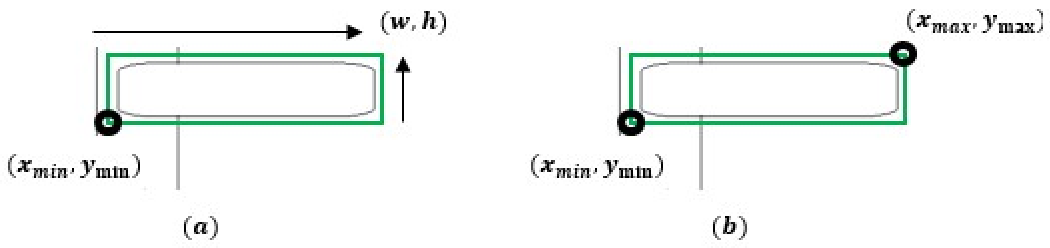


Figure 3.4: (a) Original bounding box coordinates. (b) Bonding box coordinates after applying the system of reference.

The coordinates are generated within a range of 0 to 1. However, Output images have a size of 640x640 pixels, indicating that the coordinates of the image and the bounding box are in different reference frames. Therefore, a coordinate transformation is necessary to align them in the same reference frame as the images formula 3.3.

$$\begin{gathered}\textit{The transformation of the coordinates } x'_{\min}, y'_{\min}, x'_{\max}, y'_{\max} \textit{is given by:}\\ x'_{\min} = x_{\min} X \textit{ image width}\\ x'_{\max} = x_{\max} X \textit{ image width}\\ \textit{And}\\ y'_{\min} = y_{\min} X \textit{ image height}\\ y'_{\max} = y_{\max} X \textit{ image height}\\ \textit{where:}\\ \textit{image width} = \textit{640px}\\ \textit{image height} = \textit{640px}\end{gathered} \tag{3.3}$$

Up to this point, the coordinates are in the same frame of reference as the original images, which is that of the original images. The main goal of the Euclidean model developed in this work is to calculate the gripping point on the door handle to ensure precise interaction of the robot with the handle.

The calculation of the gripping point is based on basic principles of Euclidean geometry. It starts from a plane represented by a 2D image of the door handle with coordinates (x, y). In this way, the gripping point is calculated using the coordinates of the bottom-left corner and the top-right corner of the predicted bounding box to determine the center of the bounding box Figure 3.5. After obtaining the coordinates of the center, the gripping point is calculated, positioned halfway between the center of the bounding box and the limit where the end of the door handle is visible formula 3.4.

Image bounding box center:

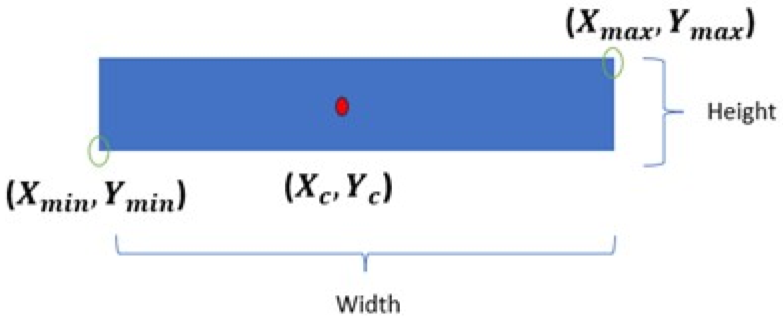


Figure 3.5: Center of bounding box calculation.

Calculating bounding box center:

$$x_c = \frac{x'_{\text{min}} + x'_{\text{max}}}{2}, \quad y_c = \frac{y'_{\text{min}} + y'_{\text{max}}}{2} \tag{3.4}$$

$$where:$$

$$(x_c, y_c) : \textit{Coordinates of center of Bounding box}$$

It is important to note that the door handle can open to the right side figure 3.6 and this has its own formula 3.5 or to the left side figure 3.7 that also has its own formula 3.6. The coordinate on the y-axis remains constant, staying in the center of the bounding box, regardless of the direction in which the door handle opens. This consideration is essential to ensure consistency in determining the gripping point.

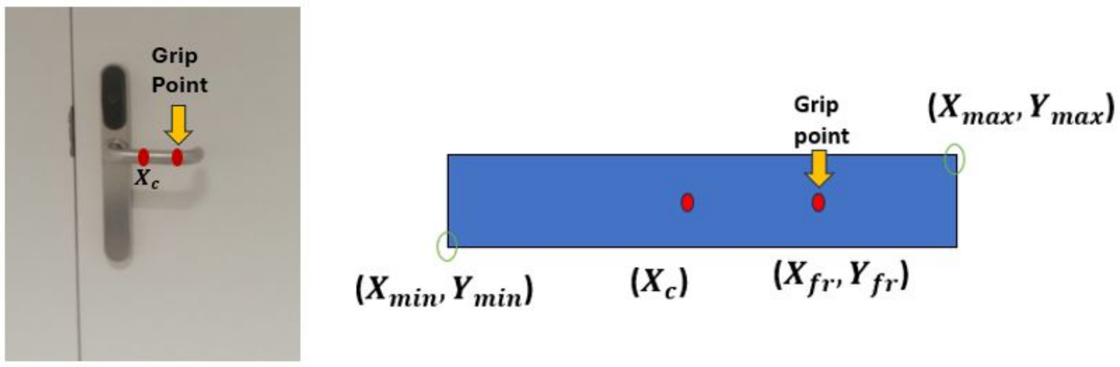


Figure 3.6: Grip point calculation from center of bounding box when the door handle opens to the right side.

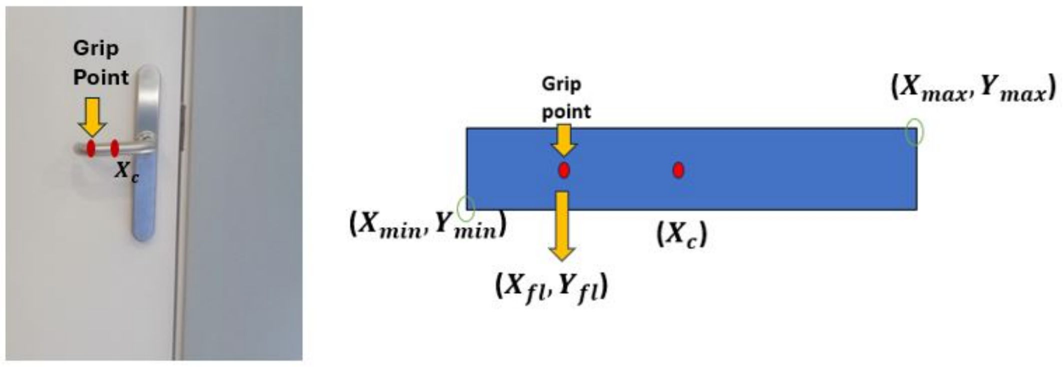


Figure 3.7: Grip point calculation from center of bounding box, when the door handle opens to the left side.

Calculation of the grip point, When the door handle opens to the right side:

$$\begin{gathered} x_{\mathrm{fr}} = \frac{x_{\mathrm{c}} + x'_{\mathrm{max}}}{2} \\ \textit{where:} \\ (x_{\mathrm{fr}}, y_{\mathrm{c}}) : \textit{Grip point position in the right side} \end{gathered} \tag{3.5}$$

Now, Calculation of the grip point, when the door handle opens to the left side:

$$\begin{gathered} x_{\mathrm{fl}} = \frac{x'_{\mathrm{min}} + x_{\mathrm{c}}}{2} \\ \textit{where :} \\ (x_{\mathrm{fl}}, y_{\mathrm{c}}) : \textit{Grip point position in the left side} \end{gathered} \tag{3.6}$$

Finally, after calculating the grip point using Euclidean geometry and considering the opening direction of the door handle, the next step is to visually highlight this point on the prediction image. This marking is crucial to provide visual information about the exact point and area where the robot should interact with the door handle.

3.3.3 Implementation in Robot simulation

This section focuses on integrating the previously developed models: YOLO-NAS for object detection and the Euclidean model for calculating the grip point. The main goal is to implement this integrated model in a simulation, as if it were being directly applied to the robot. To achieve this, a virtual environment was set up with the Ubuntu 20.04 operating system, compatible with the Robot Operating System (ROS) software development framework for robots.

In the initial stage, a Google Colab notebook is created, configuring the working environment by importing the necessary libraries and modules 3.5. In this notebook, the pretrained model in the .pth format is loaded. This file stores both the weights and bias values of the convolutional layers in the model, representing the learned capacity during training to detect and locate doors and door handles in the images from the datasets created for this project.

A random image is selected from the created dataset, and the pre-trained object detection model is applied to this image, generating its corresponding output. This output consists of the image with the detection and predicted bounding box coordinates, which are accessed and saved. Subsequently, only the coordinates associated with the "door handle" class are selected, and the aforementioned Euclidean model is applied. In this process, a coordinate transformation is performed by multiplying them by the width and height of the original image (640x640) to ensure they are in the same reference frame. Then, grip point localization formulas are applied depending on whether the handle opens to the right formula 3.5 or Left formula 3.6. To determine this orientation, a prompt is included, prompting the user to indicate the side where the handle is visible. Finally, the result is an image with the marked grip point on the door handle. The process was repeated using manually labeled images and bounding boxes to compare and validate the correct functioning of the grip point in the predicted image.

Up to this point, the integration of both models has been carried out. Now, concerning the implementation of these models in the simulation of the robot's brain, the ROS communication platform is employed. Since this tool operates optimally in the Ubuntu environment, Ubuntu was installed on a virtual machine with a RAM capacity of 6 GB. This step is essential to facilitate effective interaction between the models and the robot's operating system during the simulation.

Ubuntu 20.04 was installed along with version 1 of ROS. Additionally, a mobile device was set up to receive information from the images or frames that the robot would capture in its environment. This mobile device, equipped with a built-in camera, had an Android application which name is “exercise”, this is installed to facilitate the connection between the mobile device and the virtual machine (VM) where ROS was installed. This allowed communication between the camera and the robot simulation, i.e., the VM. Both the mobile device and the VM used IP address of the same connected network to establish interaction between them. In the virtual machine, ROS was initiated, and commands were executed to determine the identifier assigned by ROS to the camera. With this information, a Python script was created to activate the robot's camera, meaning that the information captured by the robot was now operating in ROS—essentially being processed by the robot simulator. The validation of this was evident as the information captured by the mobile device from its environment was visualized in the VM.

After confirming that the camera was operational, the recognized identifier of the camera by ROS, typically with a numerical sequence followed by 'compressimage,' was transferred to the script responsible for loading the integrated model. Finally, this script captured camera information and should display real-time recognition of doors and their handles on the environment, marking the gripping point.

3.4 Evaluation Metrics

The performance of the trained model is assessed using the most common metrics found in the literature. These metrics evaluate the accuracy and efficiency with which the model detects doors and door handles. Below, a brief description of the metrics used in our evaluation is provided.

3.4.1 Precision

Precision is a metric that evaluates the proportion of correct predictions made by the model in relation to the total predictions. In the context of door and handle detection, precision indicates how many of the locations predicted by the model genuinely correspond to real doors and handles. For example, if the model correctly identifies 9 out of 10 doors, its precision would be 90%İt can be stated that precision focuses on successful cases (PETERFLACH, n.d.).

Mathematically, precision is represented by the number of true positive cases (TP) divided by the sum of (TP) plus the number of false positive cases (FP), which are cases the model identifies as negative but are positive (Paiva et al., 2020), as depicted in the following formula 3.7:

$$Precision = \frac{TP}{TP + FP} \tag{3.7}$$

3.4.2 Recall

The recall, also known as sensitivity, measures the model ability to identify and capture all positive instances in the dataset. It specifies the proportion of positive instances that the model successfully detects in relation to the total positive instances present. In simple terms, recall emphasizes the model capability to avoid missing positive cases, prioritizing the minimization of false negatives. This is particularly useful in studies where false negatives can have significant implications (Vujović, 2021).

Mathematically, recall is given by the number of true positive cases (TP) divided by the sum of true positive and false negative cases (FN). As illustrated in formula: 3.8

$$Recall = \frac{TP}{TP + FN} \tag{3.8}$$

Applied to the context of this study, recall would be crucial to ensure that the model does not overlook any existing doors or handles. A high recall would indicate that the model is effective in identifying many doors and handles present in the environment, even at the risk of having more false positives.

3.4.3 F1 Score

The F1 score is a metric that combines both precision and recall, providing a balanced evaluation of a model performance. It is particularly useful when there is a need to strike a balance between minimizing false positives and false negatives (Zhao & Li, 2020).

Mathematically, the F1 score is represented as 2 times the product of precision and recall divided by the sum of precision and recall. This harmonic mean penalizes extreme values, making it a suitable choice for scenarios where a balanced trade-off between precision and recall is desired. (Foroughi et al., 2021).The formula for the F1 score is expressed as follows:

$$F1 = 2x\frac{PrecisionxRecall}{Precision + Recall} \tag{3.9}$$

In the context of door and handle detection, the F1 score considers both the precision, which measures the accuracy of positive door handles predictions, and the recall, which evaluates the model ability to capture all positive instances.

3.4.4 Mean Average Precision

The Mean Average Precision (mAP) is a metric commonly used in object detection tasks, providing a comprehensive assessment of a model's accuracy across multiple classes and varying levels of confidence thresholds. It takes into account both precision and recall, considering the precision-recall curve at different confidence thresholds. And for this study case is the metric in which the evaluation of the model's performance is most considered.

The threshold used in this job is mAP@0.5 means that the performance of the model is considered valid if the model has at least 50% confidence in that detection. mAP is defined mathematically by averaging the Average Precision (AP) scores for each class, and each AP is calculated by integrating the precision-recall curve (Choutri et al., 2023). The formula for mAP is expressed as:

$$\begin{gathered} F1 = \frac{\sum_{i=1}^{N} APi}{N} \\ where: \\ N : Numberofclasees \\ Api : AveragePrecisionforeachclassi \end{gathered} \tag{3.10}$$

In the context of door and door handle detection, mAP examines how the model performs at different confidence levels and demonstrates the relationship between precision and recall.

