



**Politecnico
di Torino**

POLITECNICO DI TORINO

**MASTER OF SCIENCE DEGREE IN
MECHATRONIC ENGINEERING**

Master Thesis

**Human-Robot perception for
collaborative operations of Industrial
Mobile Robots**

Supervisors

Prof. Marina INDRI

Candidate

Jia Hui Lisa LIN

April 2022

Summary

During the last decade, robots have played a very important role in many fields, such as production, manufacturing, social environments, space exploration, etc. In many of those scenarios, the robots work in collaboration with human operators. For this reason, the capacity of the robot to detect and track human operators is crucial for an effective and efficient Human-Robot Interaction.

Since one of the key aspects of robots is the capacity of executing repetitive tasks in a much more efficient and effective way compared to humans, a situation where the mobile robot remains in idle and reacts only when the operator requests the robot to execute a task, is considered in order to reduce the possibility of unexpected situations.

The purpose of this thesis is to develop a successful methodology for a mobile robot to detect and recognize specific operators in an industrial scenario and, according to which operator it identifies, be able to execute one or more predefined tasks. In particular, the human operator role is identified by a symbol attached to them.

The solution proposed in this thesis is a sensor fusion solution that combines two different cameras in order to increase the accuracy and reliability of the detection. One camera is used to detect the symbol placed on the operator, while the second camera detect the operator body using a real-time object detection algorithm. Only when both detections are positive the robot enters into the collaborative mode. At the same time, this thesis aims to implement a low cost solution in terms of both computational complexity and material cost.

Table of Contents

| | |
|---|----|
| List of Figures | VI |
| 1 Introduction | 1 |
| 1.1 Introduction | 1 |
| 1.2 Human-robot perception | 2 |
| 1.3 Goal of the thesis | 3 |
| 2 State of the art | 4 |
| 2.1 State of the art | 4 |
| 2.2 Computer Vision | 5 |
| 2.2.1 Deep learning | 6 |
| 2.2.2 Convolutional neural network | 7 |
| 2.3 Introduction to Image processing | 10 |
| 2.3.1 Color space | 10 |
| 2.3.2 Threshold | 11 |
| 2.4 Sensors | 12 |
| 2.4.1 Camera | 12 |
| 2.4.2 Laser Distance Sensor | 13 |
| 3 Tools | 14 |
| 3.1 Microsoft Kinect 360 | 14 |
| 3.2 Raspberry Pi | 15 |
| 3.3 OpenCV | 16 |
| 3.4 Robot Operating System | 16 |
| 4 Object detection | 17 |
| 4.1 Introduction to object detection | 17 |
| 4.2 Histograms of Oriented Gradient (HOG) | 19 |
| 4.3 YOLO | 20 |
| 4.3.1 How YOLO works | 20 |
| 4.3.2 YOLOv2 | 23 |

| | | |
|----------|---|-----------|
| 4.3.3 | YOLOv3 | 24 |
| 5 | Implementation | 26 |
| 5.1 | Symbol detection | 28 |
| 5.2 | Human detection | 32 |
| 5.3 | Data fusion | 35 |
| 5.4 | ROS | 39 |
| 6 | Simulation and testing | 40 |
| 6.1 | Histograms of Oriented Gradient | 40 |
| 6.2 | YOLOv3 | 40 |
| 7 | Conclusion | 42 |
| 7.1 | Conclusion | 42 |
| 7.2 | Future work | 42 |
| | Bibliography | 44 |

List of Figures

| | | |
|------|--|----|
| 2.1 | Basic concept of Artificial Intelligence, Machine learning, Deep learning [21] | 6 |
| 2.2 | Basic neuron structure [22] | 7 |
| 2.3 | Feed-forward Artificial Neural Network [22] | 8 |
| 2.4 | Convolutional Neural Network [23] | 8 |
| 2.5 | Example of convolution process with a 3x3 filter | 9 |
| 2.6 | Example of pooling process | 9 |
| 2.7 | Composition of RGB [25] | 10 |
| 2.8 | HSV color scale | 11 |
| 2.9 | Example of camera (AUKEY webcam) | 12 |
| 2.10 | Example of stereo camera (Stereolabs 2 ZED) | 13 |
| 2.11 | Example of LiDAR camera (Velodyne VLP-16 Puck LITE) | 13 |
| 3.1 | Microsoft Kinect 360 camera | 14 |
| 3.2 | Raspberry Pi 3B+ single board [27] | 15 |
| 3.3 | Communication between nodes in ROS [29] | 16 |
| 4.1 | Difference between classification, object detection and instance segmentation | 17 |
| 4.2 | Multiple RoIs [30] | 18 |
| 4.3 | Gradient representation [31] | 19 |
| 4.4 | Kernal filter | 20 |
| 4.5 | Spacial orientation cells | 20 |
| 4.6 | HOG steps structure | 21 |
| 4.7 | YOLO architecture [35] | 21 |
| 4.8 | Step 1: Residual blocks | 22 |
| 4.9 | Step 2: Bounding box regression | 22 |
| 4.10 | IOU computation | 23 |
| 4.11 | Step 4: Intersection Over Union result | 23 |
| 4.12 | YOLOv2 performance [28] | 24 |
| 4.13 | YOLOv3 layers description [38] | 25 |

| | | |
|------|---|----|
| 4.14 | YOLOv3 performance [38] | 25 |
| 5.1 | Hardware connection | 27 |
| 5.2 | Flowchart of the algorithm proposed | 27 |
| 5.3 | Example of QR code detection | 29 |
| 5.4 | Example of QR code detection | 30 |
| 5.5 | Trackbar to determine the color of the symbol | 30 |
| 5.6 | Image after the contouring process | 31 |
| 5.7 | Symbol detection | 33 |
| 5.8 | Different color/shape symbol: HSV scale, contouring process, symbol detection | 33 |
| 5.9 | Examples of object presents in the COCO dataset [42] | 35 |
| 5.10 | YOLO3: full body detection with a confidence value of 98% | 35 |
| 5.11 | YOLO3: partial body detection with a confidence value of 96% | 36 |
| 5.12 | YOLO3-tiny: full body detection with a confidence value of 67% | 36 |
| 5.13 | YOLO3-tiny: no detection when the human figure is not fully visible | 37 |
| 5.14 | Checkerboard for camera calibration [43] | 37 |
| 5.15 | Some examples of images captured for the kinect camera calibration | 37 |
| 5.16 | Some examples of images captured for the webcam calibration | 38 |
| 5.17 | ROS architecture | 39 |

Chapter 1

Introduction

1.1 Introduction

During the last decade, robots have played a very important role in many fields, such as production, manufacturing, social environments, space exploration, etc. In fact, the International Federation of Robotics (IFR) have reported that in 2020 there were 2.7 million industrial robots operating in factories around the world [1].

In most cases, in the industrial environments the robots are confined in a work-space limited by heavy fences separating the manufacturing space between robot and the operator for safety reasons. However, this lack of interaction reduces the efficiency and the flexibility of the production considerably. Moreover, although thanks to modern technologies, the robots are able to work autonomously most of the time, the human presence remains important, since there is still the necessity to supervise the work or to complete tasks that robots cannot be trained to do.

For this reason, the number of collaborative robots, or simply known as *cobot*, has notably increased over the years. The main idea behind the collaborative robots is to assist humans' actions in a safe manner through direct interactions in a shared work-space, for example, by reducing the workload of the operator or by performing actions that are challenging for humans.

The idea of this thesis is to propose and develop a successful methodology for a robot to detect and recognise specific operators and, according to which operator it identifies, be able to execute one or more predefined tasks.

Since one of the key aspects of robots is the ability of executing repetitive tasks in a much more efficient and effective way compared to humans, a situation where the mobile robot remains in idle and reacts only when the operator requests the robot to execute a task, is considered in order to reduce the possibility of unexpected situations.

There are many different elements that a robot can use to recognize and differentiate each operator, such as face features, colour of the clothes, hand gesture, badges, etc. Each of them have advantages and disadvantages, for example, the face recognition is one of the most commonly used method and it is applied in various fields, even outside of the Human-Robot Interaction scenario, such as surveillance systems, however it is affected by many conditions, like: orientation of the face, facial expression, presence of accessories (glasses, mask, etc.) [2]. Therefore, in order to minimize the percentage of false positive, it requires a big computational power and a huge set of data samples to train the algorithm, or it is required to be paired with other algorithms to improve the human recognition [3]. Techniques such as colours or hand gesture, instead, are limited by the number of possible choices available.

Considering the scenario where the robot is operating, the recognition of a symbol (or badge, tag) is sufficient for the robot to identify the operator. This solution does not limit the movement of the operator, does not require a high computational complexity level and allows a good number of combination of color and shape of the symbol to assign to each operator.

The solution proposed in this thesis is a sensor fusion solution that combines two different cameras in order to increase the precision and reliability of the detection. One camera is used to detect the symbol placed on the leg of the operator, while the second camera detect the operator and only when both detections are positive the robot enters into the collaborative mode and starts to execute the predefined tasks.

Hence, this thesis proposes a low cost solution for human identification in term of both computational complexity and material cost.

1.2 Human-robot perception

Human-Robot Interaction (HRI) is a field that focuses on researching and designing methods dedicated to the communication between a human and a robot. In particular, Fang et al. [4] defined the Human-Robot Interaction as *'the process that conveys the human operators' intention and interprets the task descriptions into a sequence of robot motions complying with the robot capability and the working requirements'*.

HRI is a multidisciplinary field that includes multiples sciences such as robotics, artificial intelligence, human-computer interaction, but also psychology, social sciences and other sciences based on the environment where the robot is deployed.

Based on the tasks that need to be performed, Human-Robot Interaction (HRI) can be classified as follows [5]:

- **Human-Robot Coexistence** - when humans and robots are in the same

environment but generally do not interact with each other since they have different aims. The coexistence is generally limited to collision avoidance.

- **Human-Robot Cooperation** - when humans and robots work in the same work-space, share the same working time and have a common purpose. In this case, more sensing techniques are needed for collision detection and avoidance.
- **Human-Robot Collaboration** - when humans and robots work with a direct interaction with each other; the collaboration can happen in two different modalities: physical collaboration or contactless collaboration. In the first case, there is an explicit contact with an exchange of forces between human and robot and the robot is able to predict the human motion and react accordingly. In the second case, the exchange of information happens through direct communication (speech, gestures, etc.) or indirect communication (facial expression, eye gaze intention, etc.).

In Human-Robot Interaction, the robot perception plays one of the most important roles since an accurate human detection and tracking can improve the interaction efficiently.

1.3 Goal of the thesis

The goal of this thesis is to propose a successful method for the robot to detect the operator through a particular symbol (e.g. QR, symbol, logo, etc) and, according to which operator it identifies, be able to execute a predefined task. In this thesis, we will use a Microsoft Kinect for human detection paired with a standard webcam for the symbol identification.

The chapters are organized as follows:

Chapter 2: focuses on the related works on human-robot interactions.

Chapter 3: presents the tools that are used in this research, both software and hardware.

Chapter 4: presents an overview of existing object detection algorithms.

Chapter 5: describes the algorithm implemented to realize the desired task.

Chapter 6: illustrates the results obtained from testing.

Chapter 7: concludes the thesis providing some final considerations and presents possible future works.

Chapter 2

State of the art

2.1 State of the art

In the last years, with the increase of the presence of collaborative robots, many researchers have focused their attention on the concept of Human-Robot Interaction, which led to the development of different instruments of perception, such as hand gesture, movement tracking, face recognition, wearable devices, etc.