3.4.5 Confusion Matrix

The confusion matrix is an essential tool in model evaluation, providing a detailed insight into its performance. In the context of door and handle detection, this matrix categorizes the model's predictions into four groups: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) (Thesis et al., n.d.).

True positives represent instances where the model correctly identified a door or handle. False positives indicate instances where the model incorrectly predicted the presence of a door or handle when there is none. True negatives reflect instances where the model correctly predicted the absence of doors or handles. False negatives are instances where the model failed to detect a door or handle present in the image.

Table 3.1: Confusion Matrix.

		Actual class	
		True	False
Predicted Class	True	True Positive	False Positive
	False	False Negative	True Negative

3.5 System Setup

For this project, two environments have been configured: Google collab and MVware with ubuntu 20.04. In google collab the notebook was configured with GPU acceleration in order to leverage enhanced computational capabilities, a vital component that expedites the training of our YOLO-NAS model. This Graphics Processing Unit (GPU) plays a pivotal role in parallelizing tasks, significantly boosting processing speed.

Simultaneously, a dedicated environment for seamless communication with the robot was set up using VMware and Ubuntu 20.04. This virtual machine emulates the ROS system, ensuring compatibility and effective data exchange with the robot's operating system. The choice of Ubuntu 20.04 is integral for ROS compatibility, and mobile aplication providing a stable environment.

3.5.1 Python

The proposed methodology is developed in Python, an object-oriented programming language with high-level data structures. Python offers various standard libraries, from string processing to system interfaces, some of which are designed and optimized to handle extensive datasets (Foundation, 2009).

Numpy (v.1.18.1)

Numpy is a library dedicated to scientific computing with Python, offering tools for manipulating N-dimensional array datasets. It uses matrix tools to store, manipulate, and process data (Community, 2020).

cv2 OpenCV (v.3.0.0)

OpenCV (Open Source Computer Vision) provides tools for image processing and computer vision. Its integration is essential for image manipulation and processing in the context of object detection (O. Documentation, 2013).Cv2 allows to work with opencv with python.

Rospy (v. 1.12.2)

Rospy is a Python library that facilitates communication with the Robot Operating System (ROS). This library allows to python developers create and publish topics, subscribe topics, send and receive messages, and more additional task related with ROS. For the project was used ros noetic version (Robotics, 2008b).

`sensor_msgs.msg` (v.1.12.2)

The sensor msgs library is a ROS message library that provides a set of messages to represent sensor data. These messages are used to communicate sensor data between different components of a robotic system (Robotics, 2008a)

CvBridge (v.1.12.2)

CvBridge is a ROS module that acts as a bridge between ROS and OpenCV. It facilitates the conversion of ROS messages to images compatible with OpenCV and vice versa (Robotics, 2009).

Torch (v. 1.10.0)

The Torch library is an open-source machine learning library used for developing deep learning models. It is crucial for the development of robotic systems, such as in this project, where it will be employed in autonomous navigation robots and for the recognition of doors and door handles (T. Documentation, 2021).

Matplotlib (v. 3.8.2)

It is a Python library for creating static, animated, and interactive charts (Hunter, 2003).

Roboflow (v. 1.12.0)

Roboflow is a platform that simplifies the preparation and management of datasets for computer vision models, making a significant contribution to the training of object detection models. The platform consists of a Python library and a web tool (R. Documentation, 2022).

Super Gradients (v. 0.5.0)

The Super Gradients library is a Python library that provides an API for training deep learning models with supergradients. Supergradients are an optimization technique that allows you to train deep learning models more efficiently and accurately (S. company, 2023).

Ultralytics (v. 2.8.0)

Ultralytics is a Python library for training and deploying YOLO models. This library provides functions such as data loading and processing, model creation, model training, model evaluation, model deployment, and more. In this project, the library was used with Google Colab and the Ubuntu operating system for its implementation with ROS. The version mentioned is for its operation with ROS (U. company, 2019)

3.5.2 Ubuntu (v. 20.04)

Ubuntu 20.04 is an open-source Linux distribution that works with ROS1 and with versions that are compatible with the mobile app installed for communication between the mobile device and the MV (Ltd., 2020).

3.5.3 ROS (v.1.15 Noetic)

It is an open-source software framework for the development of robots and robotic systems, with functionalities such as the ability to create virtual robots. For this study, the ROS Noetic version was installed for compatibility (Robotics, 2020).

3.5.4 AndroidROS1 Exercise1 (v.1)

This mobile app was developed by the CIRTESU team at the Jaume I University with the purpose of establishing a connection between the integrated cameras in Android operating systems and ROS. This app just works in android OS.

Chapter 4

Data and experimental design

This chapter delineates the materials utilized in the project concerning data. The structure of the chapter is organized as follows: Section Section 4.1 addresses Data Acquisition and Collection, Section Section 4.2 delves into Data Labeling, Section Section 4.3 focuses on preprocessing and data augmentation, and, lastly, Section Section 4.4 discusses Data Splitting.

4.1 Data acquisition and Data collection

Throughout the course of this project, two types of datasets were utilized. The first one was a self-created dataset, while the second one was obtained from a freely accessible database. This section aims to elucidate the process of acquiring both datasets, and data acquisition for Euclidean model.

Regarding the self-created dataset, images were primarily sourced from a hospital simulator situated at Jaume I University. The collection also encompassed various locations within the institution, including hallways, study rooms, bathrooms, and other relevant spaces. The meticulous data-gathering process was designed to provide the dataset with a comprehensive and diversified representation of both clinical and non-clinical environments. Consequently, this approach contributed to the robustness and heterogeneity of the database employed in the project development figure 4.1.

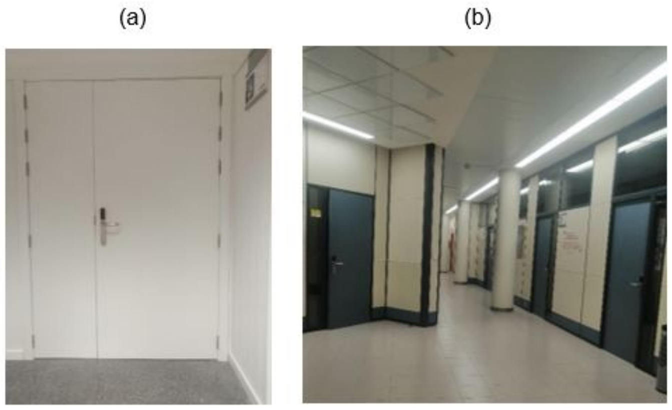


Figure 4.1: (a) UJI hospital simulation. (b) indoor UJI location.

The images were captured using a built-in dual camera with 20 megapixels each, integrated into the Huawei Mate 20 Lite mobile device running the Android operating system. The essential properties of this camera are detailed in Table 4.1. Emphasis is placed on its dual nature, indicating the presence of two lenses, each with a resolution of 20 megapixels. The total image resolution is 24 megapixels, translating to an aspect ratio of 3840 X 5120 pixels in width and height, respectively.

Regarding the camera's sensitivity to light, an average equivalent to 500 is observed, denoting ISO (International Organization for Standardization) as the unit. A higher ISO value allows greater sensitivity to light, with 500 considered a moderate value, hence suitable for standard lighting conditions. Lastly, the average shutter speed is 1/25s.

Table 4.1: Camera settings.

Camera	Resolution	ISO Speed	Shutter Speed
Dual 20MP, Camera mobile Huawei Mate 20 lite	3840 X 5120 Pixels [4:3] 24MP	500	1/25s

Photographic shots were taken at various distances, aiming to simulate the environmental observation perspective of the navigation robot, positioned at an approximate height of 1 meter above the ground. In the simulated hospital environment, photos were captured at a distance of 1 meter from the wall facing the captured door. Additionally, other spaces such as hallways were captured, varying the distances and lighting tones. The time of day when shots were taken was considered, with some in the morning and others in the evening, aiming to capture varied lighting conditions, including situations of excessive light, low luminosity, and standard lighting levels. This approach facilitated the acquisition of a set of representative and diversified images for subsequent analysis.

Various aspects were taken into consideration concerning door frames, with a particular focus on the variability of textures and colors. Glass doors were included, as well as those with frames that did not present noticeable differences in texture compared to the wall and the door figure 4.1. All these conditions were chosen based on previous research, aiming to enhance and provide rigor to the detection model (of Electrical et al., 2020). approach was adopted with the intention of ensuring a diverse and faithful representation of environmental conditions.

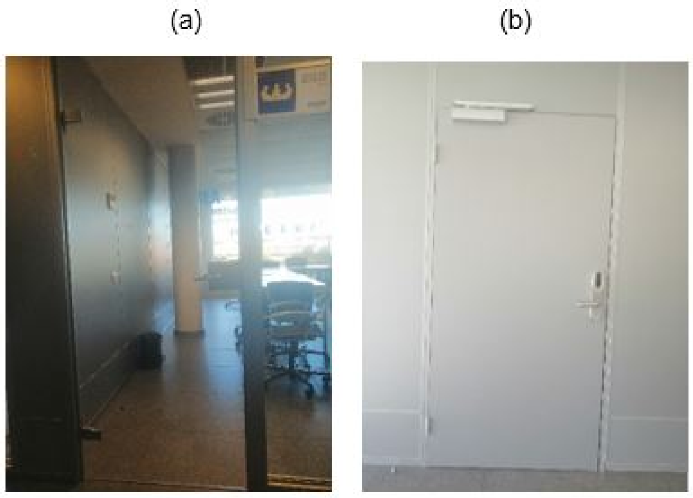


Figure 4.2: (a) Glass door with high contrast. (b) Door and wall share the same color, making it challenging to distinguish the door frame.

Regarding door handles, primarily images featuring a horizontally elongated type of handle were gathered in relation to the floor figure 4.2. This choice is grounded in the observation that this particular handle type is predominantly employed in hospital environments, which constitute the primary operational setting for the navigation robot. Nevertheless, images of other door handle types were also incorporated to prevent overfitting during the prediction phase.

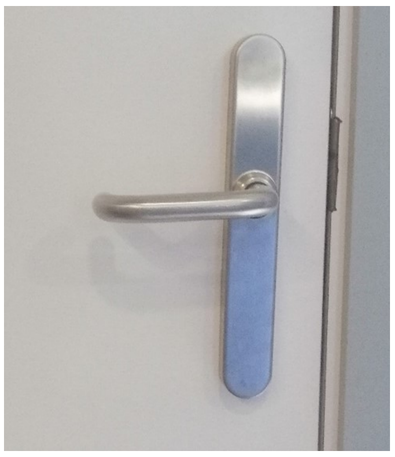


Figure 4.3: Door handle type considered in this study.

As part of this study, closed doors were systematically captured, aligning with the ultimate objective of enabling the robot to recognize and open doors. Notably, doors in open or partially closed states were excluded from consideration (Ramôa et al., 2021).This particular criterion was exclusively applied to the self-created database, as doors in varying states were considered in the dataset sourced from an open repository. To enhance the dataset, short videos were recorded within the simulated hospital corridor at UJI, thus augmenting the number of frames per second.

The free dataset was sourced from the image repository on the Roboflow platform, encompassing two distinct repositories. The first repository exclusively consists of door images (kursi kulkas, 2022), while the second repository exclusively contains images of door handles (doorhandler, 2023). These repositories were downloaded and subsequently subjected to preprocessing procedures to ensure uniformity and coherence in the dataset.

A total of 336 images were captured for the creation of the proprietary database, while 876 images were collected from the two free databases, resulting in a combined dataset of 1212 images. Within this dataset, 942 images were allocated to the "door" class, and 1094 images were assigned to the "handle" class. This meticulous class balance was addressed with particular attention to maintain an equitable distribution in the corresponding section of the study section 4.4.1. This approach ensures a fair representation of the categories of interest in the dataset.

For the Euclidean model, the input data comprises coordinates predicted by the previously trained YOLO NAS model. The model generates these coordinates in a 1x4 matrix, where the first two columns represent the (x, y) axes of the bottom-left corner, followed by the width and, finally, the length of the predicted bounding box figure 3.4. It is noteworthy that these coordinates are normalized within a range of 0 to 1. Subsequently, a transformation is applied to align them with the original dimensions of the image 640x40 section 3.3.2. This process ensures consistency between the model predictions and the actual dimensions of the image, providing an accurate Euclidean representation of detected objects in the scene.

4.2 Data Labeling

The labeling process emerges as a crucial phase, as it marks the commencement of manual teaching for object detection and their respective classes, culminating in the creation of bounding boxes with their corresponding coordinates. In the context of this project, the considered classes are "Door" and "Door handle." This procedure was executed through the utilization of the online computer vision platform, Roboflow (`https://roboflow.com/`).

In the initial phase, previously captured images are loaded, specifying the object detection process. Subsequently, the labeling process commences, where the image is presented alongside the tool for outlining the objects to be detected. Furthermore, within this procedure, class labels are assigned, with the "door" class labeled as 0 and the "door handle" class as 1 4.4.

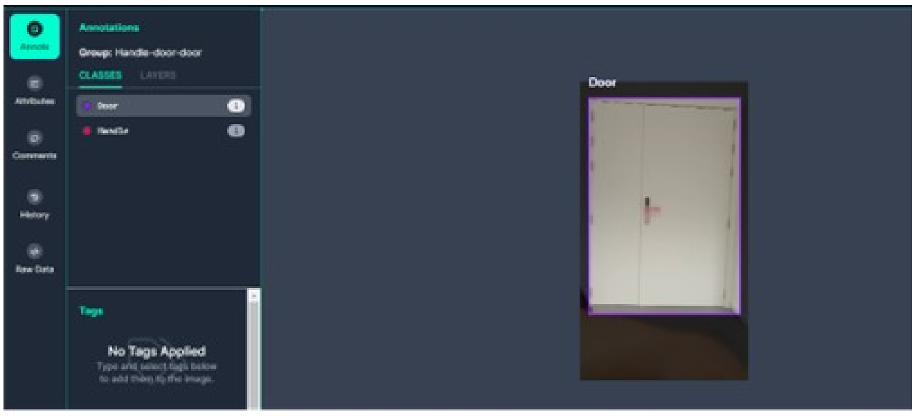


Figure 4.4: Labeling process in Roboflow.