In [6], the authors propose human-robot collaboration through natural wearable sensors in order to perform an hand-over task. The robot is able to recognise the human intentions through the human forearm posture and muscle activities, which are detected by an electromyography (EMG) wearable device. The signals are also used to send to the robot useful detailed information about the object to be handed-over. Same approach was considered in [7] but to perform an assembly human-robot collaboration task. However, to ensure a higher safety, a safety light curtain and a safety relay were used to detect the presence of the human arm. The main drawback of this method is the limitation of the possible actions that the operator can execute while interacting with the robot, which leads to be used only in very specific operations.

Many applications rely on the capability of the robot to detect, recognize and track the operator in a human shared environment. Hand gesture and face tracking are among the most common methods to interact with the robot since they allow to transmit a high variety of commands to the robot without any physical contacts.

In general, the sensors that are commonly used for those applications are vision or laser based, such as: LiDAR cameras, RGB-D cameras, stereo cameras, laser bases systems. Each method has advantages and disadvantages and they highly depend on the characteristics of the device and the environment in which it is used.

For example, in [8] a real-time and background independent hand gesture detection is proposed: Mazhar et al. utilize the OpenPose library [9] to extract

the 2D skeletal coordinates from images obtained with Microsoft Kinect V2. It is able to extract each joint independently from the overall body pose, so it is not required another sensor to localize the hands. However, the system is limited in depth range vision and the quality of the environment (e.g, bright light).

An interesting application is presented in [10], where the authors combine indoor localization, face recognition and robot navigation. First, the robot receives a Bluetooth signal that allows it to estimate the approximate position of the target. Then, with face recognition and indoor localization the mobile robot is able to identify the specific target and interact with them. Histogram of Oriented Gradients (HOG) [11] is used to detect the face and linear SVM classifier/KNN classifier is adopted for classifying the detected face.

Jian et al. [12] proposed a Haar-like classifier based on AdaBoost algorithm with a RGB-D camera, which provides both vision and depth information. To increase the accuracy of recognition, the authors provided an auxiliary recognition method based on the color features on the human figure.

In order to increase the robustness and the accuracy of the tracking task, multi-sensor approaches have been considered. Bozorgi et al. [13] proposed the integration of a RGB camera and Laser Range Finder (LRF). The first sensor, with the use of the deep learning algorithm, You Look Only Once (YOLO), is able to detect humans with high accuracy, while the laser-based sensor is usually used for leg detection and it is able to provide information about the distance. A similar approach is proposed by Yan et al. [14], where they developed an approach based on laser and monocular vision for leg detection.

Matsubara et al. [15] proposed an algorithm based on the fusion of two RGB-D cameras and OpenPose for human detection by tracking the back view of the target. The first time the algorithm detects the person, it generates an histogram containing the colors present in the human region. Subsequently, it is used to determine and track the target. Wang et al. [16] developed an algorithm that combines a monocular camera and an ultrasonic sensor by means of the extended Kalman filter (EKF).

In contrast with the authors above, Wang et al. [17] proposed a human detection and tracking method based on a single RGB-D camera, focusing on skeleton tracking and clothes color analysis.

2.2 Computer Vision

Computer vision is the computer science field that studies the techniques that enables computers to obtain relevant information from digital images, such as photographs and videos.

The logic behind Computer vision is similar to how human vision works [18]:

with the experience of a lifetime the human brain is able to recognize objects, understand how far it is and other information, while computer vision trains a computer to understand visual data by feeding it thousands or millions of images of specific objects.

Computer vision is based on two essential technologies: machine learning and convolutional neural network (CNN).

2.2.1 Deep learning

Machine Learning (ML) is a sub-field of Artificial Intelligence (AI) and it is defined by A. Samuel in 1959 as [19]: "*the field of study that gives computers the ability to learn without being explicitly programmed*".

Hence, the fundamental principle of Machine Learning is to build an algorithm that is able to learn and make decisions based on samples of data without explicit human intervention.

Deep learning [20] is a branch of Machine Learning that consists on multiple layers of interconnected nodes built to learn and process data with multiple levels of abstraction mimicking the behavior of the human brain. Deep learning is widely used in many applications including object detection, visual object detection, speech recognition, etc., and has proven to be more efficient than traditional data processing and other feature extraction techniques.

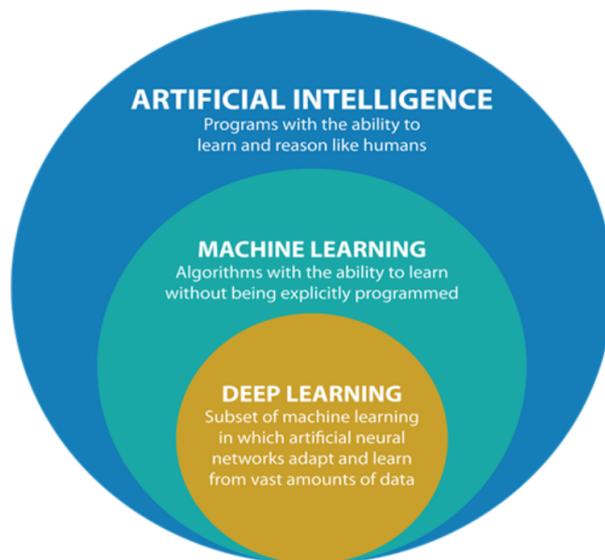


Figure 2.1: Basic concept of Artificial Intelligence, Machine learning, Deep learning [21]

Figure 2.2 shows a basic neuron structure, a single-layer, where a n number

of inputs is weighted and summed. The result is used as input of the activation function (in this case, a step function) with a defined threshold. Then the final output gives us a classification of the input values. If the classification is correct, no changes are made, otherwise, the algorithm adjusts the weights accordingly.

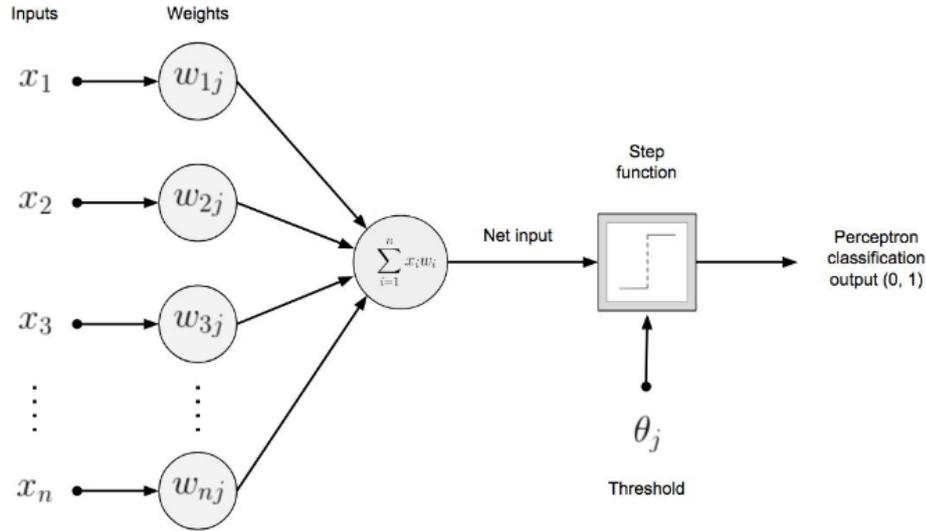


Figure 2.2: Basic neuron structure [22]

The general mathematical architecture of a deep learning model is known as Artificial Neural Network (ANN), while each node is called Artificial Neuron. Figure 2.3 shows an example of neural network:

The computational complexity of the algorithm depends on the number of neurons organised in the layers. The first layer of the network is called *input layer*, the last one is called *output layer*, while internal layers are called *hidden layers*. Every neuron in each layer is associated to a weight and every layer contains one type of activation function. As noted in the basic neural network, the goal of the process is to tune the neuron weight in order to obtain the correct classification.

2.2.2 Convolutional neural network

Convolutional neural network (CNN) is a specific artificial neural network and it is mainly designed for image classifications, which is why it is one of the most commonly used algorithm in applications such as object recognition, image classification and text analysis. The main drawback is the need of an appropriate training phase with a large amount of input data, and the precision of the output highly depends on the quality of the input data.

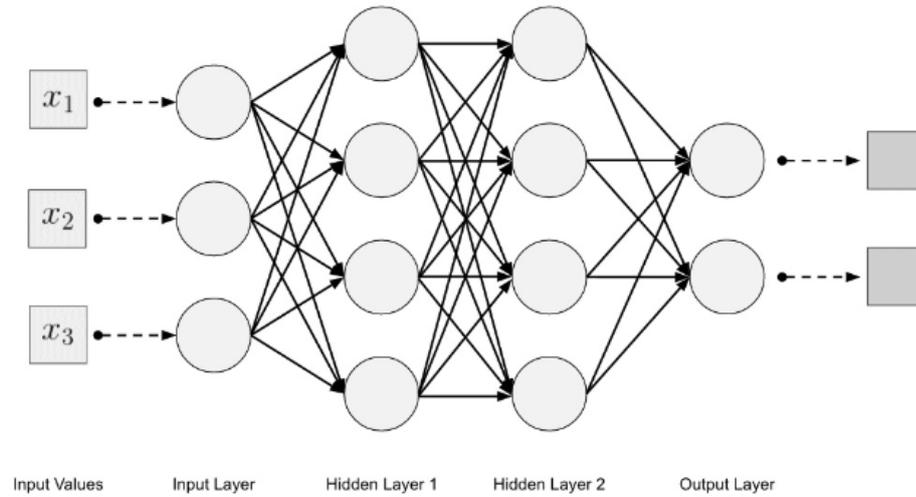


Figure 2.3: Feed-forward Artificial Neural Network [22]

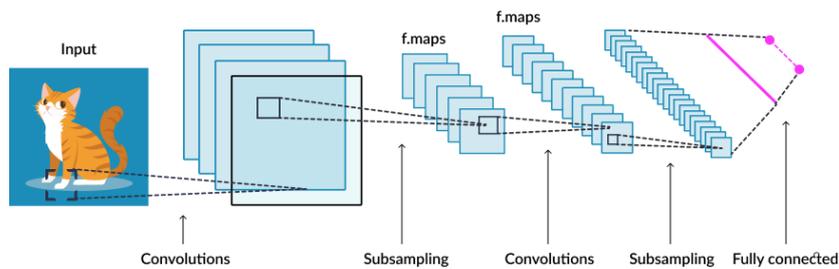


Figure 2.4: Convolutional Neural Network [23]

The CNN architecture can be divided into four main layers: convolutional layer, pooling layer, ReLu correction layer and fully-connected layer.

- **Convolutional layer**

The convolutional layer is the first layer, which applies a convolution filter with the purpose of mapping a set of features in the images received as input. The filter, known also as *kernel*, is a weight vector which slides along the input image pixels in both vertical and horizontal direction and, for each slide, it maps and multiplies a set of input. The result is collected into a vector called *output feature map*.