4.3 Data preprocessing and Data augmentation

In the data preprocessing phase, the Auto Oriented tool was employed to standardize the image orientation, ensuring that objects were aligned in accordance with common human visual perception. This approach is grounded in validated conventions of visualization practices. Additionally, a resize operation was conducted on all images to conform to the dimensions accepted by the YOLO model, thereby establishing a standardized size of 640x640 in the dataset, a recognized standard in the literature to optimize the performance of detection models (Hao et al., 2022).

Data augmentation was applied with the purpose of enhancing diversity and enlarging the dataset, both critical elements in improving the predictive capacity of the model, figure 4.4. After implementing adjustments to the images, a final dataset comprising a total of 3,334 images was obtained. The processes applied to the dataset were devised considering challenging scenarios for detection, aiming to strengthen the model's robustness. These processes included:

- Crop: Introducing a 40% zoom to the original image.
- Shear: Adding variability to the image perspective, with horizontal tilts of 10 degrees and vertical tilts of 8 degrees, simulating a camera inclination.
- Blur: Incorporating a slight 2.5px blur effect to simulate less sharp image conditions, thereby contributing to the model's adaptation to realistic situations.

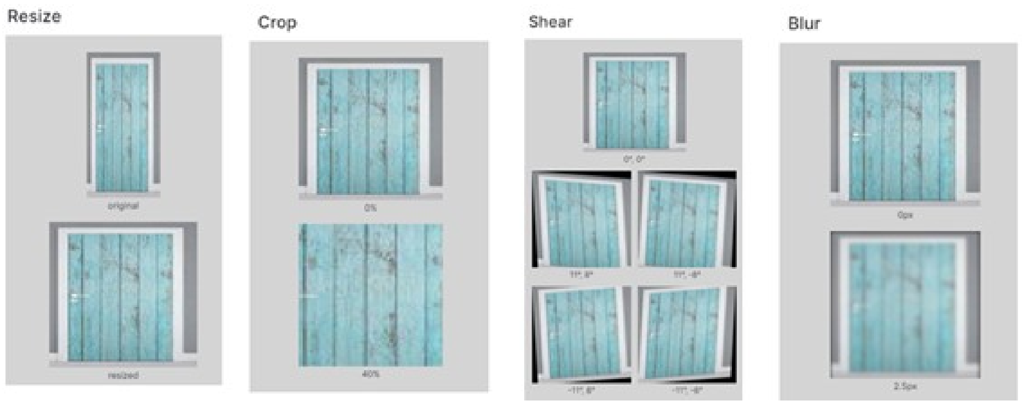


Figure 4.5: Preprocessing and Data augmentation.

4.4 Experimental design

4.4.1 Dataset Splitting

Within the scope of this project, two datasets were created. One comprised captured images and the integration of downloaded images from the free database, totaling 2722 images. The second dataset was exclusively formed with captured images, amounting to a total of 612 images. Particularly for this latter dataset, a strategy was employed to capture images under challenging conditions of varying light, distances, slight blurriness, and the introduction of diverse colors and textures on doors. This dataset was reserved for the model testing phase. Both datasets underwent identical preprocessing and data augmentation processes.

Table 4.2: Data splitting.

Dataset Name	Training	Validation	Test	Total
Dataset 1	80%	20%	0%	2722
N. Images	2177	545	0	
Dataset 2	0%	0%	100%	612
N. Images	0	0	612	
Total for both	70%	11.6%	18.4%	3.334
datasets	2333	386	612	

The data partitioning approach differed for each dataset. For the training dataset, 80% was allocated to the training phase, and 20% to the validation phase. In contrast, the second dataset was entirely 100% designated for the testing phase table 4.2. This choice was grounded in the precautionary measure of avoiding overfitting. Special attention was given to determining the size of the test set, ensuring it represented 20% of the total if both datasets were combined, thus preserving the percentage balance.

Class balance was meticulously considered during the labeling process prior to data augmentation. In the training set, 942 images were labeled for the "door" class and 1094 for the "handle door" class. In the test set, 150 images were labeled for the "Door" class and 162 for the "Door handle" class. It is crucial to emphasize that both datasets were treated independently in all project phases. Finally, for the Euclidean model, only the dataset created exclusively for testing was used as the data input.

Chapter 5

Results and Discussion

In this pivotal chapter, the outcomes of the study unfold across distinct sections. Beginning with section Section 5.1 the YOLO NAS Model Training, then the focus shifts to a section Section 5.2 detailed Comparison between Yolo NAS, Yolov8 and Yolo NAS only with one class: 'Handle', shedding light on their respective performances. Section Section 5.3 Euclidean Geometry Model provide insights into grip point localization. Moving forward section Section 5.4 A comparative of visual results between Euclidean Model's predictions and manual labeling enriches the findings. The subsequent section Section 5.5 The implementation of these outcomes in the Robot Brain sets the stage for section Section 5.6 Discussion on the achievements and implications of the study.

5.1 Yolo NAS Model Training

The YOLO NAS architecture was chosen for training, validation, and testing in Google Colab. The notebook was configured to utilize the Graphics Processing Unit (GPU) provided by Google, with GPU runtime being monitored using nvidia-smi. GPU was preferred due to its faster graphics processing capabilities. The supergradient library was installed to work with the YOLO NAS version. Key hyperparameters, including batch size, model size, and number of epochs, were set at 16, 'L', and 30, respectively. Table 5.1 YOLO NAS offers three model sizes: small (s), medium (m), and large (l). After experimenting with 'm' and 'l' sizes and comparing metrics, 'l' model size was chosen for this project. Regarding the number of epochs, experiments were conducted with 50, 70, and 100 epochs. Metrics did not significantly improve beyond epoch 50, and the best metrics were observed before reaching epoch 30. The results for 70 and 100 epochs could not be evaluated as the GPU disconnected due to processing time limitations. Additionally, the 'Adam' optimizer was applied, commonly used in object detection models for its computational efficiency and low memory usage (Thesis et al., n.d.).

Table 5.1: Key Parameters Yolo NAS model

Batch Size	Model Size	Number of epochs	Optimizer
16	L	30	Adam

In this project, metrics were evaluated using the Intersection over Union (IoU) threshold, a crucial measure to assess detection quality by quantifying the overlap between model predictions and ground truth. A threshold of 0.50 was chosen, signifying a successful detection if at least 50% of the predicted area overlaps with the actual object area. This value strikes a balance, as a higher threshold might be too strict, excluding accurate detections with slight overlaps, while a lower threshold could accept detections with minimal matches, compromising model accuracy. The decision to set an IoU threshold of 0.50 is grounded in the need to ensure adequately precise detections, considering both proper object coverage and the model's ability to generalize across diverse situations. This choice aligns with a common practice in evaluating object detection models, providing a balance between model sensitivity and specificity (Thesis et al., n.d.).

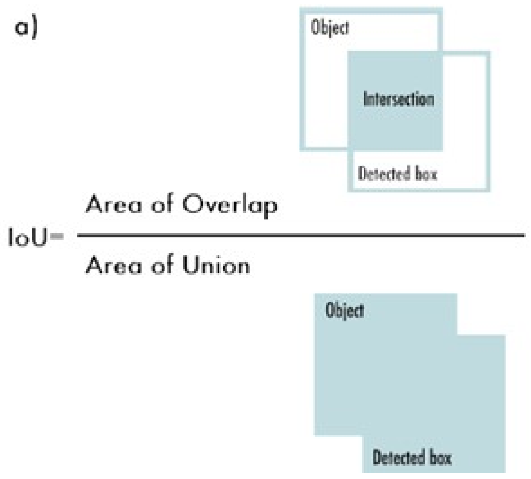


Figure 5.1: Intersection over union (IoU). Calculation of IoU is equal to dividing the intersection of two boxes (real object localization and detected box) by area of union between of these boxes.

Now, the results obtained in the evaluation metrics after training for the Yolo NAS m and Yolo NAS l models are presented in Table 5.2. Both models were trained for 30 epochs, following the same characteristics described earlier to ensure a balanced outcome in terms of parameters.

Table 5.2: Results comparison between Yolo NAS m and Yolo NAS l.

Model	Epoch	Precision0.50	Recall0.50	Map0.50	F1 0.50
Yolo Nas m	30	0.057	0.98	0.77	0.10
Yolo Nas l	30	0.095	0.99	0.89	0.17

The choice of metric for evaluation is based on recall, as this study emphasizes the importance of successful cases. It is crucial for the robot to effectively detect doors and handles, giving them priority over false alarms. The underlying reason is that not detecting an existing handle is considered more critical than incorrectly identifying an object that is not a handle. It is worth noting that, while other metrics are considered in the study, the primary comparison is conducted through recall.

In this case, a mAP of 0.89 was achieved in the Yolo Nas L version, which was 15.5% better than the Yolo Nas M version with a mAP of 0.77. In summary, Yolo Nas L showed superior results in the evaluation metrics when compared to Yolo Nas M. It is worth mentioning that the overall precision in the Yolo Nas version yields relatively low results, as observed in comparison with other studies utilizing this architecture (Casas et al., 2023). Further exploration of the precision of this model will be conducted in Section **??**. Given these results, the decision was made to proceed with the Yolo Nas L model throughout the project.

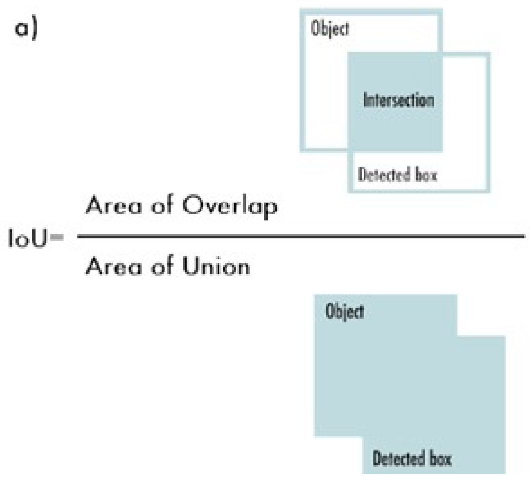


Figure 5.2: Confuxion Matrix - Yolo NAS l.

The confusion matrix generated for the YOLO NAS L model is depicted in Figure 5.2. It is evident that the model performs well in detecting successful cases. Although the percentages may appear low, this is associated with the low precision metric, as explained in Section **??**, where the implications of this low precision will be detailed. However, it is crucial to clarify that the model's overall performance is not subpar. The high mAP metric, which is the focal point and receives more attention in object detection models, validates this assertion, as illustrated in the model output Figure 5.3.

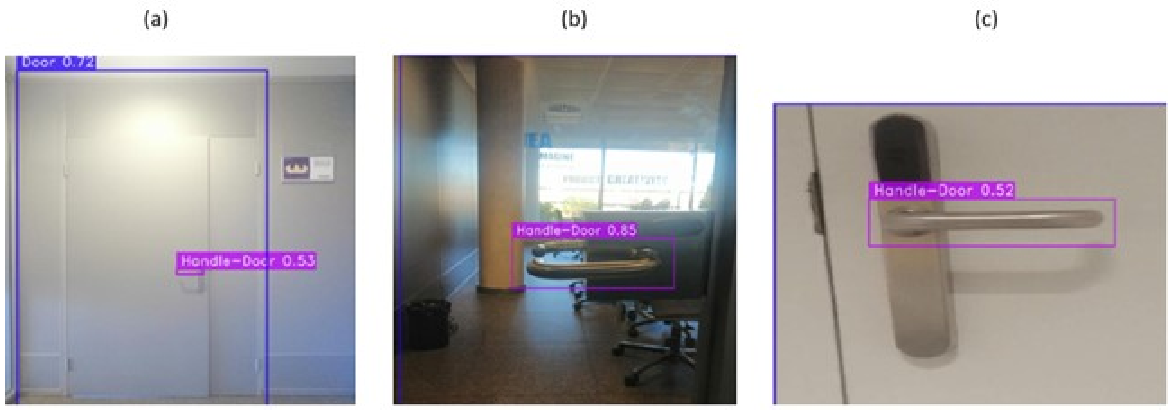


Figure 5.3: Output predictions Yolo NAS l model. (a) Wall and door with same color and texture. (b) Glass door. (c) Hospital door handle type.

Three different output scenarios are presented to showcase the model's versatility. In Figure 5.3 (a), a door with a similar texture and color to the wall is displayed, making it challenging to distinguish the door frame. Additionally, a beam of light in the upper part of the door adds complexity, and the door is at a considerable distance. Capturing the photo too close to the door does not cover the entire door frame. Although the model detects the door in this image, the bounding box is observed outside the door frame, but it accurately locates the door. As for the door handle, it is detected and localized precisely, with the bounding box correctly framing the handle door area. In Figure 5.3 (b), the most complex scenario captured is presented a glass door with beams of light generating high contrast in the image. Nevertheless, the model identifies the entire door frame, and the handle door is detected with high precision at 0.85%The bounding box is well-positioned, framing precisely the area where the handle is grasped and not the entire frame where the key is inserted. This precision is beneficial for accurately calculating the grip point. Finally, Figure 5.3(c) an example of the typical real door handle found in the UJI hospital is shown. Once again, a proper detection of the door handle is evident, identifying and localizing the region of interest (ROI) accurately.

5.2 Model Comparision

The decision was made to compare the quality of the Yolo Nas L model with two cases: a Yolo v8s model trained with the same dataset, with the size set to 's' as it exhibited good metrics, and a Yolo Nas M model trained exclusively with the door handle class and the size set to 'm'. The latter choice was based on the observation that in the initial training sessions, it showed a recall more similar to the desired performance. This decision aimed to achieve efficiency in execution times and versatility in comparisons. However, all models were trained for the same number of epochs. Table 5.3 presents the metrics used in this study (Precision, Recall, mAP, and F1) for each of the three cases.