- **Pooling layer**

The pooling layer is placed between the convolutional layer and the pooling

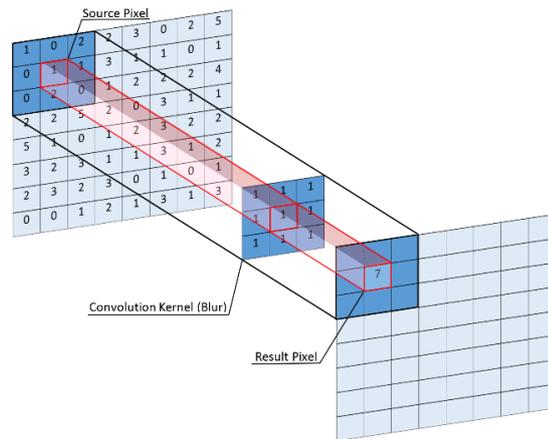


Figure 2.5: Example of convolution process with a 3x3 filter

process, it consists in a gradual reduction of the size of the input data and network parameters, while preserving the important features. The most used pooling layers are average pooling (calculate the average value of the input) and max-value pooling (select the maximum value).

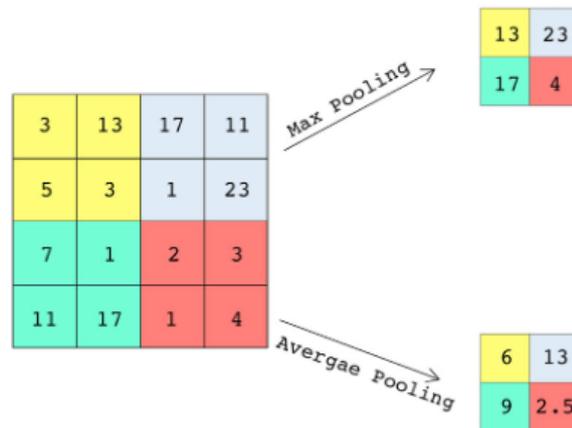


Figure 2.6: Example of pooling process

- **ReLU correction layer**

ReLU (Rectified Linear Units) layer acts as activation function.

- **Fully-connected layer**

The fully-connected layer is the last layer of the neural network. It connects the outputs obtained after all the convolution and pooling layers into one

vector, where each element indicates the probability for the input image to belong to a class.

2.3 Introduction to Image processing

Image processing is a method to perform some operations on a digital image through algorithms. Image processing techniques are usually applied in order to enhance the visual appearance of the image, to obtain useful information or to prepare the images for object recognition [24].

2.3.1 Color space

Digital images have a set of digital values, known as *pixels*. Each pixel value represents the color or the intensity at that specific point in the image. Usually each pixel has a 8-bit representation which means there are 256 different possible values.

Gray scale images have only one matrix of pixels, where the highest value '255' represents the maximum intensity, which is associated to white, and the lowest value '0' is black. Meanwhile, for colored images three matrices of pixels are required for red, green and blue intensity.

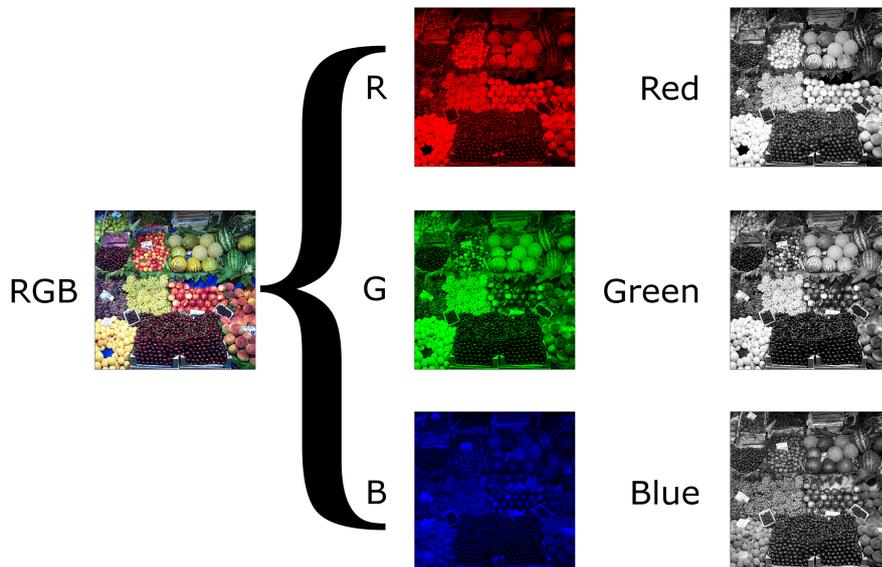


Figure 2.7: Composition of RGB [25]

Another common scale used in image processing is the HSV color scale, which

stands for Hue Saturation Value. HSV is a cylindrical color model and is measured in degrees from 0° to 360° .

- Hue: represents the angle of the color in the RGB colour scale, where 0° hue corresponds to red, 120° to green and 240° to blue.
- Saturation: controls the amount of gray in that particular color, going toward 0% introduces more gray, while increasing toward 100% leads to a faded effect.
- Value: indicates the brightness of the color. 100% brightness means there is not black mixed with the color, while 0% is pure black.

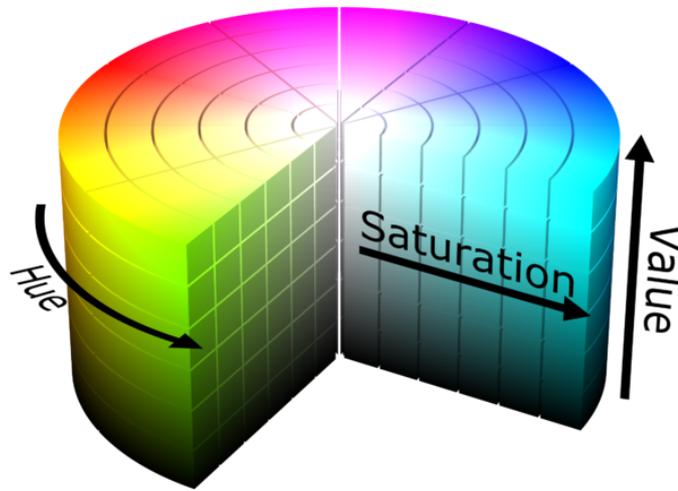


Figure 2.8: HSV color scale

2.3.2 Threshold

Thresholding is one of the most used method for image segmentation to separate an object from the background [26]. By choosing a particular value for the threshold T , the thresholding process converts the gray scale image into a binary one where the pixels are set to 0 if the gray value is below the threshold, or set to 255 if above. The process can be represented with one mathematical function:

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T \end{cases} \quad (2.1)$$

where $g(x,y)$ is the resulting image after the thresholding, $f(x,y)$ is the original image and T is the threshold.

2.4 Sensors

As previously mentioned, sensors play a crucial role in Human-Robot Interaction. They allow the robot to acquire meaningful information from the environment and to recognize the surrounding area and objects. The information obtained from the sensors are used by the robot to perform actions or as input to perform other operations.

In general, sensors can be mounted directly on the robot or installed in the environment, however, the latter case is used in specific applications since it is limited by possible changes of the environment.

Depending on the type of sensor, different information can be collected and transmitted to the robot. For object detection and autonomous navigation (which are the main focus of this thesis), vision sensors and distance sensors are the most commonly used.

2.4.1 Camera

A camera provides "vision" to the robot and obtains data information by capturing the incident light in the form of one or more images and it is stored as an array of pixels. The camera provides information such as objects' colour, shape or texture, which is the reason why is commonly used in face recognition, object detection or image processing.

Cameras are widely available and reasonable cheap, however, normally they are not highly precise and the quality of the image is greatly affected by the environmental conditions (light, fog, etc.). A single camera provides only 2D data and is not able to transmit depth information. However, the depth can be estimated by combining two cameras that observe the same target from different angles. The distance information between the camera and the object is calculated using the triangulation method. In this case, the camera is known as stereo camera. Stereo cameras require a highly computational power and need calibration over time.



Figure 2.9: Example of camera (AUKEY webcam)



Figure 2.10: Example of stereo camera (Stereolabs 2 ZED)

2.4.2 Laser Distance Sensor

Laser Distance Sensor (LDS) included different type of sensors based on laser technology: Light Detection and Raging (LiDAR), Laser Scanner and Laser Range Finder (LRF).

LiDAR is the most used laser method that sends light in the form of a laser beam to map the environment and detect objects. In particular, its principle is based on the time of flight concept: the sensor sends out laser beams in multiple directions into the surrounding environment, then the laser bounces back to the sender sensor. The time that the laser takes to come back is used to calculate the distance between the sensor and the object.

The raw data obtained from the LiDAR sensors consist of a vector of points and each point is represented by its reflectance value and its position in 3D coordinate.

LiDAR sensor is capable of providing depth estimation and dense point cloud generation. It is computationally light weighted and can calculate the distance with high precision, however, the quality of the performance can be affected by the weather and the surface of the object, that compromise the reflectivity of the signal.



Figure 2.11: Example of LiDAR camera (Velodyne VLP-16 Puck LITE)

Chapter 3

Tools

3.1 Microsoft Kinect 360

The Kinect is a motion sensing input device by Microsoft. It was initially used as a motion controller for Xbox consoles: it allows the players to interact with the screen through gestures recognition, body detection and other means without the need of a physical controller.

In this work the first version is used, Kinect for Xbox 360, which is equipped with:

1. a 3D depth sensor that consists of an infrared laser projector combined with a monochrome CMOS sensor with a resolution of 640x480.
2. a RGB camera with a resolution of 640x480 with a Bayer color filter and at a lower frame rate is capable of reaching a 1280x1024 resolution.
3. an array of four microphones capable of voice recognition.
4. a tilt motor with a range of $\pm 27^\circ$.



Figure 3.1: Microsoft Kinect 360 camera

Thanks to the low cost and its reliable depth-sensing technologies, the Kinect has been used also in non-gaming applications.

The libfreenect was developed for this purpose: it is an open source driver which runs on Linux, OSX and Windows. It supports: RGB and depth images, motors, accelerometer, LED and audio.

3.2 Raspberry Pi

The Raspberry Pi is a low-cost single-board computer (SBC) developed by the Raspberry Pi Foundation based in United Kingdom. It is able to perform most of the tasks that a standard computer can do.

A Raspberry pi 3 Model B+ was used to interact remotely with the robot and obtain data from the sensor.

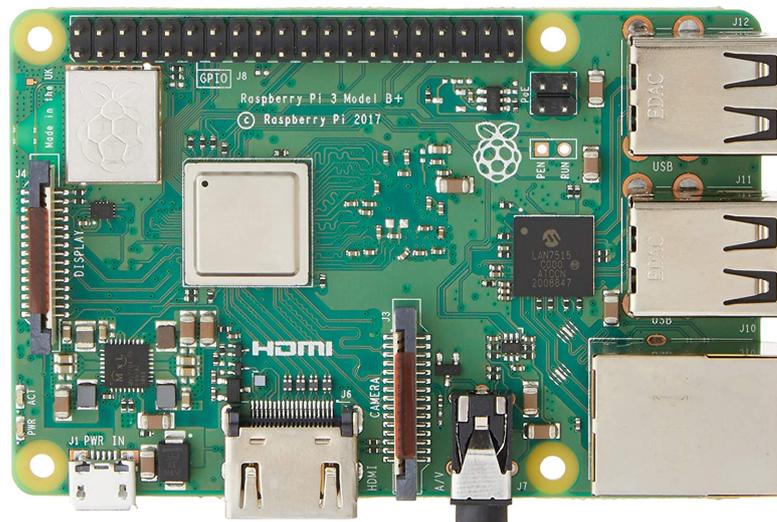


Figure 3.2: Raspberry Pi 3B+ single board [27]

Raspberry Pi 3 B+ is equipped with:

- 1.4GHz 64-bit quad-core processor
- dual-band wireless LAN
- Bluetooth 4.2/BLE
- faster Ethernet
- Power-over-Ethernet support

3.3 OpenCV

OpenCV is an open source computer vision library launched by Intel in 1999. It is written in C and C++ but can interface with other programming languages such as Python, Java, MATLAB and supports Windows, Linux, Android and OSX.