Table 5.3: Results model comparison between Yolo NAS l, Yolov8s and Yolo NAS only 1 class: Handle.

Model	Epoch	Precision0.50	Recall	Map0.50	F1
Yolo Nas l	30	0.12	0.99	0.93	0.21
Yolo v8 s	30	0.82	0.95	0.94	0.88
Yolo Nas m Handle	30	0.04	0.99	0.91	0.08

The model with the highest precision is YOLOv8, significantly outperforming the other models. This implies that this model has an 82% accuracy in correctly predicting positive instances. However, the initial idea of improving precision by training the model with only one class was not achieved, suggesting that precision in YOLO NAS is not necessarily tied to the number of classes, at least in cases with a small number of classes. Both recall and mAP@0.50 for the three models are very close, with YOLO NAS L outperforming the others, indicating a strong ability to capture positive instances. Lastly, the best F1 score was achieved by the YOLOv8 model, highlighting its superior performance, as the balance between recall and precision is more equitable compared to the other models.

Figure 5.4 presents a comparison of the outputs from the models. The same set of images was used as outputs for both the YOLO NAS L and YOLOv8 M models. In images 5.4,5.5 (a) from both models, the door is not detected in the YOLOv8 version, whereas in the YOLO NAS L model, the door and door handle are detected with higher precision. For images 5.4, 5.5 (b), a door with the same color and texture as the wall is shown, testing the models' robustness. In this case, both the door and door handle are detected, although a slightly higher precision percentage is observed. Images 5.4, 5.5 (c) display a blurry image with the same texture and color as the door; however, in both cases, the door is successfully detected, while the door handle is not, considering the image's difficulty. Regarding the outputs of the YOLO NAS model trained solely for the door handle class, it can be observed that in image 5.6(a), the handle is detected and located successfully, whereas in image 5.6 (b), the handle is not detected.

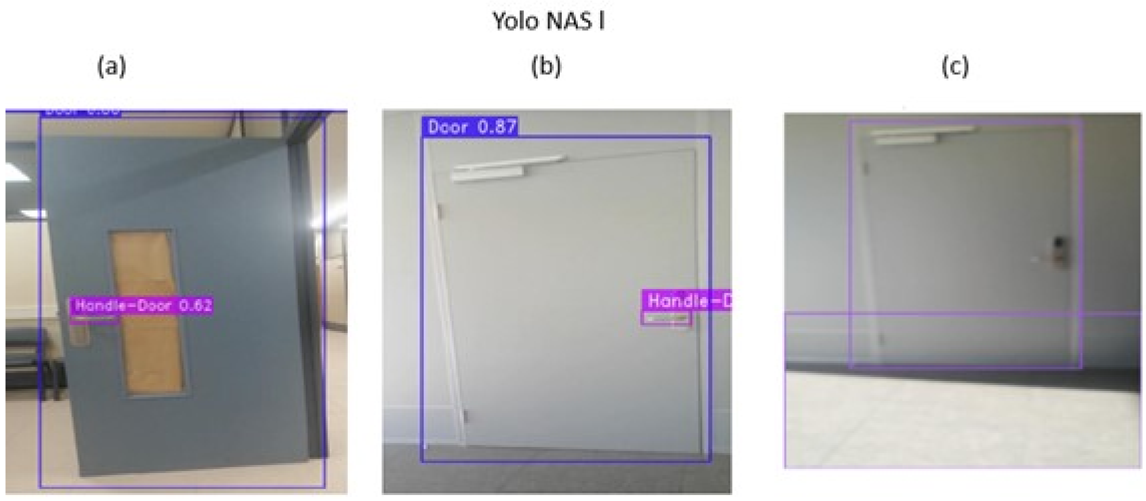


Figure 5.4: Yolo NAS l output.

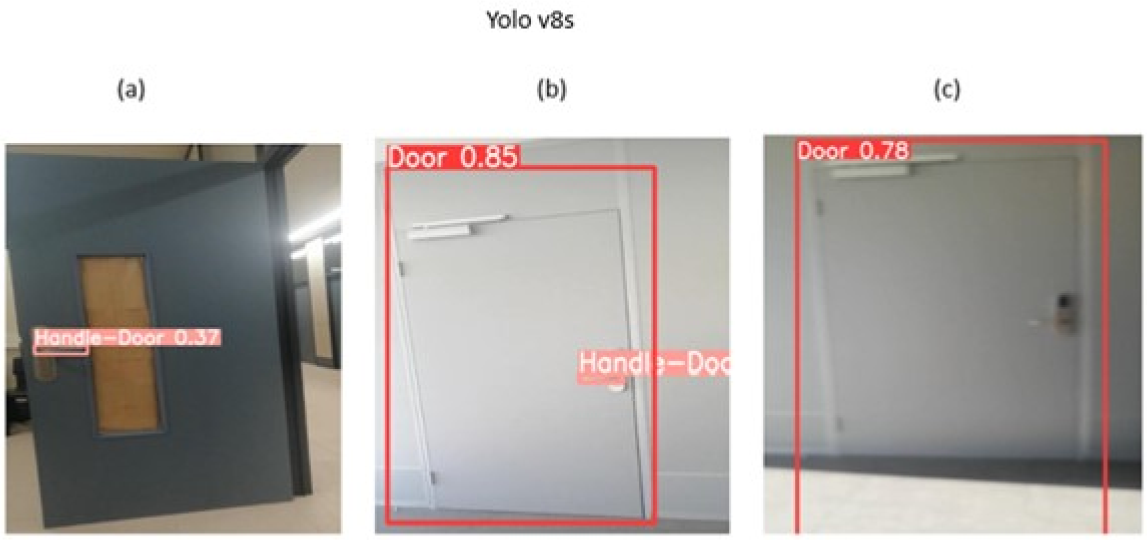


Figure 5.5: Yolo v8m Output.

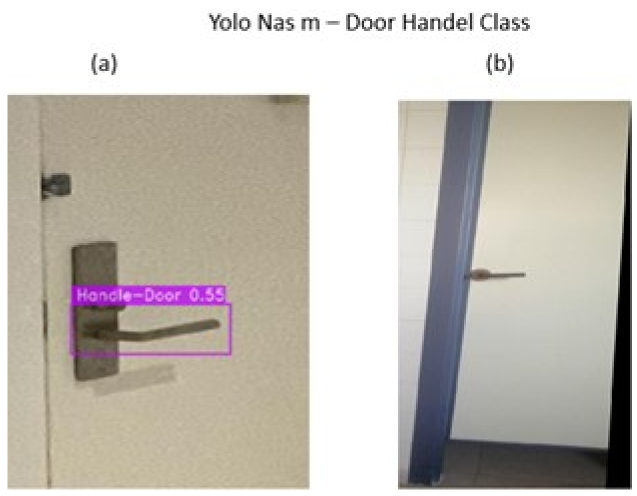
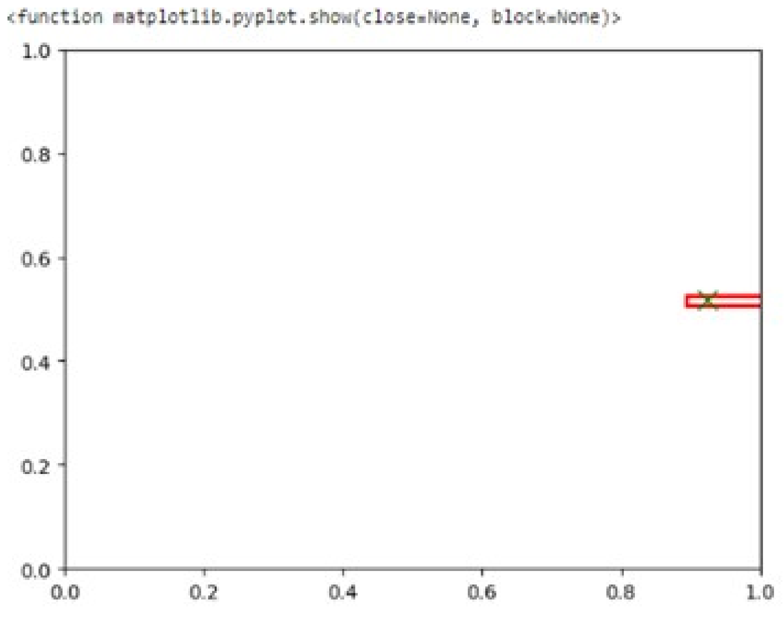


Figure 5.6: Yolo NASm Handle class.

5.3 Euclidean Geometry Model.

In this section, the results obtained from the Euclidean model will be showcased. Initially, the predicted coordinates of the bounding box for the door handle in a random image are presented Figure 5.7, then in Figure 5.8illustrating how they would appear without applying the transformation. This comparison aims to demonstrate the impact of the Euclidean model and provide insights into the effectiveness of the transformation in achieving accurate door handle localization.



```
<function matplotlib.pyplot.show(close=None, block=None)>
```

Figure 5.7: Reference frame of predicted coordinates.

As evident from the coordinates, they range from 0 to 1, signifying that they are in a different reference frame than the image displaying the prediction. Therefore, the transformation process Section 3.3.2 is necessary to align them with the image's reference frame (640x640).

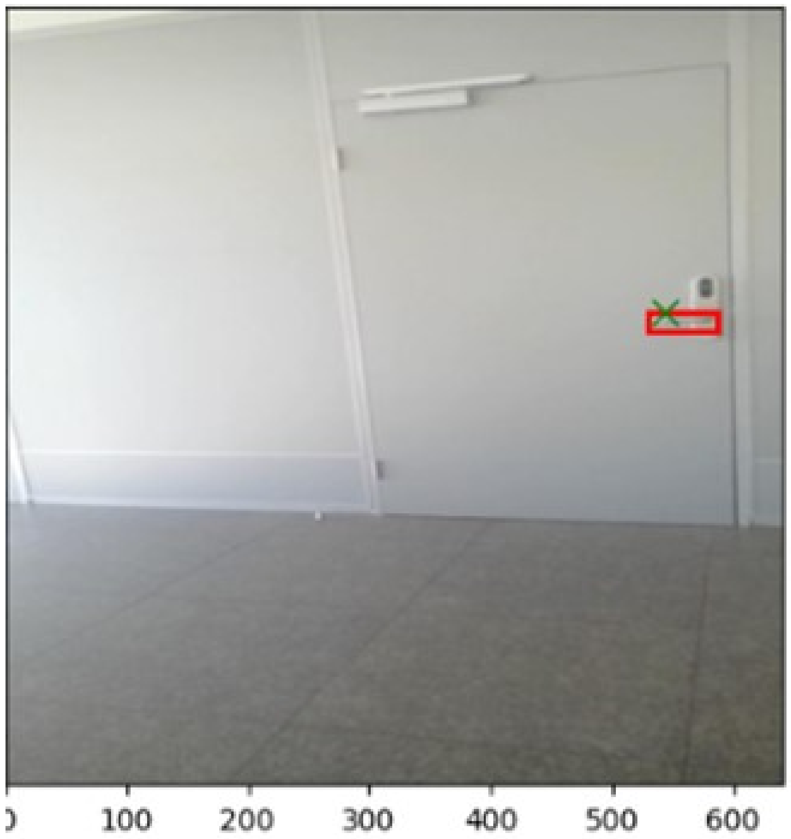
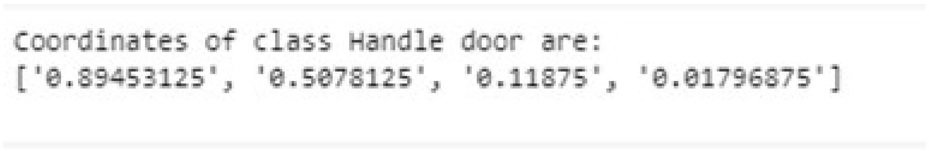


Figure 5.8: Grip point for door handle that opens to left size generated for the euclidean geometric model, with coordinates in same framework.

After transforming the coordinates, the formulas described in section 3.3.2 are applied for the handle door's grip point calculation. It is noteworthy that this process is exclusively carried out for the "handle door" class. In Figure 5.6, the final output of the Euclidean geometric model is displayed, demonstrating the consistency between the bounding box and the image, along with the accurate positioning of the grip point. This pertains to a left-opening handle door in this specific case.

5.4 Comparison Euclidean model output: Predicted and manual labeling

To validate the applicability of the Euclidean model and understand how its output should be, a similar process was conducted compared to the predicted model. This time, the coordinates of manually generated bounding boxes for the door handles were used. These coordinates were also normalized to a range of 0 to 1 figure 5.9, and the coordinate transformation process was applied. Subsequently, the Euclidean model was employed to predict the grip point.



```
Coordinates of class Handle door are:
['0.89453125', '0.5078125', '0.11875', '0.01796875']
```

Figure 5.9: Coordinates bounding box done manually.

Figure 5.10 (a) displays the image generated by the prediction, while Figure 5.10 (b) shows the image with the manually created bounding box. The close resemblance between the prediction and manually generated output suggests effective applicability for grip point generation.

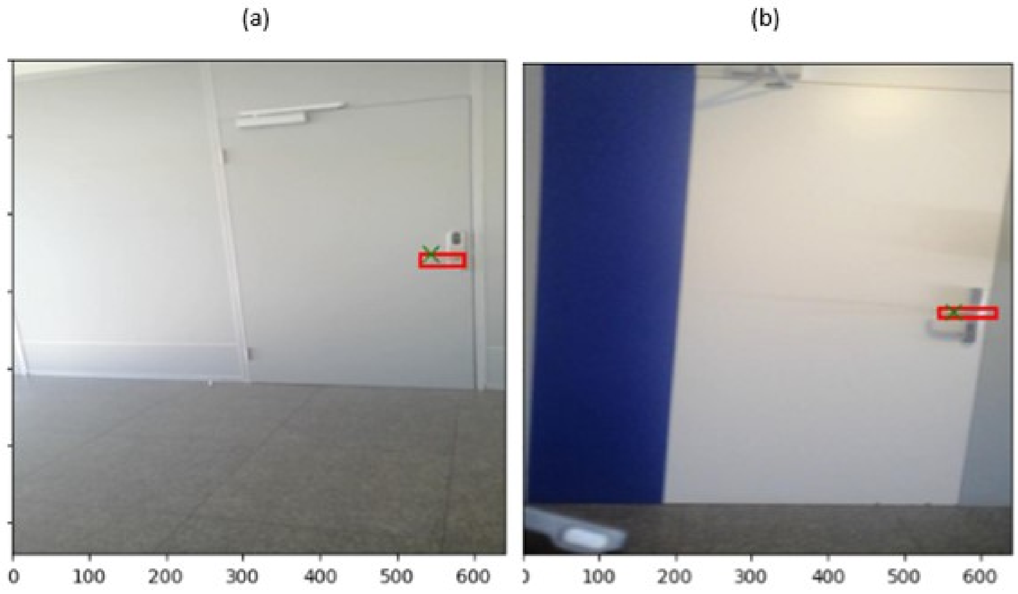
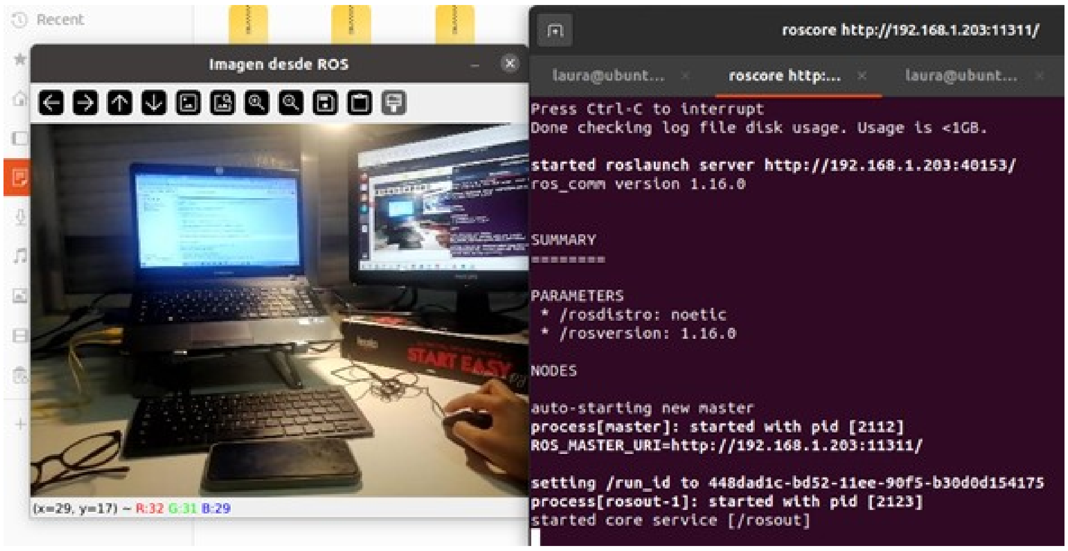


Figure 5.10: Output Euclidean model. (a) Image from predicted model. (b) Image from labeling manual.

5.5 Model implementation on Robot Brain simulation.

In this section, the initial step involves verifying the connection between real-time environmental data captured by the mobile device and the simulation of the mobile device's feed on the computer, which emulates the robot's system. To achieve this, the `sensor_msgs.msg` library is installed, compatible with Ubuntu 20.04, facilitating the establishment of the connection. Subsequently, the script is configured based on a previously developed script (rethink imcmahon, 2018), where the corresponding topic for the mobile camera is entered to establish the connection and enable the script to function. Input variables, such as the IP addresses of both the mobile device and the PC, connected to the same network, are introduced. Additionally, input variables for ROS are adjusted figure5.11.

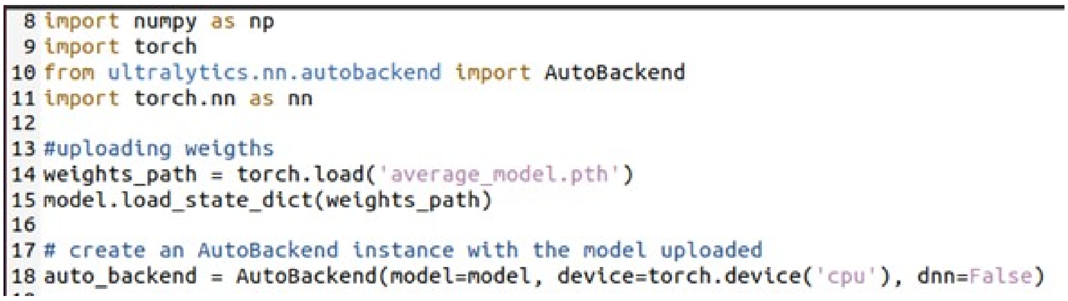


```
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.1.203:40153/
ros_comm version 1.16.0


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.16.0

NODES

auto-starting new master
process[master]: started with pid [2112]
ROS_MASTER_URI=http://192.168.1.203:11311/

setting /run_id to 448dad1c-bd52-11ee-90f5-b30d0d154175
process[rosout-1]: started with pid [2123]
started core service [/rosout]
```

Figure 5.11: Input information from Mobil read in robot view simulation.

In the next step, various errors related to the versions of super gradient libraries were encountered. To address this situation, the decision was made to install Ultralytics, a library that also allows importing YOLO NAS and running it on Ubuntu. While the installation issues were resolved, challenges arose related to the processor, as the YOLO NAS model was executed in a GPU environment in Google Colab, and in the Virtual Machine (VM), there is no access to GPU. This is because the scripts run locally (CPU), and they do not connect to Google Colab due to the incompatibility between Colab and version 1 of ROS.

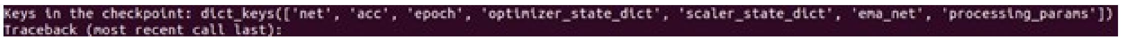
To overcome this challenge, the Autobackend library from Ultralytics was identified, facilitating the implementation of the model on various hardware platforms. This is crucial since some robots lack an integrated GPU and operate exclusively with a CPU. Autobackend optimizes the GPU-trained model and converts it into a format compatible with the hardware's CPU. In this project, Autobackend was invoked using Torch figure 5.12.



```python
import numpy as np
import torch
from ultralytics.nn.autobackend import AutoBackend
import torch.nn as nn

#uploading weigths
weights_path = torch.load('average_model.pth')
model.load_state_dict(weights_path)

# create an AutoBackend instance with the model uploaded
auto_backend = AutoBackend(model=model, device=torch.device('cpu'), dnn=False)
```

Figure 5.12: Importing and Invoking Autobackend library.

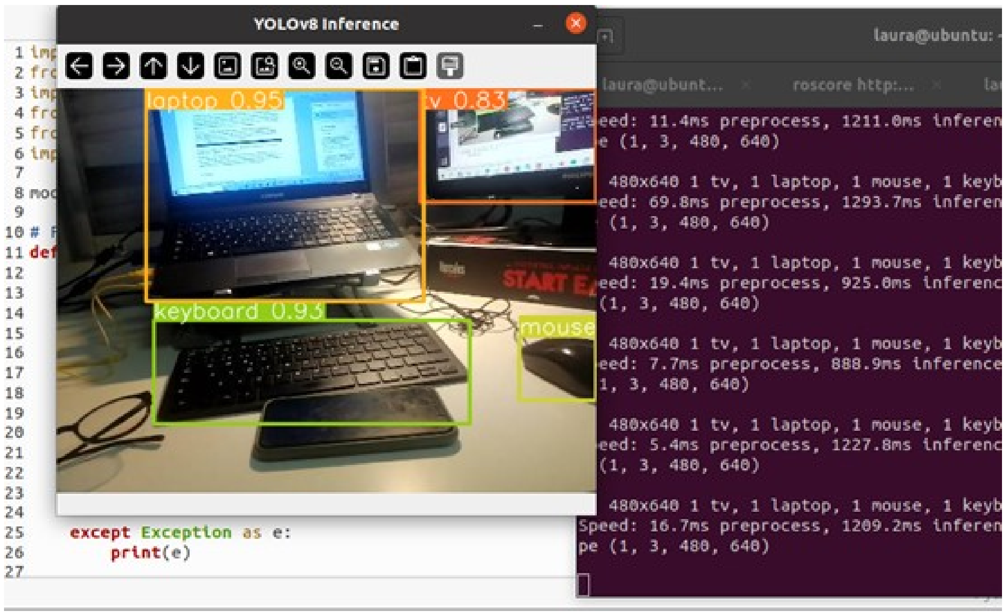
After this, the issue arose that ROS was requesting an encrypted key for the YOLO NAS model within the (.pth) file containing the model weights. Attempting to use the commonly generated key 'model' proved unsuccessful. Access to the model keys was obtained Figure 5.13 , and all of them were tested; however, it continued to generate an error of incorrect key.



```
Keys in the checkpoint: dict_keys(['net', 'acc', 'epoch', 'optimizer_state_dict', 'scaler_state_dict', 'ema_net', 'processing_params'])
Traceback (most recent call last):
```

Figure 5.13: Model keys.

Due to time constraints, a solution to this problem could not be found, and the YOLO NAS model could not be loaded to be tested in the robot simulator. Nevertheless, the decision was made to attempt loading the generated YOLOv8 model to verify the correct execution of the script and rule out potential library versioning errors or issues in how the model was being loaded. This model was successfully loaded, and real-time object detection was observed in the robot simulator, validating the script. However, there is suspicion of a conflict between the library import and the loaded model. Figure 5.14.



```
    except Exception as e:
        print(e)
```

```
eed: 11.4ms preprocess, 1211.0ms inferen
e (1, 3, 480, 640)

 480x640 1 tv, 1 laptop, 1 mouse, 1 keyb
eed: 69.8ms preprocess, 1293.7ms inferen
 (1, 3, 480, 640)

 480x640 1 tv, 1 laptop, 1 mouse, 1 keyb
eed: 19.4ms preprocess, 925.0ms inferenc
(1, 3, 480, 640)

 480x640 1 tv, 1 laptop, 1 mouse, 1 keyb
eed: 7.7ms preprocess, 888.9ms inference
1, 3, 480, 640)

 480x640 1 tv, 1 laptop, 1 mouse, 1 keyb
eed: 5.4ms preprocess, 1227.8ms inferenc
(1, 3, 480, 640)

 480x640 1 tv, 1 laptop, 1 mouse, 1 keyb
Speed: 16.7ms preprocess, 1209.2ms inferen
pe (1, 3, 480, 640)
```

Figure 5.14: Model yolov8 uploaded in robot simulation for verify the script.

It's important to note that, for this section, images from a different environment than doors were used. The sole purpose was to showcase the results obtained when testing the camera and script with the YOLOv8 model in the simulation representing a robot system.

5.6 Discussion

The training process involved setting up key parameters such as batch size, model size, and the number of epochs. The 'L' model size showed better performance compared to other sizes. The decision to limit epochs to 30 was based on the observation that optimal metrics were achieved before reaching 30 epochs. The

Adam optimizer, known for its computational efficiency, was also utilized. Choosing IoU as a metric with a threshold of 0.50 reflects a balanced approach, not losing information due to strict measures, given the project's flexibility. The YOLO NAS L version detected doors and handles 15% better than the YOLO NAS M version, as metrics were superior in the 'L' version. These outcomes influenced the decision to proceed with the 'L' size for the project.

Upon scrutinizing the metrics for the YOLO NAS L model, there is initially a high recall of 0.99, signifying the model's proficiency in detecting the majority of real cases of doors and handles. This is crucial in a scenario where the priority is to identify as many positive cases as possible, even at the risk of generating false alarms. As was explained in section 5.1, priority is given to the recall metric. This underscores the importance of specifying a precision range, as done with the mAP. On the flip side, a very low precision is observed, especially when compared to the trained YOLOv8m model. This low precision implies that, although the model is adept at detecting positive cases, it has a high rate of false positives. In simpler terms, it tends to detect doors and handles even when they are not actually present. This raises the question of how risky or problematic it is for the robot to detect a non-existent door. If this risk is deemed insignificant, the model could still be highly effective.

The low F1 score of 0.17 indicates an imbalance between precision and recall, with the latter previously identified as very low. Regarding the mAP@0.50, reflecting the model's balance, the value of 0.89 suggests effective object detection for both evaluated classes: doors and handles. Specifically, the mAP indicates a substantial overlap between the real and predicted areas of the objects, revealing a high match between the predicted bounding boxes and those manually created during the training phase.

The outputs provided additional information, highlighting the model's versatility in various scenarios. Remarkably, the model successfully tackled challenges such as doors with textures similar to walls or glass doors with high contrast, as evidenced by correctly positioned bounding boxes.

After obtaining visual results from the models, an interesting observation emerged. Despite YOLOv8 yielding better metrics, there were instances where it failed to detect objects or did so with low precision, unlike the YOLO NAS model. This observation was compared with another study on smoke and fire detection, employing various YOLO versions, including YOLOv8 and YOLO NAS. A similar pattern was identified: despite YOLOv8's favorable metrics, particularly in precision, visual examination revealed instances where it failed to detect smoke or fire (Casas et al., 2023). Nevertheless, it is acknowledged that YOLOv8 stands out as an architecture for detection, particularly for its efficiency in identifying negative cases. This aspect is crucial depending on the project's objective; for instance, in fire studies, it is essential for the model not to generate false positives.

On the other hand, the model that was train exclusively with the "handle door" class did not showcase satisfactory results. This led to the conclusion that, in scenarios with a limited number of classes, improvements in precision do not make a significant difference.

It is noteworthy that, despite the low precision recorded in the evaluation metrics of YOLO NAS L, upon inspecting the model output, it becomes evident that the model effectively fulfills its purpose. This holds particular importance in the field of robotics, where emphasis is placed on successful cases. However, it remains crucial to analyze scenarios where the failure to recognize an object could have serious consequences. Therefore, implementing additional safety redundancy measures in the robot is essential. These measures involve the robot's system executing extra verification mechanisms or correcting minor errors in situations where the consequences do not pose a significant threat. It is important to note that robots maintain a margin of error (Verne, n.d.).