It mainly focuses on image processing, video capture and analysis, but with a library of over 2500 optimized algorithms, OpenCV can be used in a wide area of applications like: object detection, face recognition, surveillance, robot and driver-less car navigation and control, medical imaging.

3.4 Robot Operating System

Robot Operating System (ROS) [28] is an open source meta-operating system for robotic applications. It provides to the user all the services expected by an operating system such as hardware abstraction, low-level device control, device communication, etc.

ROS processes are represented by nodes in a graph architecture, which are connected via topics or services. On top of all nodes there is an always-running Master node, which is connected to all the other nodes and its only job is to monitor the information exchanged between the nodes. Communication between nodes takes place by using the publish/subscribe concept: the publisher sends a message on the ROS network, while the receiver, which is the subscriber, "subscribes" to the topic in order to read the messages. There can be multiple publishers and subscribers to a topic and each topic is implemented to be unidirectional. However, not all the communications in ROS follow the publish/subscribe paradigm. The alternative communication modality is through services. A node provides a service and another one requests the service by sending a message.

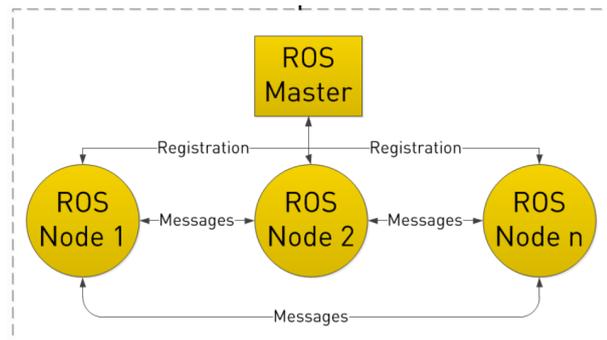


Figure 3.3: Communication between nodes in ROS [29]

Chapter 4

Object detection

4.1 Introduction to object detection

As mentioned in chapter 2.2, object detection is an important research area of the computer vision field. It is a technique used to identify and locate one or more objects within an image or video and it is able to define bounding boxes around the detected objects and define their relative position in the scene. Bounding boxes are the rectangles used to mark the position of the object in the image.

In general, classification is strictly linked to classification, which is a machine learning technique to assign a specific class to the image in consideration.

Instance segmentation is a similar technique to object detection with the only differences that it identifies the class of the object detected by considering each pixel and places the bounding boxes around specific objects.

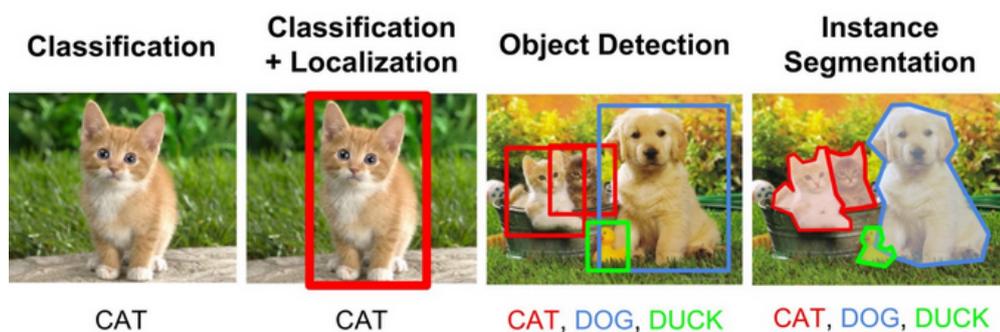


Figure 4.1: Difference between classification, object detection and instance segmentation

Object detection framework can be summarized into three main steps, revolving around the concept of Region of Interest (RoI), which consists in identifying a a

particular area of an image to work on:

1. Generation of Regions of Interests: the algorithm generates a series of bounding boxes on the image;
2. Each RoI is analyzed separately and the visual features are extracted and evaluated in order to detect the presence of a possible predefined class of object.
3. The overlapping boxes that contain the same detected object are merged into one unique box.

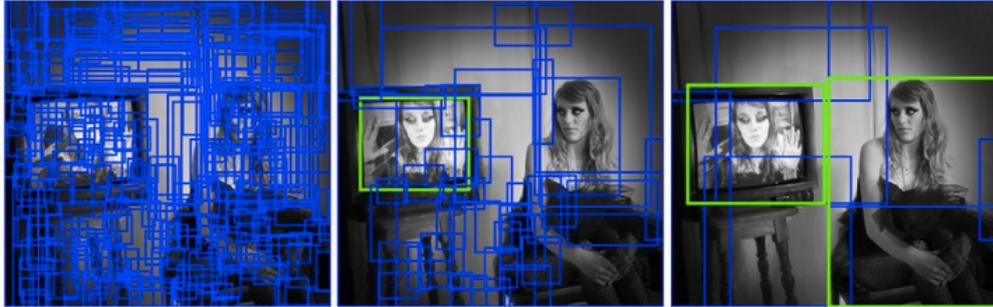


Figure 4.2: Multiple RoIs [30]

The creation of RoIs can be implemented in different ways and they are all based on the *Image Gradient Vector*. The main idea is to determine the intensity and the direction of colors changing among the pixels. The gradient contains partial derivatives, computed as the color difference between adjacent pixels of the image along the principal directions and it is discrete since each pixel is independent from others and cannot be further split.

$$\nabla f(x, y) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} f(x, y + 1) - f(x, y - 1) \\ f(x + 1, y) - f(x - 1, y) \end{bmatrix}$$

The formula can be interpreted using the figure 4.3.

The gradient gives two important values used in the algorithms:

- **Direction:** computed as the arctangent of the ration between the partial derivatives on two directions:

$$\theta = \arctan(g_y/g_x) \quad (4.1)$$

- **Magnitude:** calculated as the L2-norm of the vector

$$g = \sqrt{g_x^2 + g_y^2} \quad (4.2)$$

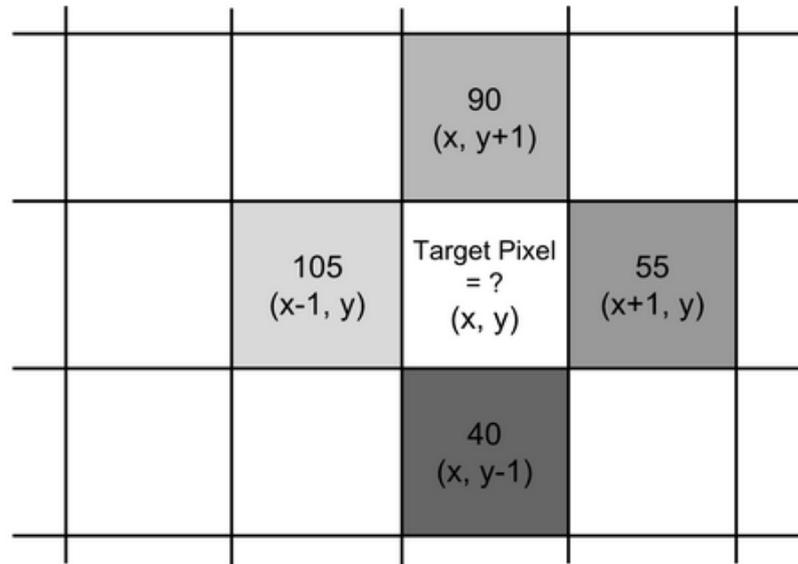


Figure 4.3: Gradient representation [31]

4.2 Histograms of Oriented Gradient (HOG)

Histograms of Oriented Gradient (HOG) is feature descriptor, that is a representation of an image that simplifies the image by extracting useful information from it. The main idea behind the algorithm is that the structure or the shape of objects can be described by the intensity of the gradients or edge directions [32].

The HOG algorithm can be described by the following steps [33]:

1. **Pre-processing:** in order to optimize the performance of the algorithm, all the images involved have to be normalized, in particular imposing an aspect ratio of 1:2.
2. **Gradient computation:** the gradient is calculated along the two main directions, vertical and horizontal, which can be easily obtained by filtering the image with the following kernels:

In this way, all the unnecessary information like the background are removed.

3. **Spacial & orientation cells:** the image is divided into cells of 8x8 pixels and for each cell, a 9-point histogram is calculated, which is essentially a vector of 9 bins corresponding to angles with range of 20° .

Block normalization: for a block of 2x2 cells, we calculate the HOG descriptor for each cell and all the values are assembled to form a single vector of 36 elements. The vector is then normalized according the L1 norm.

$$\text{Horizontal Kernal Filter} = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$\text{Vertical Kernal Filter} = \begin{array}{|c|} \hline -1 \\ \hline 0 \\ \hline 1 \\ \hline \end{array}$$

Figure 4.4: Kernal filter

| | | | | | | | | | |
|-------|---|----|----|----|----|-----|-----|-----|-----|
| Value | | | | | | | | | |
| Bins | 0 | 20 | 40 | 60 | 80 | 100 | 120 | 140 | 160 |

Figure 4.5: Spacial orientation cells

This process is useful to make the histogram values independent of lighting.

4. **Classifier:** finally, the set of descriptors can be used to feed the classifier, such as Support Vector Machine (SVM) classifier.

4.3 YOLO

You Only Look Once (YOLO) is a real-time object detection algorithm based on Convolutional Neural Network (CNN). It is considered one of the fastest algorithm in terms of detection, since it is able to detect in real-time and at the same time, it can maintain a high accuracy. As the name suggests, it requires only one forward propagation through the neural network to make predictions on the entire image and according to [34], the algorithm can process images at 45 frames per second. Another significant characteristic of YOLO is the capability of understanding generalized object representations, making it a much more powerful tool compared to other algorithms.

As shown in Figure 4.7, the YOLO network consists of 24 convolutional layers followed by 2 fully connected layers, which allow the algorithm to refine the image by reducing the features space from preceding layers, and hence to obtain a more accurate output.

4.3.1 How YOLO works

The YOLO algorithm can be divided into four steps:



Figure 4.6: HOG steps structure

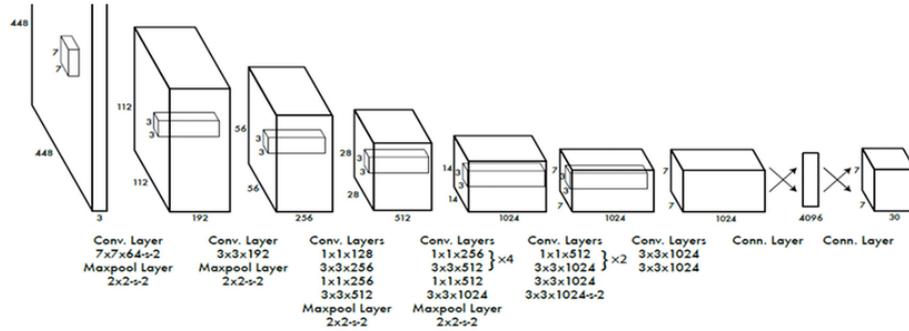


Figure 4.7: YOLO architecture [35]

1. Residual blocks
2. Bounding box regression
3. Intersection Over Union (IOU)
4. Final detection

Residual blocks

The input image is divided into grid cells of dimensions $S \times S$. Each grid is responsible of detecting the object whenever the center of that object falls into the grid cell.

Bounding box regression

Each grid cell predicts B bounding boxes, their confidence scores and the conditional class probabilities.

The confidence scores indicate how confident the network is that the selected bounding box contains an object.

Each bounding box is characterized by the following predictions: x , y , w , h and confidence prediction. The (x,y) coordinates represent the center of the box relative to the bounds of the grid cell, while w and h are respectively the width and the height of the box.

Intersection over union (IOU)

In order to select only the most fitting boxes, YOLO considers the Intersection over union (IOU) concept, which is a metric used to describe the overlapping of two boxes, greater is the region that overlaps, greater is the value of IOU.

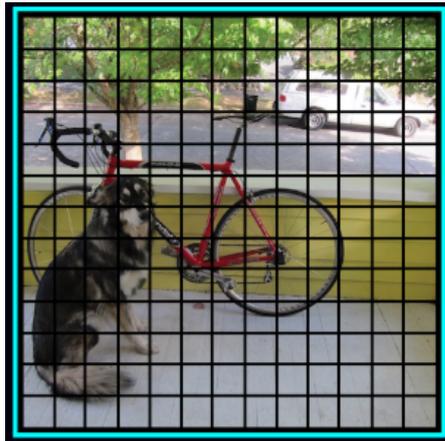


Figure 4.8: Step 1: Residual blocks

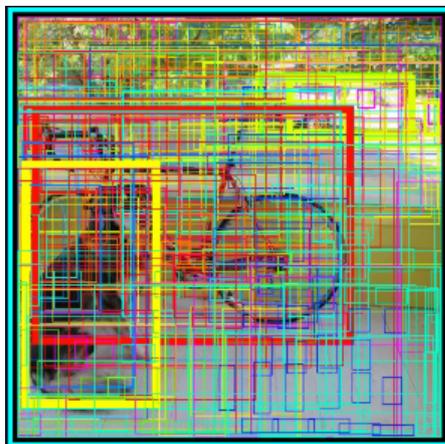


Figure 4.9: Step 2: Bounding box regression

First, the algorithm discharges the boxes that have the confidence score below a certain threshold. Then, it calculates the IOU between the actual bounding box and the predicted bounding box and if the IOU is greater than an arbitrary value, the prediction is considered good enough.

Final detection

However, this technique can end with more than one box identifying the same object. To solve this problem, the Non-Max Suppression algorithm can be used[36]: like in the previous technique, after reducing the number of boxes by considering their confidence score, the box with the highest probability is considered as output prediction and it is used to compute the IOU of all others. Finally, all the boxes with IOU greater than the threshold are discharged. In summary, all the boxes

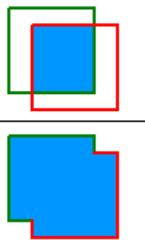
$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{area of overlap}}{\text{area of union}}$$


Figure 4.10: IOU computation

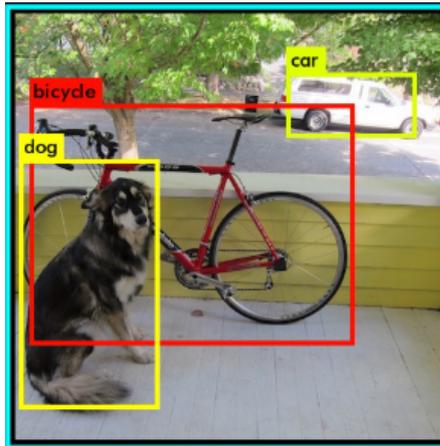


Figure 4.11: Step 4: Intersection Over Union result

that are almost overlapping the one with the highest probability are removed.

4.3.2 YOLOv2

YOLOv2 is an improved version of the YOLO algorithm. It is able to detect with more accuracy and with an extremely high speed. It is also known as "YOLO9000" because it was able to detect over 9000 categories of objects [37].

Compared to YOLO, YOLOv2 has a neural network of 19 convolutional layers compared to 24 with YOLO: the layers responsible for predicting the boundary boxes are removed and are substituted with anchor boxes.

Initially, YOLO selects randomly the boundary boxes and not always fits all the objects in the correct way. With YOLOv2, the boundary boxes are predicted with an offset constraint, the anchor boxes that include a specific class are constrained between an offset range.

In the following figure, it is reported the performance of YOLOv2 compared to other similar object detection algorithms:

| Method | data | mAP | aero | bike | bird | boat | bottle | bus | car | cat | chair | cow | table | dog | horse | mbike | person | plant | sheep | sofa | train | tv |
|-------------------|--------|------|------|------|------|------|--------|------|------|------|-------|------|-------|------|-------|-------|--------|-------|-------|------|-------|------|
| Fast R-CNN [5] | 07++12 | 68.4 | 82.3 | 78.4 | 70.8 | 52.3 | 38.7 | 77.8 | 71.6 | 89.3 | 44.2 | 73.0 | 55.0 | 87.5 | 80.5 | 80.8 | 72.0 | 35.1 | 68.3 | 65.7 | 80.4 | 64.2 |
| Faster R-CNN [15] | 07++12 | 70.4 | 84.9 | 79.8 | 74.3 | 53.9 | 49.8 | 77.5 | 75.9 | 88.5 | 45.6 | 77.1 | 55.3 | 86.9 | 81.7 | 80.9 | 79.6 | 40.1 | 72.6 | 60.9 | 81.2 | 61.5 |
| YOLO [14] | 07++12 | 57.9 | 77.0 | 67.2 | 57.7 | 38.3 | 22.7 | 68.3 | 55.9 | 81.4 | 36.2 | 60.8 | 48.5 | 77.2 | 72.3 | 71.3 | 63.5 | 28.9 | 52.2 | 54.8 | 73.9 | 50.8 |
| SSD300 [11] | 07++12 | 72.4 | 85.6 | 80.1 | 70.5 | 57.6 | 46.2 | 79.4 | 76.1 | 89.2 | 53.0 | 77.0 | 60.8 | 87.0 | 83.1 | 82.3 | 79.4 | 45.9 | 75.9 | 69.5 | 81.9 | 67.5 |
| SSD512 [11] | 07++12 | 74.9 | 87.4 | 82.3 | 75.8 | 59.0 | 52.6 | 81.7 | 81.5 | 90.0 | 55.4 | 79.0 | 59.8 | 88.4 | 84.3 | 84.7 | 83.3 | 50.2 | 78.0 | 66.3 | 86.3 | 72.0 |
| ResNet [6] | 07++12 | 73.8 | 86.5 | 81.6 | 77.2 | 58.0 | 51.0 | 78.6 | 76.6 | 93.2 | 48.6 | 80.4 | 59.0 | 92.1 | 85.3 | 84.8 | 80.7 | 48.1 | 77.3 | 66.5 | 84.7 | 65.6 |
| YOLOv2 544 | 07++12 | 73.4 | 86.3 | 82.0 | 74.8 | 59.2 | 51.8 | 79.8 | 76.5 | 90.6 | 52.1 | 78.2 | 58.5 | 89.3 | 82.5 | 83.4 | 81.3 | 49.1 | 77.2 | 62.4 | 83.8 | 68.7 |

Figure 4.12: YOLOv2 performance [28]

4.3.3 YOLOv3

Further improvement were archived with YOLOv3 [38]: it consists of 75 convolutional layers without fully connected layers which reduces greatly the size of the model. YOLOv3 prediction is based on the Feature Pyramid Networks (FPN) concept, which makes predictions at three different scales:

1. A prediction is made in the last feature map layer;
2. it considers the map from 2 previous layers and up-sample it by 2; then, it takes the feature map with higher resolution and merges it with the up-sampled one. This process allows the model to obtain more meaningful information. Then, more convolutional layers are added to the combined feature map to obtain the second prediction;
3. the second step is repeated to predict boxes for the final scale, where all the previous calculations and features are taken into account.

In terms of speed, YOLOv3 is able to outperform most of the object detection algorithms as shown in Figure 4.14:

| Type | Filters | Size | Output |
|---------------|---------------|-----------|-----------|
| Convolutional | 32 | 3 × 3 | 256 × 256 |
| Convolutional | 64 | 3 × 3 / 2 | 128 × 128 |
| 1x | Convolutional | 32 | 1 × 1 |
| | Convolutional | 64 | 3 × 3 |
| | Residual | | 128 × 128 |
| 2x | Convolutional | 128 | 3 × 3 / 2 |
| | Convolutional | 64 | 1 × 1 |
| | Convolutional | 128 | 3 × 3 |
| 8x | Residual | | 64 × 64 |
| | Convolutional | 256 | 3 × 3 / 2 |
| | Convolutional | 128 | 1 × 1 |
| 8x | Convolutional | 256 | 3 × 3 |
| | Residual | | 32 × 32 |
| | Convolutional | 512 | 3 × 3 / 2 |
| 4x | Convolutional | 256 | 1 × 1 |
| | Convolutional | 512 | 3 × 3 |
| | Residual | | 16 × 16 |
| 4x | Convolutional | 1024 | 3 × 3 / 2 |
| | Convolutional | 512 | 1 × 1 |
| | Convolutional | 1024 | 3 × 3 |
| | Residual | | 8 × 8 |
| | Avgpool | | Global |
| | Connected | | 1000 |
| Softmax | | | |

Figure 4.13: YOLOv3 layers description [38]

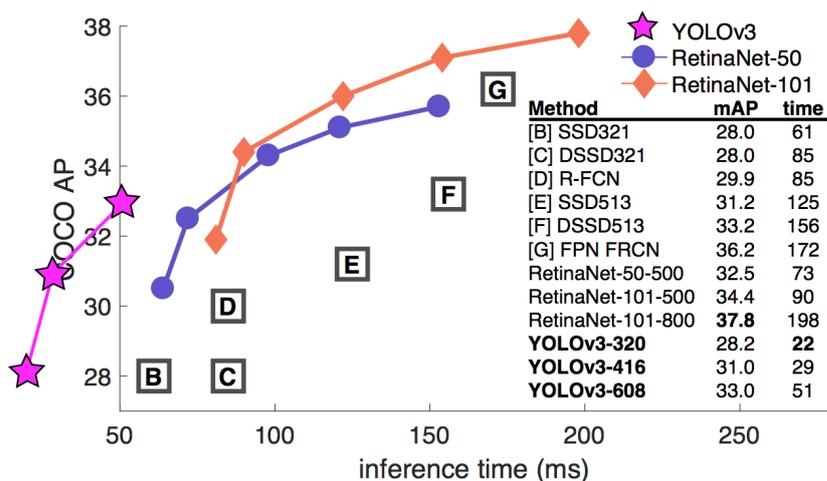


Figure 4.14: YOLOv3 performance [38]

Chapter 5

Implementation

The initial hardware implementation included a Microsoft Kinect 360 camera and a standard webcam camera connected to a single Raspberry Pi 3B+ board, which was controlled remotely using a personal computer. However, after some testing, it was discovered that the board was not able to provide an USB bandwidth large enough to process the depth and video data from the Microsoft Kinect camera, which caused packages loss and, consequently, low the performance of the detection algorithm. Therefore, the final decision was to connect only the webcam to the Raspberry Pi 3B+ and the Kinect camera to the personal computer.

First of all, Linux Ubuntu 20.04 was installed as operating system on both the Raspberry Pi 3B+ and the virtual machine present on the personal computer. Then, the following programs have been installed to perform the proposed implementation:

- freenect: driver for the Microsoft Kinect camera.
- OpenCV: the most famous and widely used open source library for computer vision compatible with different programming languages. In this case, Python was chosen.
- Thonny [39]: is the Python IDE chosen to edit the code; it is equipped with all the necessary functionality and does not require a large amount of memory to run. It was chosen in order to keep the Raspberry Pi memory as light as possible.
- ROS Noetic: which is the ROS version compatible with Ubuntu 20.04.