In the case of the model designed to detect the grip point based on Euclidean geometry, it is advantageous that the YOLO NAS L model demonstrates high accuracy in the Region of Interest (ROI) defined by the mAP. This serves as a robust input for the model. The grip point is determined within the image's frame of reference, ensuring precision. Additionally, through visual inspection and comparison with manually generated output, it is evident that the grip point is accurately located.

In this study, the achieved high precision in identifying the grasping point is highlighted, as it consistently locates within the Region of Interest (ROI) in the correct orientation. This precision makes it highly suitable for implementation in the robot, where the probability of error is minimal. The emphasis is placed on this high precision in comparison with another study that trained a door handles detection model and manipulated the grasping point using the MoveIt library, which calculates the centroid of the ROI. In that study, point clouds generated by Kinect were used as input. However, this earlier study faced challenges in detecting the handle at distances greater than 1.5 meters from the door, challenges that have been overcome in this research.

Furthermore, the use of a proprietary model based on Euclidean geometry for precise grasping point identification is underscored. Unlike the strategy employed by the mentioned study that imports the MoveIt library and locates the grasping point at the centroid, this study utilized basic mechanics concepts such as the lever theory and moment to calculate the grasping point so that it is positioned near the end of the handle. This choice significantly minimizes the force required by the robot during grasping. In this way, an additional benefit is provided, contributing to the optimization of the robot's force (of Electrical & Engineers, n.d.).

Despite conflicts between library versions, the camera was successfully enabled in the virtual environment simulating the robot system. This marks a significant milestone in obtaining the necessary input. Additionally, the proper execution of a YOLOv8 version to detect the received input was confirmed, despite encountered limitations when loading the YOLO NAS model. It is crucial to emphasize the importance of using the Autobackend library, enabling the execution of GPU-trained models on hardware equipped with CPU. This underscores the system's adaptability to diverse hardware platforms, enhancing the model's versatility and efficiency in various environments.

Chapter 6

Conclusions

In this study, YOLO NAS models were trained with sizes 'm' and 'l', along with a single class, and the YOLOv8 model, for detecting and locating doors and handles. Subsequently, a model based on Euclidean geometry was developed to identify the gripping point on the handles. A comprehensive evaluation of the project was conducted by applying the two described models in the simulation of a navigation robot system. The goal was to assess the effectiveness of the recent YOLO NAS architecture in addressing challenges associated with early detection of doors and handles. Additionally, predictions from object localization were utilized to create an efficient model for detecting the gripping point on door handles. The intention was to integrate these models into the simulation of a navigation robot, highlighting potential limitations in the process and paving the way for future contributions. All of this was undertaken with the primary motivation of integrating this model into a navigation robot capable of opening doors in indoor environments, particularly in hospitals, which served as the foundational study location for this project.

To achieve this, a dataset was employed, specially curated for door and handle detection in indoor environments, created within the scope of this study. Additionally, a freely available dataset from Roboflow, comprising a total of 3334 images, was utilized. This database serves as a valuable resource for training other models and proves to be an excellent tool for detecting doors and handles in subsequent research.

The evaluation of YOLO models employed various metrics, including Recall, mean average precision (mAP@0.50), F1 score, and precision. These metrics facilitated an effective comparison of the performance of the trained models, specifically YOLO NAS m and YOLO NAS l. The results revealed that the superior model was YOLO NAS l after 30 epochs, surpassing YOLO NAS m by 15%Ṫhis is evident in the mAP of YOLO NAS l, which reached 0.89, compared to 0.77 for YOLO NAS m.

Subsequently, the best-selected model was compared with the proposed performances: YOLO NAS m with a single class "handles" and the version preceding NAS, YOLOv8. The results indicated that having a single class did not improve precision, at least in training with a limited number of classes. Furthermore, YOLOv8 exhibited significantly higher precision compared to YOLO NAS. However, concerning Recall, the key metric considered in this study due to the importance assigned to true positive cases, as detailed in section **??**, prioritizing the accurate detection of doors and handles by the robot yielded excellent results, with a recall of 0.99. This suggests that YOLO NAS is effective in detecting positive cases, albeit with a high rate of false positives, identifying doors and handles where there are none.

The mAP of the NAS model achieved effective detection of the overlap between the real regions of objects and the predicted regions. This is particularly beneficial for the input of the Euclidean model, providing precision when calculating the grip point.

It is crucial to consider how the model evaluates its metrics based on the project's objectives. For instance, in studies related to fires, it is essential that the model minimizes false positives. This underscores the importance of tailoring the model evaluation to the specific goals of each project.

The visual results of YOLO NAS l exhibited outstanding performance by successfully detecting doors and handles under challenging conditions, including variations in lighting, distance, texture, and door color, such as glass doors. Even when exposed to images beyond the training dataset, this model provided superior visual outputs compared to YOLOv8, which showed less precision in detecting handles and doors. This analysis assessed the visual robustness of the model, concluding that YOLO NAS l effectively detects the proposed object, a significant aspect in the field of robotics.

By implementing formulas based on Euclidean geometry, the coordinates of the gripping point were calculated with high precision. As a result, a signaling indicating where the robot should anchor itself was generated, enhancing decision-making when opening the door. The simplicity of this model is emphasized, as simple calculations are expected to be reflected in optimizing the robot's performance in terms of hardware.

The initial limitations encountered during model implementation included conflicts in library versioning, particularly with ROS1. This issue led to the conclusion that thorough validation of the entire implementation environment is necessary before selecting an architecture. It cannot be automatically assumed that the latest version is the best choice for a project overall. Another challenge arose when implementing a YOLO NAS model trained with GPU in a CPU environment. This prompted the search for and use of the autobackend library, which configures the environment to process a GPU-trained model in a CPU environment. Despite efforts, the implementation of the YOLO NAS model was unsuccessful as the execution environment required an encrypted key generated by the model, which could not be obtained due to time constraints. However, a YOLOv8 detection model was successfully executed in the simulator through the implementation of the Ultralytics library.

6.1 Limitations

During the training of the YOLO NAS models, it was necessary to run them on a GPU when training from Google Colab. While adjusting the number of epochs to 70 or 100 to find the optimal epoch for the model, the GPU would disconnect due to execution time limits managed by 'nvidia-smi.'

The environment emulating the robot was configured with the Ubuntu 20.04 operating system and the collection of ROS libraries and tools. When working with YOLO NAS in an environment with this configuration, an encrypted key was requested as a security measure. This key was used to encrypt the model, making access challenging as a valid key could not be found to execute the model.

6.2 Future Work

As the project progressed, a crucial consideration arose in calculating the gripping point: the direction in which the door opens, either to the left or to the right. This variable directly influences the model's input, as it determines the formula to be applied for accurately locating the gripping point. To address this temporarily, a prompt was implemented, asking the user (who controls or monitors the robot) to indicate the door's opening direction. However, to achieve autonomous operation, the option of labeling images in two classes during the project's initial phase was proposed: "handle to the left" and "handle to the right."

Additionally, there is a need to set up an environment that allows for an extended connection to the GPU during the model training phase. Finally, the goal is to properly implement the model into the robot system, overcoming the limitations described in the project, by integrating it into hardware with an embedded GPU, such as the Jetson Nano.

Appendix A

Data Repository created

A.1 Door and Handle door detection

Door Handle Detection dataset is a collection of images sourced from the UJI environment, supplemented with additional images obtained from previously cited open datasets focusing on door and handles. This dataset encompasses a comprehensive range of scenarios and conditions, ensuring its applicability across diverse settings. Notably, the dataset has undergone data augmentation techniques, enhancing its robustness and enriching its utility for training and validation purposes. The "Door Handle Detection" dataset stands as a valuable resource for advancing research in the field of computer vision, particularly in the domain of door and handle detection algorithms.

The `Door and Handle door detection` dataset can be used and downloaded from `https://universe.roboflow.com/laura-munoz/door-handle-detection`.

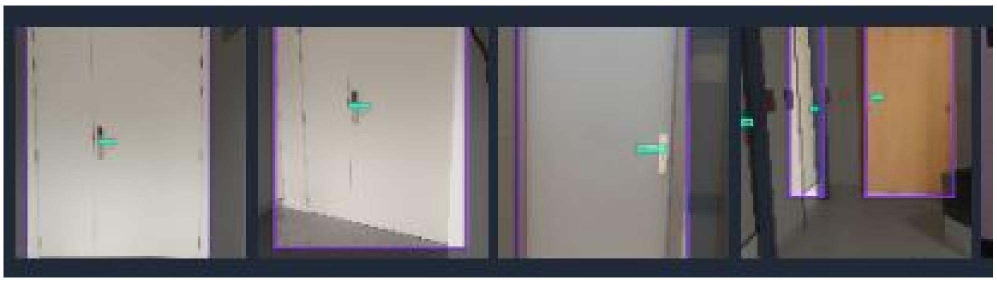


Figure A.1: Door and handle door dataset.

A.2 Door and Handle Detection for testing

The Test Door Handle Detection dataset is of particular interest due to its dedicated purpose as input data for model validation. The dataset was created to present challenging scenarios for the trained model. Attention was given to select doors that posed significant challenges, considering factors such as texture variation, inclusion of glass doors, diverse lighting contrasts, and deliberate matching of door color with surrounding walls to obscure the door frame or contour. By incorporating these complexities, the dataset effectively evaluates the robustness of the model, thereby avoiding overlap between the training and test datasets. This dataset does not have data augmentation.

The `Test door handle detection` dataset can be used and downloaded from `https://universe.roboflow.com/laura-munoz/test_door_handle_detection`.

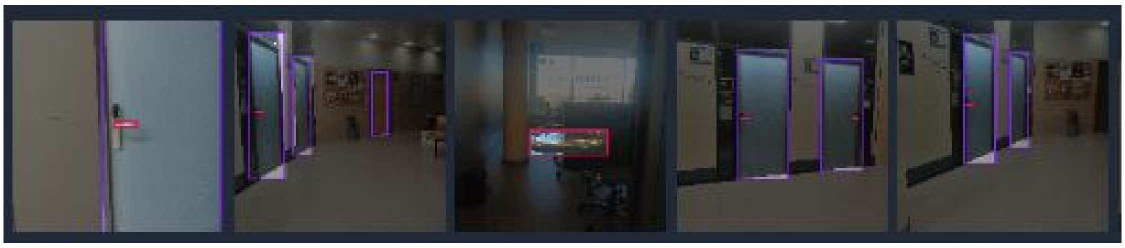


Figure A.2: Test door and handle dataset.

A.3 Just Handle door detection

The Just Handle door dataset comprises solely door handle images meticulously selected from collection and complemented with freely available data. Employing data augmentation techniques further enhanced the dataset's diversity and utility. This dataset was specifically utilized to train a model dedicated exclusively to door handle detection, utilizing the YOLO NAS architecture.

The `Just handle door detection` dataset can be used and downloaded from `https://universe.roboflow.com/laura-munoz/just_handle_door`.

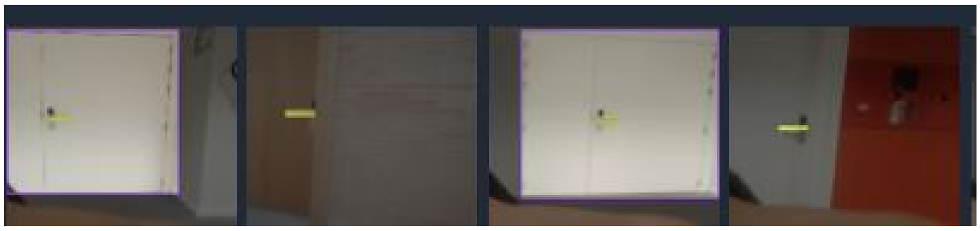


Figure A.3: Test door and handle dataset.