An abstract representation of the hardware connection is shown in Figure 5.1: Figure 5.2 is reported the flowchart of the algorithm proposed:

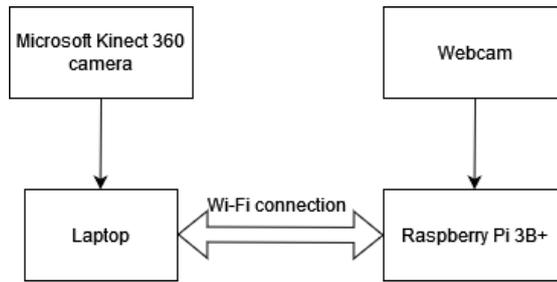


Figure 5.1: Hardware connection

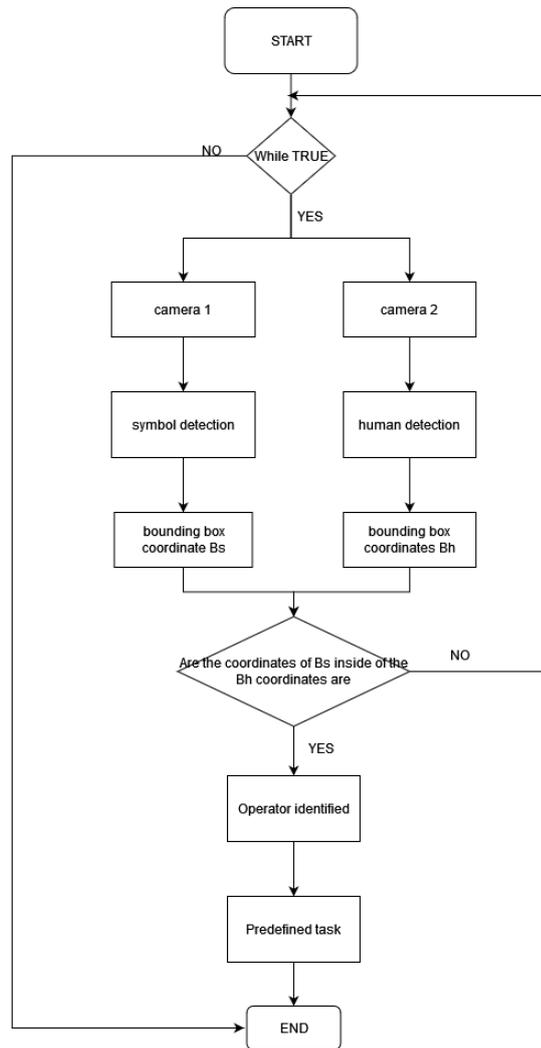


Figure 5.2: Flowchart of the algorithm proposed

5.1 Symbol detection

For the symbol recognition was proven that a simple standard webcam was sufficient, in particular the Wansview 1080p webcam was used.

Regarding the algorithm, two different methods are developed and tested: QR code recognition and color+shape recognition.

- **QR code recognition**

OpenCV by itself does not provide any dedicated modules that can be used to read QR codes. However, an already existing library is compatible and can be imported to perform the desired task, which is the ZBar [40].

ZBar is an open source software that is able to detect and decode bar codes and QR codes. It is compatible with C, C++, Python and Perl programming languages.

First, the required packages are imported:

```
1 import cv2 as cv
2 import numpy as np
3 from pyzbar.pyzbar import decode
```

Then, the camera starts recording the video and it is resized to 640x480 resolution to match the size of the images that are captured by the Microsoft Kinect camera:

```
1 cap = cv.VideoCapture(0)
2 cap.set(3, 640)
3 cap.set(4, 480)
4 cap.set(cv.CAP_PROP_BUFFERSIZE, 1)
```

The main function of the code is:

```
1 for barcode in decode(frame):
2     myData = barcode.data.decode('utf-8') #convert into
3     strings
4     print(myData)
5     peri = np.array([barcode.polygon], np.int32)
6     peri = peri.reshape((-1, 1, 2))
7     cv.polylines(frame, [peri], True, (0,255,0), 5)
8     rectangle = barcode.rect
9     cv.putText(frame, myData, (rectangle[0], rectangle[1]),
                cv.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,0), 2)
```

For each frame the function `barcode.data.decode()` finds and decodes the QR code. To check that the QR code is read correctly, the message decoded is printed.

Figure 5.3 and 5.4 show the positive detection of two different QR codes. It has been tested that the algorithm is able to detect without failing an image of dimension 7,5x7,5 cm in a range between 10 cm and 55 cm of distance from the camera. It can be assumed that the distance can be increased if we consider a larger image. Note also that the algorithm is able to detect correctly the image independently from the angle with which the image is shown.

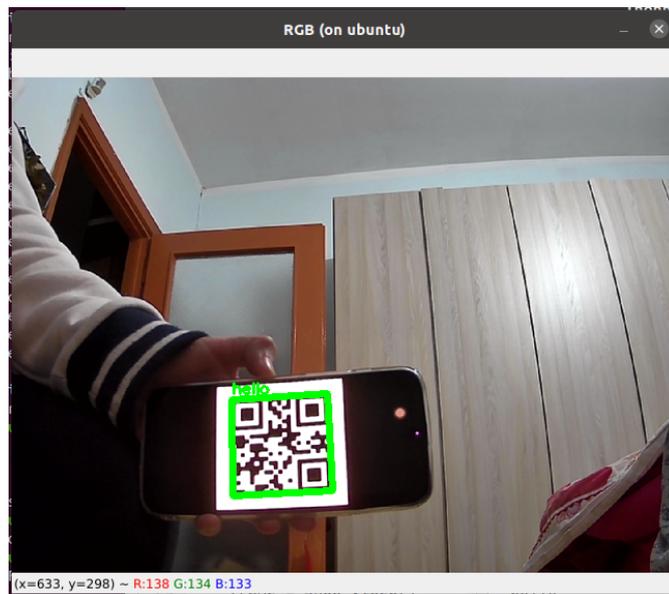


Figure 5.3: Example of QR code detection

The symbol recognition algorithm for detecting color and shape of the symbol is based on the theory of image contouring.

First, a trackbar is created to find the parameters of the desired colour. Since it is easier to determine the values in HSV scale compared to RGB scale, the trackbar is set up with three slider that change the hue, saturation and value of the colour.

```

1 cv.resizeWindow("Trackbar", 640, 240)
2 cv.createTrackbar("hue_min", "Trackbar", 28, 255, empty)
3 cv.createTrackbar("hue_max", "Trackbar", 34, 255, empty)
4 cv.createTrackbar("sat_min", "Trackbar", 143, 255, empty)

```

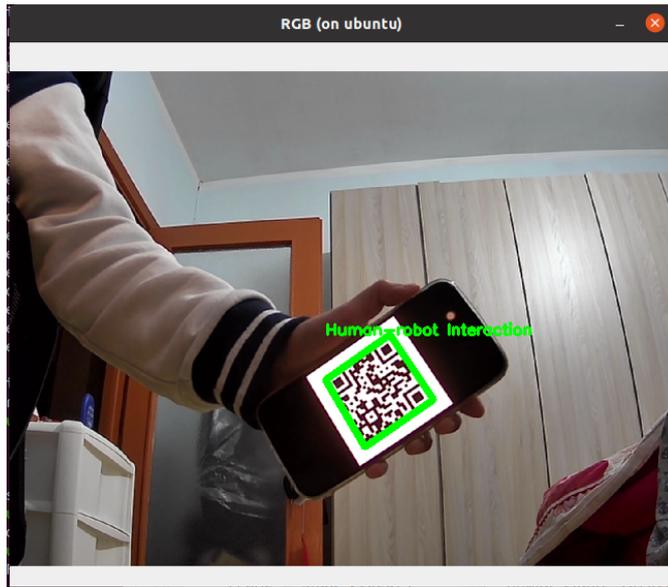


Figure 5.4: Example of QR code detection

```
5 cv.createTrackbar("sat_max", "Trackbar", 255, 255, empty)
6 cv.createTrackbar("val_min", "Trackbar", 195, 255, empty)
7 cv.createTrackbar("val_max", "Trackbar", 255, 255, empty)
```

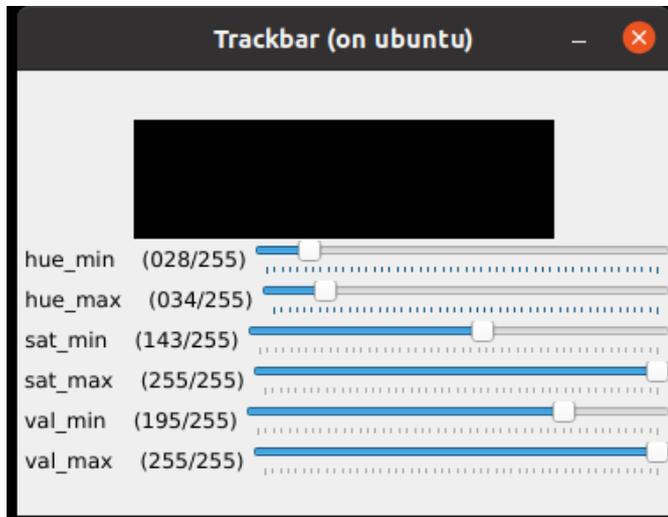


Figure 5.5: Trackbar to determine the color of the symbol

When the parameters are set up, the values are saved into two arrays, one for the low value of the range and one for the upper value of the range.

```
1 hue_min = cv.getTrackbarPos("hue_min", "Trackbar")
2 hue_max = cv.getTrackbarPos("hue_max", "Trackbar")
3 sat_min = cv.getTrackbarPos("sat_min", "Trackbar")
4 sat_max = cv.getTrackbarPos("sat_max", "Trackbar")
5 val_min = cv.getTrackbarPos("val_min", "Trackbar")
6 val_max = cv.getTrackbarPos("val_max", "Trackbar")
7 lower = np.array([hue_min, sat_min, val_min])
8 upper = np.array([hue_max, sat_max, val_max])
```

The parameters are needed in order to apply a *mask* on the image captured by the camera. The mask works as a filter with the purpose of isolating that specific colour from the rest of the image. In particular, all the pixels that have HSV values outside of the range determined with the trackbar are set to 0 (black), while the others are set to 255 (white) as shown in Figure 5.6, where the symbol considered is a yellow square.

```
1 mask = cv.inRange(hsv, lower, upper)
2 contours, hier = cv.findContours(mask, cv.RETR_EXTERNAL,
3 cv.CHAIN_APPROX_NONE)
```

The function *cv.findContours()* determines the points that represents the contour of the target and saves the coordinates of such points into the variable 'contours'.

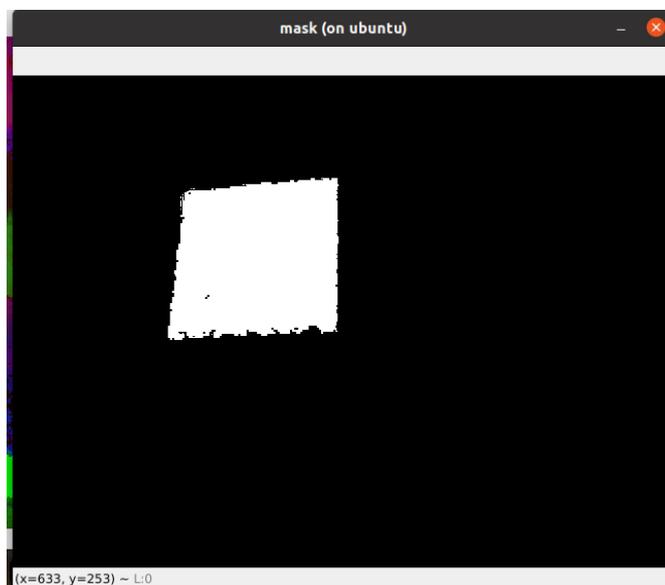


Figure 5.6: Image after the contouring process

Finally, the following 'for' cycle is the part of the code that determines if the symbol detected corresponds to one of the symbols assigned to an operator. First, in order to avoid possible false positives, a minimum of 400 points belonging to the area detected are considered to assure that the symbol detected is actually a possible target and not just noise. Then, the function *cv.approxPolyDP()* applies a contour approximation, which is an algorithm that reduces the number of points in a curve into a reduced set of points, according to the Ramer-Douglas-Peucker algorithm.

```
1   for c in contours:
2       area = cv.contourArea(c)
3       if area > 400:
4           peri = cv.arcLength(c, True)
5           approx = cv.approxPolyDP(c, 0.02*peri, True)
6           if len(approx) == 4:
7               x,y,w,h = cv.boundingRect(c)
8               ratio = w/float(h)
9               if (ratio >= 0.95 and ratio <= 1.05)
10                  cv.rectangle(frame, (x,y), (x+w, y+h),
11                               (0,255,0), thickness=2)
```

In this case, if the approximation has 4 points (vertices), the symbol can be either a square or a rectangle. For this reason, a second 'if' condition is introduced to check the aspect ratio of the bounding box: if it is approximately 1, the detected symbol is a square, otherwise it is a rectangle. The final result is shown in Figure 5.7.

So far the algorithm is able to detect only a yellow square, but it can be easily improved by substituting the trackbar with a set of predefined colours with the HSV parameters known and by adding more 'if' conditions to differentiate the possible shapes.

5.2 Human detection

For human detection, an optimal choice is the Microsoft Kinect 360 camera, which is a low cost camera able to provide both RGB and depth camera.

Two different object detection algorithms are considered and tested to decide which one is more appropriate for this work. The algorithms are:

- Histograms of Oriented Gradient (HOG)
- You Only Look Once (YOLO)

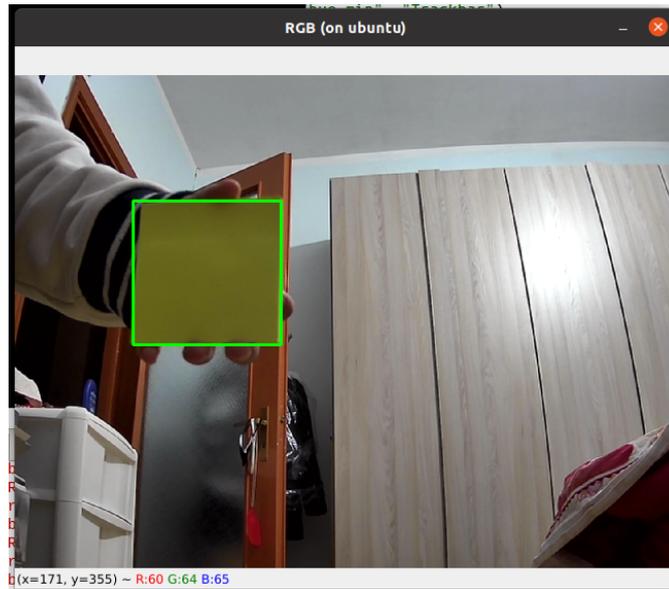


Figure 5.7: Symbol detection

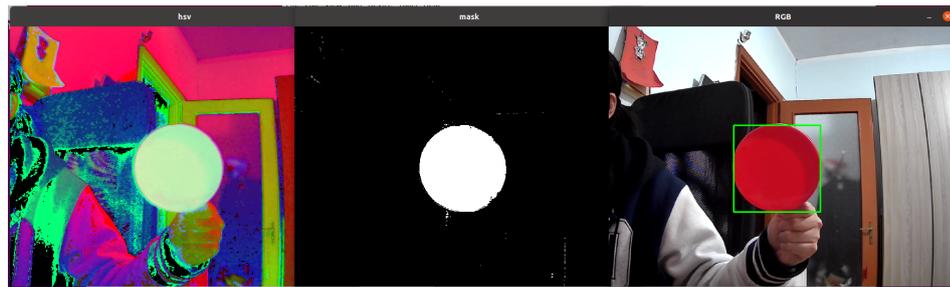


Figure 5.8: Different color/shape symbol: HSV scale, contouring process, symbol detection

HOG

Histograms of Oriented Gradient (HOG) is a real-time object detection which focuses the borders and shape of the image to extract meaning features from the background. With the increase of algorithms based on deep neural network, the performance of HOG appears to be outdated compared to other algorithms such as YOLO, CNN, SSD [41]. However, HOG is still able to provide a good performance with a positive number of detections and requires a lower computational power compared to the algorithms cited above. HOG has the advantage of been already integrated inside the OpenCV library, so there is no needs to import any external library.

First, the HOG detector and classifier are initialized with the following functions:

```
1 hog = cv.HOGDescriptor()  
2 hog.setSVMDetector(cv.HOGDescriptor_getDefaultPeopleDetector()  
   ())
```

As mentioned during the description of the process of the HOG algorithm, in order to improve the detection accuracy, it is important to resize the image:

```
1 frame = imutils.resize(frame, width=min(400, frame.shape[1]))
```

The detection is implemented with the following function:

```
1 (rects, weights) = hog.detectMultiScale(frame, winStride=(4,4),  
2   padding=(8,8), scale=1.05)
```

However, the classifier used in this case allows only the detection of the full-body. Hence, in order to obtain a positive detection the operator need to stay at a large distance that shows the full-body, which may affect the performance of the symbol detection algorithm.

YOLO

YOLO is a real-time object detection algorithm based on Convolutional Neural Network (CNN) and is considered the state-of-the-art in term of object detection thanks to its extreme accuracy.

In this work, three different version are considered:

- YOLOv3
- YOLOv3-tiny
- YOLOv4: however, this version is not compatible with the current OpenCV version (4.2) installed on the system.

In general, it is suggested to re-train the algorithm for the specific application, since the accuracy of the detection highly depends on the number of images in the dataset and its quality. However, it requires a lot of time and a huge computational time, which are unfortunately not currently available in this thesis.

Therefore, for this thesis, the algorithms are implemented considering the COCO dataset, which is one of the commonly used in object detection context.

It contains 80 different classes trained with more than 200'000 labeled images with bounding box and masks for each object, including the 'person' class that we are interested in.



Figure 5.9: Examples of object presents in the COCO dataset [42]

A few examples are shown in the following figures:



Figure 5.10: YOLO3: full body detection with a confidence value of 98%

5.3 Data fusion

Sensor fusion combines data coming from the sensors to obtain one output data with a more reliable and precise information.

First of all, a calibration process must be applied in order to align the data coming from the images of the two sensors. In general, camera calibration are based on patterns analysis to determine the 3D coordinates of the space.

The most common calibration pattern is the chessboard pattern shown in Figure 5.14, known also as checkerboard:



Figure 5.11: YOLO3: partial body detection with a confidence value of 96%



Figure 5.12: YOLO3-tiny: full body detection with a confidence value of 67%

First of all, for the camera calibration, all the corners of the checkerboard must be visible and detectable. A set of 3D points (x,y,z) and their corresponding pixel coordinated (u,v) are defined. In particular, since all the corner points lie on a plane, the coordinate z is chosen, arbitrarily, as 0 for every point, while the coordinates (x,y) can be easily obtained by taking one point as reference $(0,0)$ and defining the rest of the coordinates with respect to that reference point.

The definition of the coordinates process is repeated by moving the pattern board and keeping it in different angles and distances to obtain points with different perspective.

OpenCV provides the functions necessary for the camera calibration with the chessboard pattern.



Figure 5.13: YOLO3-tiny: no detection when the human figure is not fully visible



Figure 5.14: Checkerboard for camera calibration [43]



Figure 5.15: Some examples of images captured for the kinect camera calibration

The function:



Figure 5.16: Some examples of images captured for the webcam calibration

```
1   ret, corners = cv2.findChessboardCorners(image, patternSize,
      flags)
```

takes as input the image of the chessboard and the kind of pattern analysed and returns the information necessary to determine the position of the pattern board, in particular, it returns the coordinates of the corners.

Finally, the camera calibration is done through the function

```
1   retval, cameraMatrix, distCoeffs, rvecs, tvecs = cv2.
      calibrateCamera(objectPoints, imagePoints, imageSize)
```

which returns the camera matrix, distortion coefficients, rotation and translation vectors. The process is repeated for each camera.

After the calibration process, the stereo calibration is considered, which implementation is very similar to the individual calibration. The major difference is the usage of the method *stereoCalibrate()* to obtain the rotation and translation between the two camera and the Essential and Fundamental matrix. The Essential matrix describes the relative positioning of the two camera in the global coordinates, while the fundamental matrix is the matrix that describes the relationship between two images, in particular it expresses the epipolar geometry in stereo images, using pixels as unit of measure.

```
1   retS, new_mtxL, distL, new_mtxR, distR, Rot, Trns, Emat, Fmat
      = cv2.stereoCalibrate(objectPoints, imagePoints1,
      imagePoints2, cameraMatrix1, distCoeffs1, cameraMatrix2,
      distCoeffs2, imageSize, criteria_stereo, flags)
```

5.4 ROS

In order to communicate between the Raspberry Pi board and the personal computer, the two devices need to be connected to the same network and the PC is set to be the ROS Master.

```
1 export ROS_IP="<PC_IP>"  
2 export ROS_MASTER_URI="http://<PC_IP> :11311"
```

A subscriber-publisher system model for ROS architecture is created containing the following nodes: *symbol_detection*, *human_detection*, *operator_detection*. *symbol_detection*, *human_detection* nodes publish a messages in ROS Master containing the image data and the bounding box of the detected object, respectively the detection of the symbol and the detection of the human body. Then, the *operator_detection* with the outputs from the previous nodes performs the calibration of the cameras and the data fusion.

The final output checks if the bounding box of the symbol is contained in the bounding box of the human operator to confirm that the symbol detected is actually assigned to an operator to avoid possible false detection of other possible shapes in the environment.

A simple overview of the ROS architecture implemented is shown in figure 5.17

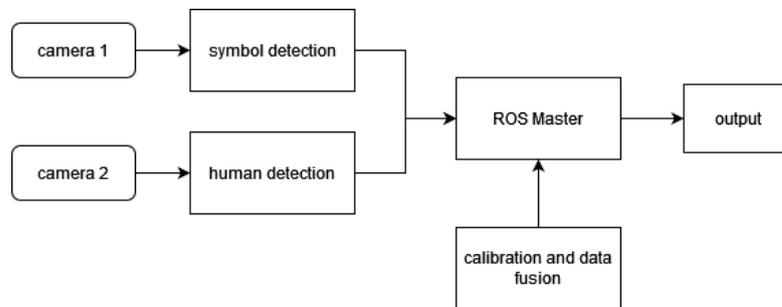


Figure 5.17: ROS architecture

Chapter 6

Simulation and testing

In this section there are presented the final results of combination of algorithms proposed and described in this work.