Appendix B

Code

B.1 Yolo Nas L model

All the .py code used in this thesis is bundled in the Handle door detection and grip point localization repository. The entire code is annotated with comments that explain the flow of the code, detailing what is done at each step, can be found on GitHub, through the following link: https://github.com/LauraMunozAmaya/Handle-door-detection-and-grip-point-localization.git. In this repository, you will find the script used to train the YOLO-NAS-L model for door and handle detection, its possible also see the outputs of the script.

```
#Requirements
#It is desired to ensure that the GPU Accelerator is being used in this notebook, in order to have significally
     speed up model training times. nvidia-smi command will be used for do that.
!nvidia-smi

%%capture
!pip install -q git+https://github.com/Deci-AI/super-gradients.git@stable
!pip install -q super-gradients==3.2.0
!pip install -q roboflow
!pip install -q supervision


#Dataset Just for Training & Valid (versin 10 Roboflow)
from roboflow import Roboflow
rf = Roboflow(api_key="XXXXXX")
project = rf.workspace("laura-munoz").project("door-handle-detection")
dataset = project.version(10).download("yolov5")


#Dataset Just for test

rf_t = Roboflow(api_key="XXXXXXX")
project_t = rf_t.workspace("laura-munoz").project("test_door_handle_detection")
dataset_t = project_t.version(2).download("yolov5")

#Class Definition
from typing import List, Dict
class config:
   # Project paths
   DATA_DIR: str = "/content/Door-handle-detection-10"
   CHECKPOINT_DIR: str = "/content/checkpoints"
   EXPERIMENT_NAME: str = "tesis_detection_localization_model_yoloNAS_L"

   # Datasets
   TRAIN_IMAGES_DIR: str = "/content/Door-handle-detection-10/train/images"
   TRAIN_LABELS_DIR: str = "/content/Door-handle-detection-10/train/labels"
   VAL_IMAGES_DIR: str = "/content/Door-handle-detection-10/valid/images"
   VAL_LABELS_DIR: str = "/content/Door-handle-detection-10/valid/labels"
   TEST_IMAGES_DIR: str = "/content/Test_Door_Handle_Detection-2/test/images"
   TEST_LABELS_DIR: str = "/content/Test_Door_Handle_Detection-2/test/labels"

   # Classes
   CLASSES: List[str] = ['Door','Handle-Door']
   NUM_CLASSES: int = len(CLASSES)

   # Model
   DATALOADER_PARAMS: Dict = {
     'batch_size': 16,
     'num_workers': 1
   }
   MODEL_NAME: str = 'yolo_nas_l'
   PRETRAINED_WEIGHTS: str = 'coco'

#Datalouders Initialization
from super_gradients.training import Trainer
from super_gradients.training.dataloaders.dataloaders import coco_detection_yolo_format_train
from super_gradients.training.dataloaders.dataloaders import coco_detection_yolo_format_val

train_data = coco_detection_yolo_format_train(
   dataset_params={
      'data_dir': config.DATA_DIR,
      'images_dir': config.TRAIN_IMAGES_DIR,
      'labels_dir': config.TRAIN_LABELS_DIR,
      'classes': config.CLASSES
   },
   dataloader_params=config.DATALOADER_PARAMS
)

test_data = coco_detection_yolo_format_val(
   dataset_params={
      'data_dir': config.DATA_DIR,
      'images_dir': config.TEST_IMAGES_DIR,
      'labels_dir': config.TEST_LABELS_DIR,
      'classes': config.CLASSES
   },
   dataloader_params=config.DATALOADER_PARAMS
)

val_data = coco_detection_yolo_format_val(
   dataset_params={
      'data_dir': config.DATA_DIR,
      'images_dir': config.VAL_IMAGES_DIR,
      'labels_dir': config.VAL_LABELS_DIR,
      'classes': config.CLASSES
   },
   dataloader_params=config.DATALOADER_PARAMS
)


train_data.dataset.transforms

#Training hyperparameters

from super_gradients.training.losses import PPYoloELoss
from super_gradients.training.metrics import DetectionMetrics_050
from super_gradients.training.models.detection_models.pp_yolo_e import PPYoloEPostPredictionCallback

train_params = {
   "average_best_models":True,
   "warmup_mode": "linear_epoch_step",
   "warmup_initial_lr": 1e-6,
   "lr_warmup_epochs": 3,
   "initial_lr": 5e-4,
   "lr_mode": "cosine",
   "cosine_final_lr_ratio": 0.1,
   "optimizer": "Adam",
   "optimizer_params": {"weight_decay": 0.001},
   "zero_weight_decay_on_bias_and_bn": True,
   "ema": True,
   "ema_params": {"decay": 0.9, "decay_type": "threshold"},
   "max_epochs": 30,
   "mixed_precision": True,
   "loss": PPYoloELoss(
      use_static_assigner=False,
      num_classes=config.NUM_CLASSES,
      reg_max=16
   ),
   "valid_metrics_list": [
      DetectionMetrics_050(
         score_thres=0.1,
         top_k_predictions=300,
         num_cls=config.NUM_CLASSES,
         normalize_targets=True,
         post_prediction_callback=PPYoloEPostPredictionCallback(
            score_threshold=0.01,
            nms_top_k=1000,
            max_predictions=300,
            nms_threshold=0.7
         )
      )
   ],
   "metric_to_watch": 'mAP@0.50'
}

#Trainig Model
# Model Download. Yolo-NAS_L Model is downloaded
from super_gradients.training import models
model = models.get(config.MODEL_NAME, #YOLO_NAS_L
              num_classes=config.NUM_CLASSES,
              pretrained_weights=config.PRETRAINED_WEIGHTS) #PRETRAINED_WEIGHTS = "coco"
```

```python
#Train
trainer.train(model=model,
          training_params=train_params,
          train_loader=train_data,
          valid_loader=val_data)

#Loading the best model
import os
best_model = models.get(config.MODEL_NAME,
                  num_classes=config.NUM_CLASSES,
                  checkpoint_path=os.path.join(config.CHECKPOINT_DIR, config.EXPERIMENT_NAME, '/content/
                       checkpoints/t'))

#Evaluating the best model
trainer.test(model=best_model,
         test_loader=test_data,
         test_metrics_list=DetectionMetrics_050(score_thres=0.1,
                                      top_k_predictions=300,
                                      num_cls=config.NUM_CLASSES,
                                      normalize_targets=True,
                                      post_prediction_callback=PPYoloEPostPredictionCallback(score_threshold=0.01,
                                                                              nms_top_k=1000,
                                                                              max_predictions=300,
                                                                              nms_threshold=0.7)
                                     ))

#Visualization

#Selecting a random image from path image
import cv2
import numpy as np
import os
import random

#path to the directory containing images
path_img = ("/content/Test_Door_Handle_Detection-2/test/images/")

#List all images in the directory
file_img = os.listdir(path_img)

#Selecting a random subset of images from the list (file_imgs)
selected_imgs = random.sample(file_img, k=10)

#initialize a list to store the loaded images
images = []

#Loop through the selected image file names, read and convert in RGB color
for img_file in selected_imgs:
  img_path = os.path.join(path_img, img_file) #recreating path file complete name
  img = cv2.imread(img_path) #Uploading images in cv2 format
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #converting in RGB color

  if img is not None:
    images.append(img) #Adding image selected in "images" list
  else:
    print(f"Failed to load image: {img_file}")

print(outputs)
outputs.show()

#Confusion Matrix
!export LC_ALL=C.UTF-8
!export LANG=C.UTF-8
!pip install onemetric

import os

import numpy as np

from onemetric.cv.object_detection import ConfusionMatrix

keys = list(ds.images.keys())

annotation_batches, prediction_batches = [], []

for key in keys:
   annotation=ds.annotations[key]
   annotation_batch = np.column_stack((
      annotation.xyxy,
      annotation.class_id
   ))
   annotation_batches.append(annotation_batch)

   prediction=predictions[key]
   prediction_batch = np.column_stack((
      prediction.xyxy,
      prediction.class_id,
      prediction.confidence
   ))
   prediction_batches.append(prediction_batch)

confusion_matrix = ConfusionMatrix.from_detections(
   true_batches=annotation_batches,
   detection_batches=prediction_batches,
   num_classes=len(ds.classes),
   conf_threshold=CONFIDENCE_TRESHOLD
)

confusion_matrix.plot(os.path.join(HOME, "confusion_matrix.png"), class_names=ds.classes)
```

```
#Inference results
import supervision as sv

CONFIDENCE_TRESHOLD = 0.5

predictions = {}

for image_name, image in ds.images.items():
   result = list(best_model.predict(image, conf=CONFIDENCE_TRESHOLD))[0]
   detections = sv.Detections(
      xyxy=result.prediction.bboxes_xyxy,
      confidence=result.prediction.confidence,
      class_id=result.prediction.labels.astype(int)
   )
   predictions[image_name] = detections
import random
random.seed(10)


import supervision as sv

MAX_IMAGE_COUNT = 5

n = min(MAX_IMAGE_COUNT, len(ds.images))

keys = list(ds.images.keys())
keys = random.sample(keys, n)

box_annotator = sv.BoxAnnotator()

images = []
titles = []

for key in keys:
   frame_with_annotations = box_annotator.annotate(
      scene=ds.images[key].copy(),
      detections=ds.annotations[key],
      skip_label=True
   )
   images.append(frame_with_annotations)
   titles.append('annotations')
   frame_with_predictions = box_annotator.annotate(
      scene=ds.images[key].copy(),
      detections=predictions[key],
      skip_label=True
   )
   images.append(frame_with_predictions)
   titles.append('predictions')

%matplotlib inline
sv.plot_images_grid(images=images, titles=titles, grid_size=(n, 2), size=(2 * 4, n * 4))
```

B.2 Yolo v8

The python code used in this thesis for implementing the yolo v8 model can be found at https://github.com/LauraMunozAmaya/Handle-door-detection-and-grip-point-l blob/2ee33f63febf1d9a9a82c96ddba0e47c80a0a873/Yolov8.ipynb.

```
!nvidia-smi

# Pip install method (recommended)

!pip install ultralytics==8.0.20

from IPython import display
display.clear_output()
import ultralytics
ultralytics.checks()

!mkdir {HOME}/datasets
%cd {HOME}/datasets

!pip install roboflow --quiet
#Training & Validation Dataset
from roboflow import Roboflow
rf = Roboflow(api_key="XXXXXXX")
project = rf.workspace("laura-munoz").project("door-handle-detection")
dataset = project.version(10).download("yolov5")

#Test Dataset
!pip install roboflow

from roboflow import Roboflow
rf_t = Roboflow(api_key="XXXXXX")
project_t = rf_t.workspace("laura-munoz").project("test_door_handle_detection")
dataset_t = project_t.version(2).download("yolov5")


%cd {HOME}
```

```
!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=50 imgsz=800 plots=True

%cd {HOME}
Image(filename=f'{HOME}/runs/detect/train/confusion_matrix.png', width=600)


%cd {HOME}
Image(filename=f'{HOME}/runs/detect/train/results.png', width=1000)

#Outputs

%cd {HOME}
Image(filename=f'{HOME}/runs/detect/train/val_batch0_pred.jpg', width=1000)

#Validate Custom Model

%cd {HOME}
#Predictions for training dataset
!yolo task=detect mode=val model={HOME}/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml
```

B.3 Euclidean Geometric Model

The python code used in this thesis for implementing the Euclidena geometric model for the grip point localization can be found at `https://github.com/LauraMunozAmaya/Handle-door-detection-and-grip-point-localization/blob/2ee33f63febf1d9a9a82c96ddba0e47c80a0a873/Geometric_Model.ipynb`.

```
#Instaling super gradients and neccesay packages
%%capture
!pip install -q git+https://github.com/Deci-AI/super-gradients.git@stable
!pip install -q super-gradients==3.2.0
!pip install -q roboflow
!pip install -q supervision


#Supergradients libraries
from super_gradients.training import Trainer
from super_gradients.training.dataloaders.dataloaders import coco_detection_yolo_format_train
from super_gradients.training.dataloaders.dataloaders import coco_detection_yolo_format_val

#Super gradients libraries for model
from super_gradients.training.losses import PPYoloELoss
from super_gradients.training.metrics import DetectionMetrics_050
from super_gradients.training.models.detection_models.pp_yolo_e import PPYoloEPostPredictionCallback


#Dataset Just for test
from roboflow import Roboflow

rf_t = Roboflow(api_key="XXXXXXX")
project_t = rf_t.workspace("laura-munoz").project("test_door_handle_detection")
dataset_t = project_t.version(2).download("yolov5")

#Selecting best model
from super_gradients.training import models
import os
best_model = models.get(config.MODEL_NAME,
                  num_classes=config.NUM_CLASSES,
                  checkpoint_path=os.path.join(config.CHECKPOINT_DIR, config.EXPERIMENT_NAME, '/content/
                       average_model.pth'))

#training
trainer = Trainer(experiment_name=config.EXPERIMENT_NAME,
             ckpt_root_dir=config.CHECKPOINT_DIR)

#Selecting a random image from path image
import cv2
import numpy as np
import os
import random

#path to the directory containing images
path_img = ("/content/Test_Door_Handle_Detection-2/test/images/")

#List all images in the directory
file_img = os.listdir(path_img)

#Selecting a random subset of images from the list (file_imgs)
selected_imgs = random.sample(file_img, k=1)

#initialize a list to store the loaded images
images = []

#Loop through the selected image file names, read and convert in RGB color
for img_file in selected_imgs:
  img_path = os.path.join(path_img, img_file) #recreating path file complete name
  img = cv2.imread(img_path) #Uploading images in cv2 format
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #converting in RGB color

  if img is not None:
   images.append(img) #Adding image selected in "images" list
  else:
   print(f"Failed to load image: {img_file}")

#Appliying best model to the random image
outputs = best_model.predict(images)

#Show random image with the model predicted apliyed
print(outputs)
outputs.show()

# Tomando la informacin generada por la prediccion realizdad donde se guarda las coordenadas de los bounding box
     predichos
output_str = str(outputs)

# Buscar la cadena que contiene la informacin de ’prediction’
prediction_str_start = output_str.find("prediction=")

if prediction_str_start != -1:
   # Encontrar el final de la cadena de ’prediction’
   prediction_str_end = output_str.find(")", prediction_str_start)

   # Extraer la subcadena que contiene la informacin de ’prediction’
   prediction_info_str = output_str[prediction_str_start:prediction_str_end + 1]

   # Ahora puedes trabajar con la subcadena que contiene la informacin de ’prediction’
   print(prediction_info_str)
else:
   print("No se encontr la informacin de ’prediction’ en la cadena de texto de ’output’.")


prediction=DetectionPrediction(bboxes_xyxy=array([[246.73242 , 44.691605, 611.0118 , 437.4508 ],
     [529.7692 , 250.95485 , 587.66473 , 267.86963 ]], dtype=float32)

output_str = str(outputs)

# Buscar las cadenas que contienen la informacin de ’prediction’ y ’labels’
prediction_str_start = output_str.find("prediction=")
labels_str_start = output_str.find("labels=")

if prediction_str_start != -1 and labels_str_start != -1:
   # Encontrar el final de la cadena de ’prediction’
   prediction_str_end = output_str.find(")", prediction_str_start)
   labels_str_end = output_str.find("]", labels_str_start) + 1 # Asumiendo que labels es una lista

   # Extraer las subcadenas que contienen la informacin de #’prediction’ y ’labels’
   prediction_info_str = output_str[prediction_str_start:prediction_str_end + 1]
   labels_info_str = output_str[labels_str_start:labels_str_end]

   # Ahora puedes trabajar con las subcadenas que contienen #la informacin de ’prediction’ y ’labels’
   print("Informacin de ’prediction’:", prediction_info_str)
   print("Informacin de ’labels’:", labels_info_str)
else:
   print("No se encontr la informacin completa en la cadena de texto de ’output’.")

# Usar expresiones regulares para extraer el array de labels
array_labels = re.search(r’array
’, labels_info_str).group(1)

# Convertir la cadena de labels en un array de nmeros
labels_array = np.array(eval(array_labels))

# Encontrar la posicin donde el valor es ’1’
posicion_1 = np.where(labels_array == 1)[0]

# Imprimir la posicin
print(posicion_1)

# Usar expresiones regulares para encontrar las coordenadas dentro de los corchetes
coordenadas_encontradas = re.findall(r’
([^
]*)]’, prediction_info_str)

posicion_1= int(posicion_1)
# Extraer las coordenadas del segundo conjunto (ndice 1)
segundo_conjunto = coordenadas_encontradas[posicion_1]

# Convertir la cadena de coordenadas en una lista de nmeros
coordenadas_handledoor = [float(x) for x in segundo_conjunto.split(’, ’)]

# Imprimir las coordenadas del segundo conjunto
print(coordenadas_handledoor)

side = input("The door handle is in right or left side:")

The door handle is in right or left side:left

####### APPLYING GEOMETRIC MODEL #######

import matplotlib.pyplot as plt
import matplotlib.patches as patches
from matplotlib.transforms import Affine2D

# Creating figure and axes
fig, ax = plt.subplots(1)

#Converting coords class to plane list (because coords_class is anidated list, it means have twice [[]])
#Coordinates here are in a range 0 to 1
#bbox = [float(value) for value in coords_class[0]]
```

```
#print("Bounding box values:", bbox) # botton left corner Coordinate's (x,y), width and height

#Adjusting coordinates to img scale (640 to 640)
img_height, img_width = img.shape[:2] #img's size

#Creating the rectangle of the bounding box from scaled bbox_coordinates
#with patches form Matplotlib its created the rectangle using xy bottom left corner and ..
#This rectagles extends from this corner +width and height from lable's file
rect = patches.Rectangle((coordenadas_handledoor[0]-ajuste_x, coordenadas_handledoor[1]-ajuste_y),
      coordenadas_handledoor[2]-coordenadas_handledoor[0], coordenadas_handledoor[3]-coordenadas_handledoor[1],
      linewidth=2, edgecolor='r', facecolor='none')
print("Rectangle coordinates:", rect.get_bbox())

#Adding the rectangle to plot
ax.add_patch(rect)

#Saving the bbox coordinates generated for Rectangle function in 'rect_bbox' variable
rect_bbox = np.array(rect.get_bbox()).flatten()
#Extracting splits values of the box (Remeber that this coordinates are bottom left corner), and saving an array
      for better access
x_min, y_min, x_max, y_max = rect_bbox

#Calculating Bounding box's center coordinates:
x_center = (x_min + x_max)/2
y_center = (y_min + y_max)/2

#Grab point coordinates in the RIGTH size for X axes, because Y axes it stays the same
x_right = (x_center + x_max)/2

#Grip point coordinates in the LEFT size for X axes, because Y axes it stays the same
x_left = (x_min + x_center)/2

#Identigying if handle door is left or rigth
if side == 'right':
  #Adding the grip point to the image in the moment manually
  #Saving grip point coordinates
  grip_point = [x_right, y_center]
  ax.plot(x_right, coordenadas_handledoor[1], marker='x', markersize=8, color='green')
else:
  grip_point = [x_left, y_center]
  ax.plot(x_left, coordenadas_handledoor[1], marker='x', markersize=8, color='green')

#Adjusting axes limits for making match with real image dimensions
ax.set_xlim(0, img_width) #X axes
ax.set_ylim(img_height, 0) #Y axes -Invest y axes for have coherence with image representation

# Showing image
ax.imshow(img)
coordenadas_handledoor

[529.7692, 250.95485, 587.66473, 267.86963]

#Gri point coordinates
grip_point

[305.375, 290.5]
```

References

teinkraus', D., I. U. C. K., & ', P. Y. I. M. A. R. D. (2005). *Using gpus for machine learning algorithms.*

Bunn, J. H. (2017). *Archimedes' lever*. Springer International Publishing. https://doi.org/10.1007/978-3-319-46106-9_2

Casas, E., Ramos, L., Bendek, E., & Rivas-Echeverria, F. (2023). Assessing the effectiveness of yolo architectures for smoke and wildfire detection. *IEEE Access*, *11*, 96554–96583. https://doi.org/10.1109/ACCESS.2023.3312217

Choutri, K., Lagha, M., Meshoul, S., Batouche, M., Bouzidi, F., & Charef, W. (2023). Fire detection and geo-localization using uav's aerial images and yolo-based models. *Applied Sciences*, *13*, 11548. https://doi.org/10.3390/app132011548

Community, N. (2020). *Numpy community.* https://numpy.org/doc/1.18/

company, S. (2023). *Super gradients.* https://docs.deci.ai/super-gradients/latest/documentation/source/welcome.html

company, U. (2019). *Ultralytics.* https://docs.ultralytics.com/

Deci.ai. (2023). *Yolo-nas by deci achieves sota performance on object detection using neural architecture search.* https://deci.ai/blog/yolo-nas-object-detection-foundation-model/

Dhillon, A., & Verma, G. K. (2020, June). *Convolutional neural network: A review of models, methodologies and applications to object detection.* https://doi.org/10.1007/s13748-019-00203-0

Diwan, T., Anirudh, G., & Tembhurne, J. V. (2023). Object detection using yolo: Challenges, architectural successors, datasets and applications. *Multimedia Tools and Applications*, *82*, 9243–9275. https://doi.org/10.1007/s11042-022-13644-y

Documentation, O. (2013). *Opencv documentation.* https://docs.opencv.org/4.x/d1/dfb/intro.html

Documentation, R. (2022). *Roboflow.* https://docs.roboflow.com/

Documentation, T. (2021). *Torch.* https://pytorch.org/docs/stable/index.html doorhandler. (2023). *Door handle 2 dataset.* https://universe.roboflow.com/doorhandler/door-handle-2

Fernández-Caramés, C., Moreno, V., Curto, B., Rodríguez-Aragón, J. F., & Serrano, F. J. (2014). A real-time door detection system for domestic robotic navigation. *Journal of Intelligent and Robotic Systems: Theory and Applications*, *76*, 119–136. https://doi.org/10.1007/s10846-013-9984-6

Foroughi, F., Chen, Z., & Wang, J. (2021). A cnn-based system for mobile robot navigation in indoor environments via visual localization with a small dataset. *World Electric Vehicle Journal*, *12*. https://doi.org/10.3390/wevj12030134

Foundation, T. P. S. (2009). *The python software foundation.* https://docs.python.org/

Goe, G. (1972). Archimedes' theory of the lever and mach's critique. *Studies in History and Philosophy of Science Part A*, *2*, 329–345. https://doi.org/https://doi.org/10.1016/0039-3681(72)90002-7

Hao, Y., Pei, H., Lyu, Y., Yuan, Z., Rizzo, J.-R., Wang, Y., & Fang, Y. (2022). Understanding the impact of image quality and distance of objects to object detection performance. http://arxiv.org/abs/2209.08237

Hunter, J. D. (2003). *Matplotlib.* https://matplotlib.org/stable/users/index.html

Jiang, P., Ergu, D., Liu, F., Cai, Y., & Ma, B. (2021). A review of yolo algorithm developments. *Procedia Computer Science*, *199*, 1066–1073. https://doi.org/10.1016/j.procs.2022.01.135

Krichen, M. (2023). Convolutional neural networks: A survey. *Computers*, *12*. https://doi.org/10.3390/computers12080151

kursi kulkas. (2022). Trainingdoor$_d$*ataset. Roboflow Universe.* https://universe.roboflow.com/kursi-kulkas/trainingdoor

Lecrosnier, L., Khemmar, R., Ragot, N., Decoux, B., Rossi, R., Kefi, N., & Ertaud, J. Y. (2021). Deep learning-based object detection, localisation and tracking for smart wheelchair healthcare mobility. *International Journal of Environmental Research and Public Health*, *18*, 1–17. https://doi.org/10.3390/ijerph18010091

Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2022). A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, *33*, 6999–7019. https://doi.org/10.1109/TNNLS.2021.3084827

Lin, W., Adetomi, A., & Arslan, T. (2021). Low-power ultra-small edge ai accelerators for image recognition with convolution neural networks: Analysis and future directions. *Electronics (Switzerland)*, *10*. https://doi.org/10.3390/electronics10172048

Ltd., C. (2020). *Ubuntu 20.04.* https://releases.ubuntu.com/focal/

Magro, T. D., & García-Pérez, M. J. (2019). University of the basque country (upv/ehu) on euclidean diagrams and geometrical knowledge-sobre los diagramas euclidianos y el conocimiento geométrico. *34*, 255–276. https://doi.org/10.2307/26775175

Moritz, H. (n.d.). *Geodetic reference system 1980*.

Nickolls, J., & Dally, W. J. (2010). The gpu computing era. *IEEE Micro*, *30*, 56–69. https://doi.org/10.1109/MM.2010.41

Nugraha, I. G. B., & Supangkat, S. H. (2023). *International journal of intelligent systems and applications in engineering enhancing abandoned object detection with dual background models and yolo-nas* (2). www.ijisae.org

of Electrical, I., & Engineers, E. (n.d.). *2017 3rd international conference on control, automation and robotics : Iccar 2017 : 22 apr - 24 apr, 2017, nagoya, japan*.

of Electrical, I., Engineers, E., on Cyber-Physical Systems, I. C., & Online), I.-T. (: 2. : (2020). *2020 9th mediterranean conference on embedded computing (meco)*.

Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Phillips, J. C. (2008). Gpu computing. *Proceedings of the IEEE*, *96*, 879–899. https://doi.org/10.1109/JPROC.2008.917757

Paiva, A. C., of Computing, U. F. F. I., of Electrical, I., de Janeiro Section., E. E. R., of Electrical, I., & Engineers., E. (2020). *Proceedings of the 2020 international conference on systems, signals and image processing (iwssip) : July 1-3, 2020, niterói, brazil*.

PETERFLACH, P. A. F. (n.d.). *An analysis of rule evaluation metrics johannes f ¨ urnkranz*.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (n.d.). *Ros: An open-source robot operating system*. http://stair.stanford.edu

Ramalingam, B., Yin, J., Elara, M. R., Tamilselvam, Y. K., Rayguru, M. M., Muthugala, M. A. J., & Gómez, B. F. (2020). A human support robot for the cleaning and maintenance of door handles using a deep-learning framework. *Sensors (Switzerland)*, *20*, 1–18. https://doi.org/10.3390/s20123543

Ramôa, J. G., Lopes, V., Alexandre, L. A., & Mogo, S. (2021). Real-time 2d–3d door detection and state classification on a low-power device. *SN Applied Sciences*, *3*. https://doi.org/10.1007/s42452-021-04588-3

rethink imcmahon. (2018). *Ros image saver*. https://gist.github.com/rethink-imcmahon/77a1a4d5506258f3dc1f

Robotics, O. (2008a). *Rospy*. http://wiki.ros.org/rospy

Robotics, O. (2008b). *Sensor$_m$sgs*. http://wiki.ros.org/sensor_msgs

Robotics, O. (2009). *Cvbridge.* http://wiki.ros.org/cv_bridge

Robotics, O. (2020). *Ros noetic.* https://wiki.ros.org/noetic

Rosebrock, A. (2018). *Yolo object detection with opencv.* https://pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

Saha, S. (2018). *A comprehensive guide to convolutional neural networks — the eli5 way.* https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Singh, K. J., Kapoor, D. S., Thakur, K., Sharma, A., & Gao, X. Z. (2022). Computer-vision based object detection and recognition for service robot in indoor environment. *Computers, Materials and Continua*, *72*, 197–213. https://doi.org/10.32604/cmc.2022.022989

Suárez, R., Rosell, J., Vinagre, M., Cortes, F., Ansuategui, A., Maurtua, I., Martin, D., Guash, A., Azpiazu, J., Serrano, D., & García, N. (n.d.). *Robot operating system (ros).* http://wiki.ros.org/catkin

Terven, J., & Cordova-Esparza, D. (2023). A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. https://doi.org/10.3390/make5040083

Thesis, B., Marsman, R., Toorn, K. V. D., & Wams, T. (n.d.). *Optimizing object detection models for flood risk assessment by street view imagery.*

Tian, Y. (2020). Artificial intelligence image recognition method based on convolutional neural network algorithm. *IEEE Access*, *8*, 125731–125744. https://doi.org/10.1109/ACCESS.2020.3006097

Valtchev, S. S., Babuska, R., Júri, T.-D., Arguente, D. M. H. S. F., de Brito Palma Vogal, D. L. F. F., & Valtchev, D. S. S. (2014). *Francisco miguel da silva vieira do coito study on the development of an autonomous mobile robot.*

Verne, J. (n.d.). *"la ciencia, muchacho, está hecha de errores, pero errores que son buenos de cometer, pues conducen poco a poco hacia la verdad."*

Vujović, Ž. (2021). Classification model evaluation metrics. *International Journal of Advanced Computer Science and Applications*, *12*, 599–606. https://doi.org/10.14569/IJACSA.2021.0120670

Xu, B., of Electrical, I., Section, E. E. H., of Electrical, I., & Engineers, E. (2018). *Proceedings of 2018 ieee 4th information technology and mechatronics engineering conference (itoec 2018) : December 14-16, 2018, chongqing, china.*

Xu, S., Wang, J., Shou, W., Ngo, T., Sadick, A. M., & Wang, X. (2021). Computer vision techniques in construction: A critical review. *Archives of Computational Methods in Engineering*, *28*, 3383–3397. https://doi.org/10.1007/s11831-020-09504-3

Zhao, L., & Li, S. (2020). Object detection algorithm based on improved yolov3. *Electronics (Switzerland)*, *9*. https://doi.org/10.3390/electronics9030537

Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J. (2023). Object detection in 20 years: A survey. *Proceedings of the IEEE*, *111*, 257–276. https://doi.org/10.1109/JPROC.2023.3238524

2024 ***A Yolo-NAS Based Approach for Door and Door Handle Detection and Integration of a Euclidean Geometric Model For Grip Point Localization: An Application In Robot Navigation.*** Laura Milena Muñoz Amaya

Masters
Program
in **Geospatial**
Technologies