6.1 Histograms of Oriented Gradient

As explained previously in 5.2, the Histograms of Oriented Gradient algorithm provides a lower computational power at the expense of the quality of the detection. However, in order to have a positive detection, the full-body must be completely framed by the camera. For this reason, during the experiment the human operator had to stay at a distance of 2.5m circa from the camera.

6.2 YOLOv3

Two types of YOLO are tested in this project:

- YOLOv3: it is able to detect human with an average confidence of 96% full-body; 95% when only the upper body is visible; and a confidence that swings between 75% and 93% when only the legs are detected. However, due to the limited specs of the instruments used in this experiment, it shows a huge drop of frames.
- YOLOv3-tiny: there is a notable improvement in terms of fps, but the accuracy of the detection is considerably lower. It detects with a confidence of 54-72% when the full-body is visible and unlike the YOLOv3 version it is not able to positively detect the human when only a partial part of the body is visible.

From the results obtained from the testing, it can be concluded that:

- for the symbol detection, the QR code algorithm shows a better performance compared to the color + shape algorithm, in terms of both speed and accuracy. Indeed, the color + shape algorithm proposed in this work is more sensitive of the background noises which leads to possible false positive detections.
- for human detection, the YOLO shows a more robust detection compared to HOG algorithm, which demonstrate how Deep Learning (e.g. YOLO) algorithms are more robust respect to Machine Learning base algorithms (e.g. HOG).

Chapter 7

Conclusion

7.1 Conclusion

The main goal of the thesis was to develop a system able to detect and recognize human operators in order to assist them with specific tasks, using tools affordable for small and medium-size enterprises and for applications in industrial environments where high precision sensors are not necessary required.

Different combinations of algorithms and sensors were analyzed in order to obtain the optimal solution for the problem presented in this work.

The proposed solution makes use of two cameras in order to obtain a more robust and accurate detection. The operator is identified through a double detection: detection of the human body and detection of the symbol associated to the operator. This work propose two possible solution for the human detection based on machine learning and deep learning: Histograms of Oriented Gradient (HOG) and You Only Look Once (YOLO).

The obtained results demonstrate that proposed implementation satisfies the requests of the thesis, in particular the combination of QR code + YOLO algorithm for detection displays the best outcome. However, due to the limitations of the current hardware setup, the implementation proposed shows a limited number of FPS which is important for real-time applications.

7.2 Future work

This thesis can be considered as a starting point for a more complex system. Future works to improve this project can be:

- pair the human detection with depth camera in order to understand and exploit the distance between the robot and the operator.

- re-train the YOLO algorithm for human detection only.
- try different cameras in order to improve the accuracy of the detection in darker environments, such as camera based on infrared cameras.

Bibliography

- [1] International Federation of Robotics. «IFR presents World Robotics Report 2020». In: (2020). URL: <https://ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe> (cit. on p. 1).
- [2] Cruz C., Sucar L., and Morales E. «Real-Time face recognition for human-robot interaction». In: Oct. 2008, pp. 1–6. DOI: 10.1109/AFGR.2008.4813386 (cit. on p. 2).
- [3] Bellotto N. and Hu H. «Multisensor integration for human-robot interaction». In: *The IEEE Journal of Intelligent Cybernetic Systems* 1 (Jan. 2005) (cit. on p. 2).
- [4] H. C. Fang, S. K. Ong, and A. Y. C. Nee. «A novel augmented reality-based interface for robot path planning». English. In: *International Journal on Interactive Design and Manufacturing* 8.1 (2014), pp. 33–42. URL: www.scopus.com (cit. on p. 2).
- [5] A. Hentout, M. Aouache, A. Maoudj, and I. Akli. «Human-robot interaction in industrial collaborative robotics: a literature review of the decade 2008–2017». English. In: *Advanced Robotics* 33.15-16 (2019), pp. 764–799 (cit. on p. 2).
- [6] W. Wang, R. Li, Z. M. Diekel, Y. Chen, Z. Zhang, and Y. Jia. «Controlling object hand-over in human-robot collaboration via natural wearable sensing». English. In: *IEEE Transactions on Human-Machine Systems* 49.1 (2019), pp. 59–71 (cit. on p. 4).
- [7] M. Coban and G. Gelen. «Realization of human-robot collaboration in hybrid assembly systems by using wearable technology». English. In: *2018 6th International Conference on Control Engineering and Information Technology, CEIT 2018*. Cited By :2. 2018 (cit. on p. 4).
- [8] O. Mazhar, B. Navarro, S. Ramdani, R. Passama, and A. Cherubini. «A real-time human-robot interaction framework with robust background invariant hand gesture detection». English. In: *Robotics and Computer-Integrated Manufacturing* 60 (2019). Cited By :33, pp. 34–48 (cit. on p. 4).

- [9] Z. Cao, G. Hidalgo, T. Simon, S. - Wei, and Y. Sheikh. «OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields». English. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.1 (2021), pp. 172–186 (cit. on p. 4).
- [10] K. - Song, P. - Lu, and S. - Song. «Human-Robot Interaction Design Based on Specific Person Finding and Localization of a Mobile Robot». English. In: *2019 International Automatic Control Conference, CACS 2019*. 2019 (cit. on p. 5).
- [11] N. Dalal and B. Triggs. «Histograms of oriented gradients for human detection». English. In: *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*. Vol. I. 2005, pp. 886–893 (cit. on p. 5).
- [12] Z. Jian, F. Zhu, and L. Tang. «Research on Human Body Recognition and Position Measurement Based on AdaBoost and RGB-D». English. In: *Chinese Control Conference, CCC*. Vol. 2020-July. Cited By :1. 2020, pp. 5184–5189. URL: www.scopus.com (cit. on p. 5).
- [13] H. Bozorgi, X. T. Truong, H. M. La, and T. D. Ngo. «2D Laser and 3D Camera Data Integration and Filtering for Human Trajectory Tracking». English. In: *2021 IEEE/SICE International Symposium on System Integration, SII 2021*. 2021, pp. 634–639 (cit. on p. 5).
- [14] G. Yan, J. Shi, Z. Yu, and J. Wang. «Human tracking based on vision and laser sensor». English. In: *Proceedings of 2017 IEEE International Conference on Unmanned Systems, ICUS 2017*. Vol. 2018-January. Cited By :1. 2018, pp. 177–181 (cit. on p. 5).
- [15] S. Matsubara, A. Honda, Y. Ji, and K. Umeda. «Three-dimensional Human Tracking of a Mobile Robot by Fusion of Tracking Results of Two Cameras». English. In: *2020 21st International Conference on Research and Education in Mechatronics, REM 2020*. 2020 (cit. on p. 5).
- [16] M. Wang, D. Su, L. Shi, Y. Liu, and J. V. Miro. «Real-time 3D human tracking for mobile robots with multisensors». English. In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2017, pp. 5081–5087. URL: www.scopus.com (cit. on p. 5).
- [17] Y. - Wang, T. - Wang, J. - Yen, and F. - Wang. «Dynamic human object recognition by combining color and depth information with a clothing image histogram». English. In: *International Journal of Advanced Robotic Systems* 16.1 (2019). URL: www.scopus.com (cit. on p. 5).
- [18] IBM. *What is Computer Vision?* URL: <https://www.ibm.com/topics/computer-vision> (cit. on p. 5).

- [19] A. L. Samuel. «Some Studies in Machine Learning Using the Game of Checkers». In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229. DOI: 10.1147/rd.33.0210 (cit. on p. 6).
- [20] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis. «Deep Learning for Computer Vision: A Brief Review». English. In: *Computational Intelligence and Neuroscience* 2018 (2018) (cit. on p. 6).
- [21] *The difference between Artificial Intelligence, Machine Learning and Deep Learning*. URL: <https://datacatchup.com/artificial-intelligence-machine-learning-and-deep-learning/> (cit. on p. 6).
- [22] Josh Patterson and Adam Gibson. *Deep Learning: A Practitioner’s Approach*. O’Reilly, 2017. ISBN: 978-1-4919-1425-0 (cit. on pp. 7, 8).
- [23] *Faster Image Classification using Tensorflow’s Graph mode*. URL: <https://medium.com/artificialis/faster-image-classification-using-tensorflows-graph-mode-67098154808b> (cit. on p. 8).
- [24] J.C. Russ. *The Image Processing Handbook*. CRC Press, 2007 (cit. on p. 10).
- [25] *Grayscale*. *Wikipedia, the free encyclopedia*. URL: <https://en.wikipedia.org/wiki/Grayscale> (cit. on p. 10).
- [26] N.V. Kalyankar Salem Saleh Al-amri and Khamitkar S.D. «Image Segmentation by Using Thershod Techniques». English. In: *JOURNAL OF COMPUTING* 2.5 (2010) (cit. on p. 11).
- [27] *Raspberry Pi 3 Model B+*. URL: <https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/> (cit. on p. 15).
- [28] *ROS - Robot Operating System*. URL: <https://www.ros.org/> (cit. on p. 16).
- [29] *CLEARPATH Robotics, ROS Tutorials*. URL: <https://www.clearpathrobotics.com/assets/guides/melodic/ros/Intro> (cit. on p. 16).
- [30] *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. URL: <https://jamielang.github.io/2017/05/28/faster-r-cnn/> (cit. on p. 18).
- [31] *Object Detection for Dummies Part 1: Gradient Vector, HOG, and SS*. URL: <https://lilianweng.github.io/posts/2017-10-29-object-recognition-part-1/> (cit. on p. 19).
- [32] N. Dalal and B. Triggs. «Histograms of oriented gradients for human detection». English. In: *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*. Vol. I. 2005, pp. 886–893. URL: www.scopus.com (cit. on p. 19).

- [33] M. Kachouane, S. Sahki, M. Lakrouf, and N. Ouadah. «HOG based fast human detection». English. In: *Proceedings of the International Conference on Microelectronics, ICM*. 2012. URL: www.scopus.com (cit. on p. 19).
- [34] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. «You only look once: Unified, real-time object detection». English. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2016-December. 2016, pp. 779–788. URL: www.scopus.com (cit. on p. 20).
- [35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. «You Only Look Once: Unified, Real-Time Object Detection». In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2016, pp. 779–788 (cit. on p. 21).
- [36] Lakshini Kuganandamurthy Uplie H.D.I. «Real-Time Object Detection using YOLO: A review». In: 2021 (cit. on p. 22).
- [37] J. Redmon and A. Farhadi. «YOLO9000: Better, faster, stronger». English. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. Vol. 2017-January. 2017, pp. 6517–6525. URL: www.scopus.com (cit. on p. 23).
- [38] Ali Farhadi Joseph Redmon. «YOLOv3: An Incremental Improvement». In: 2018 (cit. on pp. 24, 25).
- [39] Thonny. *Thonny, Python IDE for beginners*. URL: <https://thonny.org/> (cit. on p. 26).
- [40] SourceForge. *ZBar bar code reader*. URL: <http://zbar.sourceforge.net/index.html> (cit. on p. 28).
- [41] H. zota and M. Dhande. «A Comparative Study of Widely Used Image Detection Algorithms». In: *International Research Journal of Engineering and Technology (IRJET)* 7 (2020) (cit. on p. 33).
- [42] *COCO Dataset*. URL: <https://cocodataset.org/#home> (cit. on p. 35).
- [43] *OpenCV, Camera Calibration*. URL: https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html (cit. on p. 37).